ESP32-S2

Technical Reference Manual

Version 1.0

Espressif Systems

Copyright © 2021

About This Manual

The **ESP32-S2 Technical Reference Manual** is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32-S2 memory and peripherals.

For pin definition, electrical characteristics and package information, please see *ESP32-S2 Datasheet*.

Document Updates

Please always refer to the latest version on https://www.espressif.com/en/support/download/documents.

Revision History

For any changes to this document over time, please refer to the last page.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Contents

1 ULP Coprocessor (ULP) 27
1.1 Overview 27
1.2 Features 27
1.3 Programming Workflow 29
1.4 ULP Coprocessor Workflow 29
1.5 ULP-FSM 31
1.5.1 Features 31
1.5.2 Instruction Set 31
ALU - Perform Arithmetic and Logic Operations 32
ST – Store Data in Memory 35
LD – Load Data from Memory 37
JUMP – Jump to an Absolute Address 38
JUMPR – Jump to a Relative Offset (Conditional upon R0) 39
JUMPS – Jump to a Relative Address (Conditional upon Stage Count Register) 39
HALT – End the Program 41
WAKE – Wake up the Chip 41
WAIT – Wait for a Number of Cycles 41
TSENS – Take Measurement with Temperature Sensor 42
ADC – Take Measurement with ADC 42
REG_RD – Read from Peripheral Register 43
REG_WR – Write to Peripheral Register 44
1.6 ULP-RISC-V 44
1.6.1 Features 44
1.6.2 Multiplier and Divider 44
1.6.3 ULP-RISC-V Interrupts 45
1.7 RTC I2C Controller 46
1.7.1 Connecting RTC I2C Signals 46
1.7.2 Configuring RTC I2C 46
1.7.3 Using RTC I2C 46
Instruction Format 46
I2C_RD - I2C Read Workflow 46
I2C_WR - I2C Write Workflow 47
Detecting Error Conditions 48
1.7.4 RTC I2C Interrupts 48
1.8 Base Address 49
1.8.1 ULP Coprocessor Base Address 49
1.8.2 RTC I2C Base Address 49
1.9 Register Summary 49
1.9.1 ULP (ALWAYS_ON) Register Summary 49
1.9.2 ULP (RTC_PERI) Register Summary 50
1.9.3 RTC I2C (RTC_PERI) Register Summary 50
1.9.4 RTC I2C (I2C) Register Summary 50

1.10 Registers 51
1.10.1 ULP (ALWAYS_ON) Registers 52
1.10.2 ULP (RTC_PERI) Registers 55
1.10.3 RTC I2C (RTC_PERI) Registers 58
1.10.4 RTC I2C (I2C) Registers 60

2 DMA Controller (DMA) 74
2.1 Overview 74
2.2 Features 74
2.3 Functional Description 75
2.3.1 DMA Engine Architecture 75
2.3.2 Linked List 75
2.3.3 Enabling DMA 76
2.3.4 Linked List reading process 77
2.3.5 EOF 77
2.3.6 Internal DMA 77
2.3.7 EDMA 78
2.3.7.1 Accessing Internal RAM 78
2.3.8 Accessing External RAM 78
2.4 Copy DMA Controller 79
2.5 UART DMA (UDMA) Controller 79
2.6 SPI DMA Controller 80
2.7 I2S DMA Controller 81
2.8 Crypto DMA 82
2.9 Copy DMA Interrupts 82
2.10 Crypto DMA Interrupts 82
2.11 Base Address 83
2.12 Register Summary 83
2.13 Registers 86

3 System and Memory 110
3.1 Overview 110
3.2 Features 110
3.3 Functional Description 111
3.3.1 Address Mapping 111
3.3.2 Internal Memory 112
3.3.2.1 Internal ROM 0 113
3.3.2.2 Internal ROM 1 113
3.3.2.3 Internal SRAM 0 113
3.3.2.4 Internal SRAM 1 113
3.3.2.5 RTC FAST Memory 114
3.3.2.6 RTC SLOW Memory 114
3.3.3 External Memory 114
3.3.3.1 External Memory Address Mapping 114
3.3.3.2 Cache 115
3.3.3.3 Cache Operations 115

3.3.4 DMA 116
3.3.5 Modules / Peripherals 116
3.3.5.1 Naming Conventions for Peripheral Buses 116
3.3.5.2 Differences Between PeriBus1 and PeriBus2 117
3.3.5.3 Module / Peripheral Address Mapping 117
3.3.5.4 Addresses with Restricted Access from PeriBus1 118

4 eFuse Controller (eFuse) 120
4.1 Overview 120
4.2 Features 120
4.3 Functional Description 120
4.3.1 Structure 120
4.3.1.1 EFUSE_WR_DIS 124
4.3.1.2 EFUSE_RD_DIS 124
4.3.1.3 Data Storage 124
4.3.2 Software Programming of Parameters 125
4.3.3 Software Reading of Parameters 126
4.3.4 Timing 128
4.3.4.1 eFuse-Programming Timing 128
4.3.4.2 eFuse VDDQ Timing Setting 129
4.3.4.3 eFuse-Read Timing 129
4.3.5 The Use of Parameters by Hardware Modules 130
4.3.6 Interrupts 130
4.4 Base Address 130
4.5 Register Summary 130
4.6 Registers 134

5 IO MUX and GPIO Matrix (GPIO, IO_MUX) 156
5.1 Overview 156
5.2 Peripheral Input via GPIO Matrix 157
5.2.1 Overview 157
5.2.2 Synchronization 157
5.2.3 Functional Description 158
5.2.4 Simple GPIO Input 159
5.3 Peripheral Output via GPIO Matrix 159
5.3.1 Overview 159
5.3.2 Functional Description 159
5.3.3 Simple GPIO Output 160
5.3.4 Sigma Delta Modulated Output 161
5.3.4.1 Functional Description 161
5.3.4.2 SDM Configuration 161
5.4 Dedicated GPIO 162
5.4.1 Overview 162
5.4.2 Features 162
5.4.3 Functional Description 162
5.4.3.1 Accessing GPIO via Registers 163

5.4.3.2 Accessing GPIO with CPU 163
5.5 Direct I/O via IO MUX 164
5.5.1 Overview 164
5.5.2 Functional Description 164
5.6 RTC IO MUX for Low Power and Analog I/O 165
5.6.1 Overview 165
5.6.2 Functional Description 165
5.7 Pin Functions in Light-sleep 165
5.8 Pad Hold Feature 165
5.9 I/O Pad Power Supplies 166
5.9.1 Power Supply Management 166
5.10 Peripheral Signal List 166
5.11 IO MUX Pad List 170
5.12 RTC IO MUX Pin List 172
5.13 Base Address 173
5.14 Register Summary 173
5.14.1 GPIO Matrix Register Summary 173
5.14.2 IO MUX Register Summary 174
5.14.3 Sigma Delta Modulated Output Register Summary 175
5.14.4 Dedicated GPIO Register Summary 176
5.14.5 RTC IO MUX Register Summary 176
5.15 Registers 177
5.15.1 GPIO Matrix Registers 177
5.15.2 IO MUX Registers 189
5.15.3 Sigma Delta Modulated Output Registers 191
5.15.4 Dedicated GPIO Registers 192
5.15.5 RTC IO MUX Registers 201

6 Reset and Clock 215
6.1 Reset 215
6.1.1 Overview 215
6.1.2 Reset Source 215
6.2 Clock 216
6.2.1 Overview 216
6.2.2 Clock Source 217
6.2.3 CPU Clock 218
6.2.4 Peripheral Clock 219
6.2.4.1 APB_CLK Source 219
6.2.4.2 REF_TICK Source 219
6.2.4.3 LEDC_PWM_CLK Source 220
6.2.4.4 APLL_SCLK Source 220
6.2.4.5 PLL_160M_CLK Source 220
6.2.4.6 Clock Source Considerations 220
6.2.5 Wi-Fi Clock 220
6.2.6 RTC Clock 221
6.2.7 Audio PLL Clock 221

7 Chip Boot Control (BOOTCTRL) 222
7.1 Overview 222
7.2 Boot Mode 222
7.3 ROM Code Printing to UART 223
7.4 VDD_SPI Voltage 223

8 Interrupt Matrix (INTERRUPT) 224
8.1 Overview 224
8.2 Features 224
8.3 Functional Description 224
8.3.1 Peripheral Interrupt Sources 224
8.3.2 CPU Interrupts 228
8.3.3 Allocate Peripheral Interrupt Source to CPU Interrupt 229
8.3.3.1 Allocate one peripheral interrupt source Source_X to CPU 229
8.3.3.2 Allocate multiple peripheral interrupt sources Source_X*n* to CPU 229
8.3.3.3 Disable CPU peripheral interrupt source Source_X 229
8.3.4 Disable CPU NMI Interrupt Sources 230
8.3.5 Query Current Interrupt Status of Peripheral Interrupt Source 230
8.4 Base Address 230
8.5 Register Summary 230
8.6 Registers 234

9 Low-Power Management (RTC_CNTL) 263
9.1 Introduction 263
9.2 Features 263
9.3 Functional Description 263
9.3.1 Power Management Unit 265
9.3.2 Low-Power Clocks 267
9.3.3 Timers 268
9.3.4 Regulators 269
9.3.4.1 Digital System Voltage Regulator 269
9.3.4.2 Low-power Voltage Regulator 270
9.3.4.3 Flash Voltage Regulator 271
9.3.4.4 Brownout Detector 272
9.4 Power Modes Management 273
9.4.1 Power Domain 273
9.4.2 RTC States 273
9.4.3 Pre-defined Power Modes 275
9.4.4 Wakeup Source 275
9.5 RTC Boot 277
9.6 Base Address 278
9.7 Register Summary 278
9.8 Registers 280

10 System Timer (SYSTIMER) 317
10.1 Overview 317

10.2 Main Features 317
10.3 Clock Source Selection 317
10.4 Functional Description 317
10.4.1 Read System Timer Value 318
10.4.2 Configure a Time-Delay Alarm 318
10.4.3 Configure Periodic Alarms 318
10.4.4 Update after Deep-sleep and Light-sleep 318
10.5 Base Address 318
10.6 Register Summary 319
10.7 Registers 320

11 Timer Group (TIMG) 328
11.1 Overview 328
11.2 Functional Description 329
11.2.1 16-bit Prescaler and Clock Selection 329
11.2.2 64-bit Time-based Counter 329
11.2.3 Alarm Generation 329
11.2.4 Timer Reload 329
11.2.5 Interrupts 330
11.3 Configuration and Usage 331
11.3.1 Timer as a Simple Clock 331
11.3.2 Timer as One-shot Alarm 331
11.3.3 Timer as Periodic Alarm 332
11.4 Base Address 332
11.5 Register Summary 332
11.6 Registers 334

12 Watchdog Timers (WDT) 347
12.1 Overview 347
12.2 Features 347
12.3 Functional Description 347
12.3.1 Clock Source and 32-Bit Counter 347
12.3.2 Stages and Timeout Actions 348
12.3.3 Write Protection 348
12.3.4 Flash Boot Protection 349
12.4 Super Watchdog 349
12.4.1 Features 349
12.4.2 Super Watchdog Controller 349
12.4.2.1 Structure 350
12.4.2.2 Workflow 350
12.5 Registers 350

13 XTAL32K Watchdog Timer (XTWDT) 351
13.1 Overview 351
13.2 Features 351
13.2.1 XTAL32K Watchdog Timer Interrupts and Wake-up 351

13.2.2 BACKUP32K_CLK 351
13.3 Functional Description 351
13.3.1 Workflow 351
13.3.2 Configuring the Divisor of BACKUP32K_CLK 352

14 Permission Control (PMS) 353
14.1 Overview 353
14.2 Features 353
14.3 Functional Description 353
14.3.1 Internal Memory Permission Controls 353
14.3.1.1 Permission Control for the Instruction Bus (IBUS) 354
14.3.1.2 Permission Control for the Data Bus (DBUS0) 356
14.3.1.3 Permission Control for On-chip DMA 357
14.3.1.4 Permission Control for PeriBus1 358
14.3.1.5 Permission Control for PeriBus2 360
14.3.1.6 Permission Control for Cache 361
14.3.1.7 Permission Control of Other Types of Internal Memory 361
14.3.2 External Memory Permission Control 362
14.3.2.1 Cache MMU 362
14.3.2.2 External Memory Permission Controls 362
14.3.3 Non-Aligned Access Permission Control 363
14.4 Base Address 364
14.5 Register Summary 364
14.6 Registers 366

15 System Registers (SYSTEM) 394
15.1 Overview 394
15.2 Features 394
15.3 Function Description 394
15.3.1 System and Memory Registers 394
15.3.2 Reset and Clock Registers 396
15.3.3 Interrupt Matrix Registers 396
15.3.4 JTAG Software Enable Registers 396
15.3.5 Low-power Management Registers 396
15.3.6 Peripheral Clock Gating and Reset Registers 397
15.4 Base Address 399
15.5 Register Summary 399
15.6 Registers 400

16 SHA Accelerator (SHA) 416
16.1 Introduction 416
16.2 Features 416
16.3 Working Modes 416
16.4 Function Description 417
16.4.1 Preprocessing 417
16.4.1.1 Padding the Message 417

16.4.1.2 Parsing the Message 418
16.4.1.3 Initial Hash Value 419
16.4.2 Hash Computation Process 420
16.4.2.1 Typical SHA Process 420
16.4.2.2 DMA-SHA Process 422
16.4.3 Message Digest 423
16.4.4 Interrupt 424
16.5 Base Address 425
16.6 Register Summary 425
16.7 Registers 427

17 AES Accelerator (AES) 431
17.1 Introduction 431
17.2 Features 431
17.3 Working Modes 431
17.4 Typical AES Working Mode 432
17.4.1 Key, Plaintext, and Ciphertext 432
17.4.2 Endianness 433
17.4.3 Operation Process 437
17.5 DMA-AES Working Mode 438
17.5.1 Key, Plaintext, and Cipertext 438
17.5.2 Endianness 439
17.5.3 Standard Incrementing Function 440
17.5.4 Block Number 440
17.5.5 Initialization Vector 440
17.5.6 Block Operation Process 440
17.5.7 GCM Operation Process 441
17.6 GCM Algorithm 443
17.6.1 Hash Subkey 444
17.6.2 J_0 444
17.6.3 Authenticated Tag 444
17.6.4 AAD Block Number 444
17.6.5 Remainder Bit Number 445
17.7 Base Address 445
17.8 Memory Summary 445
17.9 Register Summary 446
17.10 Registers 447

18 RSA Accelerator (RSA) 452
18.1 Introduction 452
18.2 Features 452
18.3 Functional Description 452
18.3.1 Large Number Modular Exponentiation 452
18.3.2 Large Number Modular Multiplication 454
18.3.3 Large Number Multiplication 454
18.3.4 Acceleration Options 455

18.4 Base Address 457
18.5 Memory Summary 457
18.6 Register Summary 457
18.7 Registers 458

19 HMAC Accelerator (HMAC) 462
19.1 Overview 462
19.2 Main Features 462
19.3 Functional Description 462
19.3.1 Upstream Mode 462
19.3.2 Downstream JTAG Enable Mode 463
19.3.3 Downstream Digital Signature Mode 463
19.3.4 HMAC eFuse Configuration 464
19.3.5 HMAC Process (Detailed) 464
19.4 HMAC Algorithm Details 466
19.4.1 Padding Bits 466
19.4.2 HMAC Algorithm Structure 467
19.5 Base Address 467
19.6 Register Summary 468
19.7 Registers 469

20 Digital Signature (DS) 475
20.1 Overview 475
20.2 Features 475
20.3 Functional Description 475
20.3.1 Overview 475
20.3.2 Private Key Operands 475
20.3.3 Conventions 476
20.3.4 Software Storage of Private Key Data 476
20.3.5 DS Operation at the Hardware Level 477
20.3.6 DS Operation at the Software Level 478
20.4 Base Address 479
20.5 Memory Blocks 479
20.6 Register Summary 479
20.7 Registers 480

21 External Memory Encryption and Decryption (XTS_AES) 482
21.1 Overview 482
21.2 Features 482
21.3 Functional Description 482
21.3.1 XTS Algorithm 483
21.3.2 Key 483
21.3.3 Target Memory Space 484
21.3.4 Data Padding 484
21.3.5 Manual Encryption Block 485
21.3.6 Auto Encryption Block 486

21.3.7 Auto Decryption Block 486
21.4 Base Address 487
21.5 Register Summary 487
21.6 Registers 488

22 Random Number Generator (RNG) 492
22.1 Introduction 492
22.2 Features 492
22.3 Functional Description 492
22.4 Programming Procedure 493
22.5 Base Address 493
22.6 Register Summary 493
22.7 Register 494

23 UART Controller (UART) 495
23.1 Overview 495
23.2 Features 495
23.3 Functional Description 495
23.3.1 UART Introduction 495
23.3.2 UART Structure 496
23.3.3 UART RAM 497
23.3.4 Baud Rate Generation and Detection 497
23.3.4.1 Baud Rate Generation 498
23.3.4.2 Baud Rate Detection 498
23.3.5 UART Data Frame 499
23.3.6 RS485 500
23.3.6.1 Driver Control 500
23.3.6.2 Turnaround Delay 501
23.3.6.3 Bus Snooping 501
23.3.7 IrDA 501
23.3.8 Wake-up 502
23.3.9 Flow Control 502
23.3.9.1 Hardware Flow Control 503
23.3.9.2 Software Flow Control 504
23.3.10 UDMA 504
23.3.11 UART Interrupts 504
23.3.12 UHCI Interrupts 505
23.4 Base Address 506
23.5 Register Summary 506
23.6 Registers 509

24 SPI Controller (SPI) 555
24.1 Overview 555
24.2 Features 556
24.2.1 GP-SPI2 Features 556
24.2.1.1 Functioning as a Master 556

24.2.1.2 Functioning as a Slave 556
24.2.1.3 Functioning as a Master or a Slave 557
24.2.2 GP-SPI3 Features 557
24.2.2.1 Functioning as a Master 557
24.2.2.2 Functioning as a Slave 557
24.2.2.3 Functioning as a Master or a Slave 558
24.2.3 SPI Interrupt Features 558
24.3 GP-SPI Interfaces 558
24.4 GP-SPI2 Works as a Master 560
24.4.1 State Machine 560
24.4.2 Register Configuration Rules for State Control 561
24.4.3 Full-Duplex Communication (1-bit Mode Only) 564
24.4.4 Half-Duplex Communication (1/2/4/8-bit Mode) 565
24.4.5 Access Flash and External RAM in Master Half-Duplex Mode 566
24.4.6 Access 8-bit I8080/MT6800 LCD in Master Half-Duplex Mode 566
24.4.7 DMA Controlled Segmented-Configure-Transfer 568
24.4.8 Access Parallel 8-bit RGB Mode LCD via Segmented-Configure-Transfer 571
24.4.9 CS Setup Time and Hold Time Control 573
24.5 GP-SPI2 Works as a Slave 574
24.5.1 Communication Formats 574
24.5.2 Supported CMD Values in Half-Duplex Communication 575
24.5.3 GP-SPI2 Slave Mode Single Transfer 577
24.5.4 GP-SPI2 Slave Mode Segmented-Transfer 578
24.6 Differences Between GP-SPI2 and GP-SPI3 579
24.7 CPU Controlled Data Transfer 580
24.7.1 CPU Controlled Master Mode 580
24.7.2 CPU Controlled Slave Mode 581
24.8 DMA Controlled Data Transfer 582
24.9 GP-SPI Clock Control 583
24.9.1 GP-SPI Clock Phase and Polarity 583
24.9.2 GP-SPI Clock Control in Master Mode 584
24.9.3 GP-SPI Clock Control in Slave Mode 585
24.9.4 GP-SPI Timing Compensation 585
24.10 SPI Pin Mapping 586
24.11 GP-SPI Interrupt Control 586
24.11.1 GP-SPI Interrupt 590
24.11.2 GP-SPI DMA Interrupts 590
24.12 Register Base Address 591
24.13 Register Summary 591
24.14 Registers 594

25 I2C Controller (I2C) 632
25.1 Overview 632
25.2 Features 632
25.3 I2C Functional Description 632
25.3.1 I2C Introduction 632

25.3.2 I2C Architecture 633
25.3.2.1 TX/RX RAM 634
25.3.2.2 CMD_Controller 634
25.3.2.3 SCL_FSM 635
25.3.2.4 SCL_MAIN_FSM 636
25.3.2.5 DATA_Shifter 636
25.3.2.6 SCL_Filter and SDA_Filter 636
25.3.3 I2C Bus Timing 636
25.4 Typical Applications 637
25.4.1 An I2C Master Writes to an I2C Slave with a 7-bit Address in One Command Sequence 638
25.4.2 An I2C Master Writes to an I2C Slave with a 10-bit Address in One Command Sequence 639
25.4.3 An I2C Master Writes to an I2C Slave with Two 7-bit Addresses in One Command Sequence 640
25.4.4 An I2C Master Writes to an I2C Slave with a 7-bit Address in Multiple Command Sequences 640
25.4.5 An I2C Master Reads an I2C Slave with a 7-bit Address in One Command Sequence 641
25.4.6 An I2C Master Reads an I2C Slave with a 10-bit Address in One Command Sequence 642
25.4.7 An I2C Master Reads an I2C Slave with Two 7-bit Addresses in One Command Sequence 643
25.4.8 An I2C Master Reads an I2C Slave with a 7-bit Address in Multiple Command Sequences 644
25.5 Clock Stretching 644
25.6 Interrupts 645
25.7 Base Address 646
25.8 Register Summary 646
25.9 Registers 647

26 I2S Controller (I2S) 670
26.1 Overview 670
26.2 System Diagram 670
26.3 Features 672
26.4 Supported Audio Standards 673
26.4.1 Philips Standard 673
26.4.2 MSB Alignment Standard 674
26.4.3 PCM Standard 674
26.5 I2S Clock 674
26.6 I2S Reset 676
26.7 I2S Master/Slave Mode 676
26.7.1 Master/Slave Transmitting Mode 676
26.7.2 Master/Slave Receiving Mode 677
26.8 Transmitting Data 677
26.8.1 Data Transmitting When I2S_TX_DMA_EQUAL = 0 678
26.8.2 Data Transmitting When I2S_TX_DMA_EQUAL = 1 681
26.8.3 Configuring I2S as TX Mode 682
26.9 Receiving Data 683
26.9.1 Data Receiving When I2S_RX_DMA_EQUAL = 0 683
26.9.2 Data Receiving When I2S_RX_DMA_EQUAL = 1 684
26.9.3 Configuring I2S as RX Mode 685
26.10 LCD Master Transmitting Mode 686
26.10.1 Overview 686

26.10.2 Configure I2S as LCD Master Transmitting Mode 686
26.11 Camera Slave Receiving Mode 687
26.11.1 Overview 687
26.11.2 Configure I2S as Camera Slave Receiving Mode 688
26.12 I2S Interrupts 689
26.12.1 FIFO Interrupts 689
26.12.2 DMA Interrupts 689
26.13 Base Address 690
26.14 Register Summary 690
26.15 Registers 692

27 Pulse Count Controller (PCNT) 712
27.1 Features 712
27.2 Functional Description 713
27.3 Applications 715
27.3.1 Channel 0 Incrementing Independently 715
27.3.2 Channel 0 Decrementing Independently 716
27.3.3 Channel 0 and Channel 1 Incrementing Together 716
27.4 Base Address 717
27.5 Register Summary 717
27.6 Registers 719

28 USB On-The-Go (USB) 725
28.1 Overview 725
28.2 Features 725
28.2.1 General Features 725
28.2.2 Device Mode Features 725
28.2.3 Host Mode Features 725
28.3 Functional Description 725
28.3.1 Controller Core and Interfaces 726
28.3.2 Memory Layout 727
28.3.2.1 Control & Status Registers 727
28.3.2.2 FIFO Access 728
28.3.3 FIFO and Queue Organization 728
28.3.3.1 Host Mode FIFOs and Queues 728
28.3.3.2 Device Mode FIFOs 730
28.3.4 Interrupt Hierarchy 730
28.3.5 DMA Modes and Slave Mode 731
28.3.5.1 Slave Mode 732
28.3.5.2 Buffer DMA Mode 732
28.3.5.3 Scatter/Gather DMA Mode 732
28.3.6 Transaction and Transfer Level Operation 733
28.3.6.1 Transaction and Transfer Level in DMA Mode 733
28.3.6.2 Transaction and Transfer Level in Slave Mode 733
28.4 OTG 735
28.4.1 ID Pin Detection 735

28.4.2 OTG Interface 736
28.4.3 Session Request Protocol (SRP) 737
28.4.3.1 A-Device SRP 737
28.4.3.2 B-Device SRP 737
28.4.4 Host Negotiation Protocol (HNP) 738
28.4.4.1 A-Device HNP 738
28.4.4.2 B-Device HNP 740
28.5 Base Address 741

29 Two-wire Automotive Interface (TWAI) 742
29.1 Overview 742
29.2 Features 742
29.3 Functional Protocol 742
29.3.1 TWAI Properties 742
29.3.2 TWAI Messages 743
29.3.2.1 Data Frames and Remote Frames 744
29.3.2.2 Error and Overload Frames 746
29.3.2.3 Interframe Space 747
29.3.3 TWAI Errors 748
29.3.3.1 Error Types 748
29.3.3.2 Error States 748
29.3.3.3 Error Counters 749
29.3.4 TWAI Bit Timing 750
29.3.4.1 Nominal Bit 750
29.3.4.2 Hard Synchronization and Resynchronization 751
29.4 Architectural Overview 751
29.4.1 Registers Block 751
29.4.2 Bit Stream Processor 753
29.4.3 Error Management Logic 753
29.4.4 Bit Timing Logic 753
29.4.5 Acceptance Filter 753
29.4.6 Receive FIFO 753
29.5 Functional Description 753
29.5.1 Modes 753
29.5.1.1 Reset Mode 754
29.5.1.2 Operation Mode 754
29.5.2 Bit Timing 754
29.5.3 Interrupt Management 755
29.5.3.1 Receive Interrupt (RXI) 755
29.5.3.2 Transmit Interrupt (TXI) 756
29.5.3.3 Error Warning Interrupt (EWI) 756
29.5.3.4 Data Overrun Interrupt (DOI) 756
29.5.3.5 Error Passive Interrupt (TXI) 756
29.5.3.6 Arbitration Lost Interrupt (ALI) 757
29.5.3.7 Bus Error Interrupt (BEI) 757
29.5.4 Transmit and Receive Buffers 757

29.5.4.1 Overview of Buffers 757
29.5.4.2 Frame Information 758
29.5.4.3 Frame Identifier 758
29.5.4.4 Frame Data 759
29.5.5 Receive FIFO and Data Overruns 759
29.5.6 Acceptance Filter 760
29.5.6.1 Single Filter Mode 760
29.5.6.2 Dual FIlter Mode 761
29.5.7 Error Management 762
29.5.7.1 Error Warning Limit 763
29.5.7.2 Error Passive 763
29.5.7.3 Bus-Off and Bus-Off Recovery 763
29.5.8 Error Code Capture 764
29.5.9 Arbitration Lost Capture 765
29.6 Base Address 765
29.7 Register Summary 766
29.8 Register Description 767

30 LED PWM Controller (LEDC) 780
30.1 Overview 780
30.2 Features 780
30.3 Functional Description 780
30.3.1 Architecture 780
30.3.2 Timers 780
30.3.3 PWM Generators 782
30.3.4 Duty Cycle Fading 782
30.3.5 Interrupts 783
30.4 Base Address 783
30.5 Register Summary 784
30.6 Registers 786

31 Remote Control Peripheral (RMT) 793
31.1 Introduction 793
31.2 Functional Description 793
31.2.1 RMT Architecture 793
31.2.2 RMT RAM 794
31.2.3 Clock 794
31.2.4 Transmitter 794
31.2.5 Receiver 795
31.2.6 Interrupts 795
31.3 Base Address 796
31.4 Register Summary 796
31.5 Registers 798

32 On-Chip Sensor and Analog Signal Processing 807
32.1 Overview 807

32.2 SAR ADCs 807
32.2.1 Overview 807
32.2.2 Features 807
32.2.3 Functional Description 808
32.2.3.1 Input Signals 809
32.2.3.2 ADC Conversion and Attenuation 810
32.2.4 RTC ADC Controllers 810
32.2.5 DIG ADC Controllers 811
32.2.5.1 Workflow of DIG ADC Controller 811
32.2.5.2 DMA 813
32.2.5.3 ADC Filter 813
32.2.5.4 Threshold Monitoring 814
32.2.6 SAR ADC2 Arbiter 814
32.3 DACs 815
32.3.1 Overview 815
32.3.2 Features 815
32.3.3 DAC Conversion 816
32.3.4 Cosine Wave Generator 816
32.3.5 DMA Support 817
32.4 Temperature Sensor 817
32.4.1 Overview 817
32.4.2 Features 817
32.4.3 Operation Sequence 818
32.4.4 Temperature Conversion 818
32.5 Interrupts 819
32.6 Base Address 819
32.7 Register Summary 819
32.7.1 SENSOR (RTC_PERI) Register Summary 819
32.7.2 SENSOR (DIG_PERI) Register Summary 820
32.8 Register 821
32.8.1 SENSOR (RTC_PERI) Registers 821
32.8.2 SENSOR (DIG_PERI) Registers 829

Glossary 841
Abbreviations for Peripherals 841
Abbreviations for Registers 841

Revision History 842

List of Tables

1 Comparison of the Two Coprocessors 28
2 ALU Operations Among Registers 33
3 ALU Operations with Immediate Value 34
4 ALU Operations with Stage Count Register 34
5 Data Storage Type - Automatic Storage Mode 36
6 Data Storage - Manual Storage Mode 37
7 Input Signals Measured Using the ADC Instruction 42
8 Instruction Efficiency 45
9 ULP-RISC-V Interrupt List 45
10 ULP Coprocessor Base Address 49
11 RTC I2C Base Address 49
16 Relationship Between Configuration Register, Block Size and Alignment 79
17 Copy DMA and Crypto DMA Base Address 83
20 Address Mapping 112
21 Internal Memory Address Mapping 112
22 External Memory Address Mapping 114
23 Peripherals with DMA Support 116
24 Module / Peripheral Address Mapping 117
25 Addresses with Restricted Access 119
26 Parameters in BLOCK0 120
27 Key Purpose Values 123
28 Parameters in BLOCK1-10 123
29 Registers for Software Reading Parameters 127
30 Configuration of eFuse-Programming Timing Parameters 128
31 Configuration of VDDQ Timing Parameters 129
32 Configuration of eFuse-Reading Parameters 129
33 eFuse Controller Base Address 130
35 Pin Function Register for IO MUX Light-sleep Mode 165
36 GPIO Matrix 166
37 IO MUX Pad List 170
38 RTC IO MUX Pin Summary 172
39 GPIO, IO MUX, GPIOSD, Dedicated GPIO, and RTCIO Base Addresses 173
45 Reset Source 216
46 CPU_CLK Source 218
47 CPU_CLK Selection 218
48 Peripheral Clock Usage 219
49 APB_CLK Source 219
50 REF_TICK Source 220
51 LEDC_PWM_CLK Source 220
52 Default Configuration of Strapping Pins 222
53 Boot Mode 222
54 ROM Code Printing Control 223
55 CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources 225
56 CPU Interrupts 228

57 Interrupt Matrix Base Address 230
59 Low-Power Clocks 268
60 The Triggering Conditions for the RTC Timer 268
61 Brown-out Detector Configuration 272
62 RTC Statues Transition 274
63 Predefined Power Modes 275
64 Wakeup Source 276
65 Low-power Management Base Address 278
67 System Timer Base Address 319
69 64-bit Timers Base Address 332
71 Offset Address Range of Each SRAM Block 353
72 Permission Control for IBUS to Access SRAM 354
73 Permission Control for IBUS to Access RTC FAST Memory 355
74 Permission Control for DBUS0 to Access SRAM 356
75 Permission Control for DBUS0 to Access RTC FAST Memory 357
76 Permission Control for On-chip DMA to Access SRAM 358
77 Peripherals and FIFO Address 359
78 Permission Control for PeriBus1 359
79 Permission Control for PeriBus2 to Access RTC SLOW Memory 360
80 Configuration of Register PMS_PRO_CACHE_1_REG 361
81 MMU Entries 362
82 Non-Aligned Access to Peripherals 364
83 Permission Control Base Address 364
85 ROM Controlling Bit 395
86 SRAM Controlling Bit 395
87 Peripheral Clock Gating and Reset Bits 397
88 System Register Base Address 399
90 SHA Accelerator Working Mode 417
91 SHA Hash Algorithm 417
95 The Storage and Length of Message digest from Different Algorithms 424
96 SHA Accelerator Base Address 425
98 AES Accelerator Working Mode 432
99 Operation Type under Typical AES Working Mode 432
100 Working Status under Typical AES Working Mode 432
101 Text Endianness Types for Typical AES 433
102 Key Endianness Types for AES-128 Encryption and Decryption 435
103 Key Endianness Types for AES-192 Encryption and Decryption 435
104 Key Endianness Types for AES-256 Encryption and Decryption 436
105 Operation Type under DMA-AES Working Mode 438
106 Working Status under DMA-AES Working mode 438
107 TEXT-PADDING 439
108 Text Endianness for DMA-AES 439
109 AES Accelerator Base Address 445
110 AES Accelerator Memory Blocks 445
112 Acceleration Performace 456
113 RSA Accelerator Base Address 457

114 RSA Accelerator Memory Blocks 457
116 HMAC Function and Configuration Value 464
117 HMAC Base Address 468
119 Digital Signature Base Address 479
120 Digital Signature Memory Blocks 479
122 Key 483
123 Mapping Between Offsets and Registers 485
124 Manual Encryption Block Base Address 487
126 Random Number Generator Base Address 493
128 UART0, UART1 and UHCI0 Base Address 506
131 Data Modes Supported by GP-SPI2 and GP-SPI3 559
132 Register Configuration Rules for State Control in 1/2-bit Modes 561
132 Register Configuration Rules for State Control in 1/2-bit Modes 562
133 Register Configuration Rules for State Control in 4/8-bit Modes 562
134 GP-SPI Master BM Table for CONF State 569
135 An Example of CONF bufferi in Segmented-Configure-Transfer 570
136 BM Bit Value v.s. Register to Be Updated in the Example 570
136 BM Bit Value v.s. Register to Be Updated in the Example 571
137 Supported CMD Values in SPI Mode 576
137 Supported CMD Values in SPI Mode 577
138 Supported CMD Values in QPI Mode 577
139 Invalid Registers and Fields for GP-SPI3 579
139 Invalid Registers and Fields for GP-SPI3 580
140 Clock Phase and Polarity Configuration in Master Mode 584
141 Clock Phase and Polarity Configuration in Slave Mode 585
142 Mapping of SPI Signal Buses and Chip Pads 586
143 GP-SPI Master Mode Interrupts 587
144 GP-SPI Slave Mode Interrupts 588
145 SPI Base Address 591
147 I2C Controller Base Address 646
149 I2S Signal Description 671
150 Endianness Mode of TX Data 678
151 TX Channel Mode When I2S_TX_DMA_EQUAL = 0 680
152 TX Channel Mode When I2S_TX_DMA_EQUAL = 1 682
153 RX Channel Mode When I2S_RX_DMA_EQUAL = 0 684
154 RX Channel Mode When I2S_RX_DMA_EQUAL = 1 684
155 I2S Register Base Address 690
157 Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in Low State 714
158 Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in High State 714
159 Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in Low State 714
160 Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in High State 714
161 PCNT Base Address 717
163 IN and OUT Transactions in Slave Mode 733
164 UTMI OTG Interface 736
165 USB OTG Base Address 741
166 Data Frames and Remote Frames in SFF and EFF 745

167 Error Frame 746
168 Overload Frame 747
169 Interframe Space 748
170 Segments of a Nominal Bit Time 750
171 Bit Information of TWAI_CLOCK_DIVIDER_REG; TWAI Address 0x18 754
172 Bit Information of TWAI_BUS_TIMING_1_REG; TWAI Address 0x1c 755
173 Buffer Layout for Standard Frame Format and Extended Frame Format 757
174 TX/RX Frame Information (SFF/EFF) TWAI Address 0x40 758
175 TX/RX Identifier 1 (SFF); TWAI Address 0x44 758
176 TX/RX Identifier 2 (SFF); TWAI Address 0x48 759
177 TX/RX Identifier 1 (EFF); TWAI Address 0x44 759
178 TX/RX Identifier 2 (EFF); TWAI Address 0x48 759
179 TX/RX Identifier 3 (EFF); TWAI Address 0x4c 759
180 TX/RX Identifier 4 (EFF); TWAI Address 0x50 759
181 Bit Information of TWAI_ERR_CODE_CAP_REG; TWAI Address 0x30 764
182 Bit Information of Bits SEG.4 - SEG.0 764
183 Bit Information of TWAI_ARB LOST CAP_REG; TWAI Address 0x2c 765
184 TWAI Base Address 766
186 LED PWM Controller Base Address 783
188 RMT Base Address 796
190 SAR ADC Controllers 808
191 SAR ADC Input Signals 809
192 Fields of Pattern Table Register 812
193 DMA Data Format (Type I) 813
194 DMA Data Format (Type II) 813
195 Temperature Offset 819
196 On-Chip Sensor, SAR ADCs, and DACs Base Addresses 819

List of Figures

1-1 ULP Coprocessor Overview 27
1-2 ULP Coprocessor Diagram 28
1-3 Programing Workflow 29
1-4 Sample of a ULP Operation Sequence 30
1-5 Control of ULP Program Execution 31
1-6 ULP-FSM Instruction Format 32
1-7 Instruction Type — ALU for Operations Among Registers 32
1-8 Instruction Type — ALU for Operations with Immediate Value 33
1-9 Instruction Type — ALU for Operations with Stage Count Register 34
1-10 Instruction Type - ST 35
1-11 Instruction Type - Offset in Automatic Storage Mode (ST-OFFSET) 35
1-12 Instruction Type - Data Storage in Automatic Storage Mode (ST-AUTO-DATA) 35
1-13 Data Structure of RTC_SLOW_MEM[Rdst + Offset] 36
1-14 Instruction Type - Data Storage in Manual Storage Mode 37
1-15 Instruction Type - LD 37
1-16 Instruction Type- JUMP 38
1-17 Instruction Type - JUMPR 39
1-18 Instruction Type - JUMPS 39
1-19 Instruction Type- HALT 41
1-20 Instruction Type - WAKE 41
1-21 Instruction Type - WAIT 41
1-22 Instruction Type - TSENS 42
1-23 Instruction Type - ADC 42
1-24 Instruction Type - REG_RD 43
1-25 Instruction Type - REG_WR 44
1-26 I2C Read Operation 47
1-27 I2C Write Operation 48
2-1 Modules with DMA and Supported Data Transfers 74
2-2 DMA Engine Architecture 75
2-3 Structure of a Linked List 75
2-4 Relationship among Linked Lists 77
2-5 EDMA Receiving Data Frames in Internal 78
2-6 Copy DMA Engine Architecture 79
2-7 Data Transfer in UDMA Mode 80
2-8 SPI DMA 80
3-1 System Structure and Address Mapping 111
3-2 Cache Structure 115
4-1 Shift Register Circuit 125
4-2 eFuse-Programming Timing Diagram 128
4-3 Timing Diagram for Reading eFuse 130
5-1 IO MUX, RTC IO MUX and GPIO Matrix Overview 156
5-2 GPIO Input Synchronized on Clock Rising Edge or on Falling Edge 158
5-3 Filter Timing Diagram of GPIO Input Signals 158
5-4 Dedicated GPIO Diagram 162

6-1 System Reset 215
6-2 System Clock 217
8-1 Interrupt Matrix Structure 224
9-1 Low-power Management Schematics 264
9-2 Power Management Unit Workflow 265
9-3 Fast Clocks for RTC Power Domains 267
9-4 Slow Clocks for RTC Power Domains 267
9-5 Low-Power Clocks for RTC Power Domains 267
9-6 Digital System Regulator 270
9-7 Low-power voltage regulator 270
9-8 Flash voltage regulator 271
9-9 Brown-out detector 272
9-10 RTC States Transition 274
9-11 ESP32-S2 Boot Flow 278
10-1 System Timer Structure 317
11-1 Timer Units within Groups 328
12-1 Super Watchdog Controller Structure 350
13-1 XTAL32K Watchdog Timer 351
17-1 GCM Encryption Process 443
19-1 HMAC SHA-256 Padding Diagram 466
19-2 HMAC Structure Schematic Diagram 467
20-1 Preparations and DS Operation 476
21-1 Architecture of the External Memory Encryption and Decryption Module 482
22-1 Noise Source 492
23-1 UART Structure 496
23-2 UART Controllers Sharing RAM 497
23-3 UART Controllers Division 498
23-4 The Timing Diagram of Weak UART Signals Along Falling Edges 499
23-5 Structure of UART Data Frame 499
23-6 AT_CMD Character Structure 500
23-7 Driver Control Diagram in RS485 Mode 501
23-8 The Timing Diagram of Encoding and Decoding in SIR mode 502
23-9 IrDA Encoding and Decoding Diagram 502
23-10 Hardware Flow Control Diagram 503
23-11 Connection between Hardware Flow Control Signals 503
24-1 SPI Block Diagram 555
24-2 GP-SPI2/GP-SPI3 Block Diagram 558
24-3 GP-SPI2 State Flow in Master Mode 560
24-4 Full-Duplex Communication Between GP-SPI2 Master and a Slave 564
24-5 Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode 566
24-6 SPI Quad Read Command Sequence Sent by GP-SPI2 to Flash 566
24-7 Connection of GP-SPI2 to 8-bit LCD Driver 567
24-8 Write Command Sequence to an 8-bit LCD Driver 567
24-9 Segmented-Configure-Transfer in DMA Controlled Master Mode 568
24-10 Video Frame Structure in Parallel RGB 8-bit LCD Mode 571
24-11 Timing Sequence in Parallel RGB 8-bit LCD Mode 572

24-12 Recommended CS Timing and Settings When Access External RAM 573
24-13 Recommended CS Timing and Settings When Access Flash 574
24-14 Data Buffer Used in CPU-Controlled Mode 581
24-15 SPI CLK Mode 0 or 2 584
24-16 SPI CLK Mode 1 or 3 585
25-1 I2C Master Architecture 633
25-2 I2C Slave Architecture 633
25-3 Structure of I2C Command Register 634
25-4 I2C Timing Diagram 636
25-5 An I2C Master Writing to an I2C Slave with a 7-bit Address 638
25-6 A Master Writing to a Slave with a 10-bit Address 639
25-7 An I2C Master Writing Address M in the RAM to an I2C Slave with a 7-bit Address 640
25-8 An I2C Master Writing to an I2C Slave with a 7-bit Address in Multiple Sequences 640
25-9 An I2C Master Reading an I2C Slave with a 7-bit Address 641
25-10 An I2C Master Reading an I2C Slave with a 10-bit Address 642
25-11 An I2C Master Reading N Bytes of Data from addrM of an I2C Slave with a 7-bit Address 643
25-12 An I2C Master Reading an I2C Slave with a 7-bit Address in Segments 644
26-1 ESP32-S2 I2S System Diagram 670
26-2 Philips Standard 673
26-3 MSB Alignment Standard 674
26-4 PCM Standard 674
26-5 I2S Clock 675
26-6 ESP32-S2 I2S Data Transmitting Flow When I2S_TX_DMA_EQUAL = 0 679
26-7 I2S Output Format When I2S_TX_CHAN_MOD[2:0] = 0 and I2S_TX_DMA_EQUAL = 0 680
26-8 I2S TX Data When I2S_TX_DMA_EQUAL = 1 681
26-9 I2S RX Data When I2S_RX_DMA_EQUAL = 0 683
26-10 I2S RX Data When I2S_RX_DMA_EQUAL = 0 684
26-11 ESP32-S2 I2S RX Data When I2S_RX_DMA_EQUAL = 1 685
26-12 LCD Master Transmitting Mode 686
26-13 Data Frame Format 1 in LCD Master Transmitting Mode 686
26-14 Data Frame Format 2 in LCD Master Transmitting Mode 687
26-15 Camera Slave Receiving Mode 688
27-1 PCNT Block Diagram 712
27-2 PCNT Unit Architecture 713
27-3 Channel 0 Up Counting Diagram 715
27-4 Channel 0 Down Counting Diagram 716
27-5 Two Channels Up Counting Diagram 716
28-1 OTG_FS System Architecture 726
28-2 OTG_FS Register Layout 727
28-3 Host Mode FIFOs 729
28-4 Device Mode FIFOs 730
28-5 OTG_FS Interrupt Hierarchy 731
28-6 Scatter/Gather DMA Descriptor List 732
28-7 A-Device SRP 737
28-8 B-Device SRP 738
28-9 A-Device HNP 739

28-10 B-Device HNP 740
29-1 The bit fields of Data Frames and Remote Frames 744
29-2 Various Fields of an Error Frame 746
29-3 The Bit Fields of an Overload Frame 747
29-4 The Fields within an Interframe Space 747
29-5 Layout of a Bit 750
29-6 TWAI Overview Diagram 752
29-7 Acceptance Filter 760
29-8 Single Filter Mode 761
29-9 Dual Filter Mode 762
29-10 Error State Transition 763
29-11 Positions of Arbitration Lost Bits 765
30-1 LED_PWM Architecture 780
30-2 LED_PWM generator Diagram 781
30-3 LED_PWM Divider 781
30-4 LED_PWM Output Signal Diagram 782
30-5 Output Signal Diagram of Fading Duty Cycle 783
31-1 RMT Architecture 793
31-2 Format of Pulse Code in RAM 794
32-1 SAR ADC Overview 807
32-2 SAR ADC Function Overview 809
32-3 RTC SAR ADC Outline 810
32-4 Diagram of DIG ADC Controllers 812
32-5 Diagram of DAC Function 816
32-6 Workflow of CW Generator 816
32-7 Structure of Temperature Sensor 818

1. ULP Coprocessor (ULP)

1.1 Overview

The ULP coprocessor is an ultra-low-power processor that remains powered on when the chip is in Deep-sleep (see Chapter 9 *Low-Power Management (RTC_CNTL)*). Hence, users can store in RTC memory a program for the ULP coprocessor to access peripheral devices, internal sensors and RTC registers during Deep-sleep.

In power-sensitive scenarios, the main CPU goes to sleep mode to lower power consumption. Meanwhile, the coprocessor is woken up by ULP timer, and then monitors the external environment or interacts with the external circuit by controlling peripheral devices such as RTCIO, RTC I2C, SAR ADC, or temperature sensor (TSENS). The coprocessor wakes the main CPU up once a wakeup condition is reached.

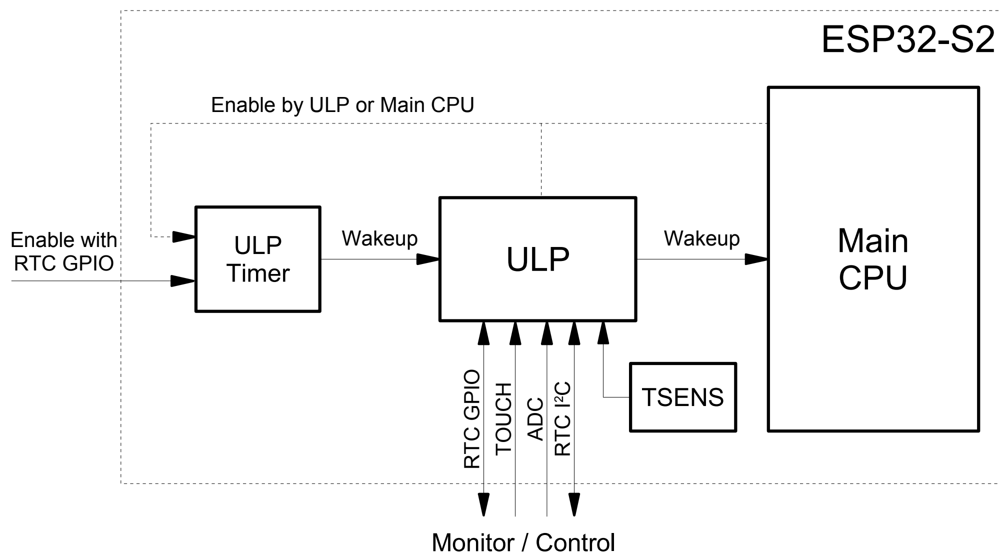


Figure 1-1. ULP Coprocessor Overview

ESP32-S2 has two ULP coprocessors, with one based on RISC-V instruction set architecture (ULP-RISC-V) and the other on finite state machine (ULP-FSM). Users can choose between the two coprocessors depending on their needs.

1.2 Features

- Access up to 8 KB of SRAM RTC slow memory for instructions and data
- Clocked with 8 MHz RTC_FAST_CLK
- Support working in normal mode and in monitor mode
- Wake up the CPU or send an interrupt to the CPU
- Access peripherals, internal sensors and RTC registers

ULP-FSM and ULP-RISC-V can not be used simultaneously. Users can only choose one of them as the ULP coprocessor of ESP32-S2. The differences between the two coprocessors are shown in the table below.

Feature		Coprocessors	
		ULP-FSM	ULP-RISC-V
Memory (RTC Slow Memory)		8 KB	
Work Clock Frequency		8 MHz	
Wakeup Source		ULP Timer	
Work Mode	Normal Mode	Assist the main CPU to complete some tasks after the system is woken up.	
	Monitor Mode	Control sensors to do tasks such as monitoring environment, when the system is in sleep.	
Control Low-Power Peripherals		ADC0/ADC1	
		DAC0/DAC1	
		RTC I2C	
		RTC GPIO	
		Touch Sensor	
		Temperature Sensor	
Architecture		Programmable FSM	RISC-V
Development		Special instruction set	Standard C compiler

Table 1: Comparison of the Two Coprocessors

ULP coprocessor can access the modules in RTC domain via RTC registers. In many cases the ULP coprocessor can be a good supplement to, or replacement of, the main CPU, especially for power-sensitive applications. Figure 1-2 shows the overall layout of ESP32-S2 coprocessor.

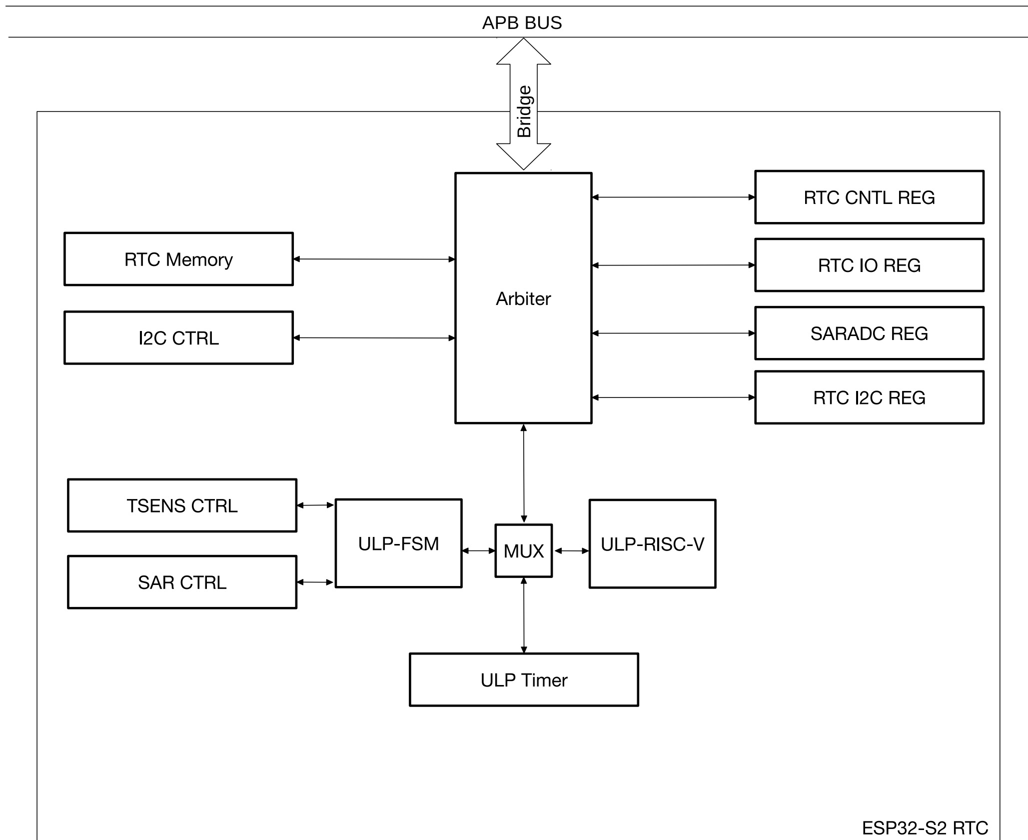


Figure 1-2. ULP Coprocessor Diagram

1.3 Programming Workflow

The ULP-RISC-V is intended for programming using C language. The program in C is then compiled to RV32IMC standard instruction code. The ULP-FSM is using custom instructions normally not supported by high-level programming language. Users develop their programs using ULP-FSM instructions (see Section 1.5.2).

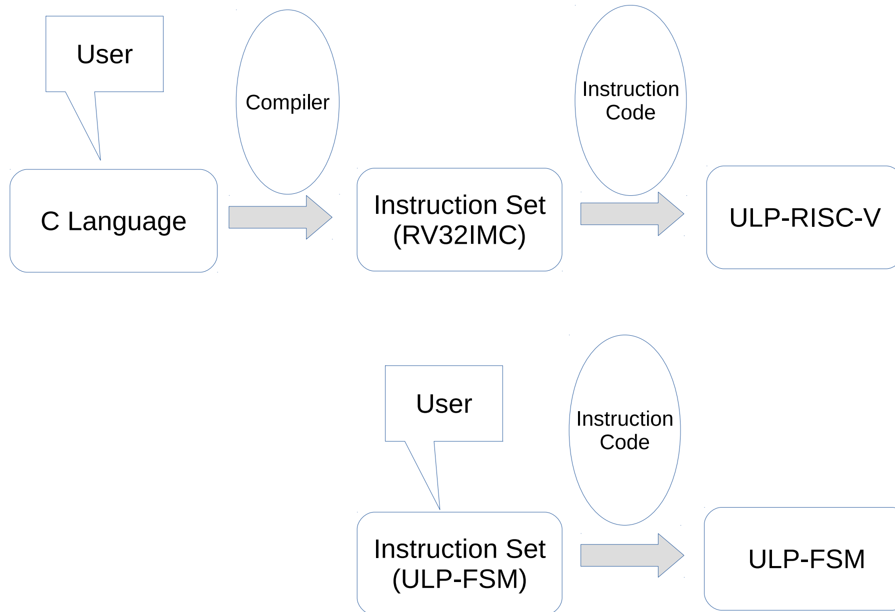


Figure 1-3. Programing Workflow

1.4 ULP Coprocessor Workflow

ULP coprocessor is designed to operate independently of the CPU, while the CPU is either in sleep or running.

In a typical power-saving scenario, the chip goes to Deep-sleep mode to lower power consumption. Before setting the chip to sleep mode, users should complete the following operations.

1. Flash the program to be executed by ULP coprocessor into RTC slow memory.
2. Select the working ULP coprocessor by configuring the register RTC_CNTL_COCPU_SEL.
    - 0: select ULP-RISC-V
    - 1: select ULP-FSM
3. Set sleep cycles for the timer by configuring RTC_CNTL_ULP_CP_TIMER_1_REG.
4. Enable the timer by software or by RTC GPIO;
    - By software: set the register RTC_CNTL_ULP_CP_SLP_TIMER_EN.
    - By RTC GPIO: set the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA.
5. Set the system into sleep mode.

When the system is in Deep-sleep mode:

1. The timer periodically sets the low-power controller (see Chapter 9 *Low-Power Management (RTC_CNTL)*) to Monitor mode and then wakes up the coprocessor.

2. Coprocessor executes some necessary operations, such as monitoring external environment via low-power sensors.
3. After the operations are finished, the system goes back to Deep-sleep mode.
4. ULP coprocessor goes back to halt mode and waits for next wakeup.

In monitor mode, ULP coprocessor is woken up and goes to halt as shown in Figure 1-4.

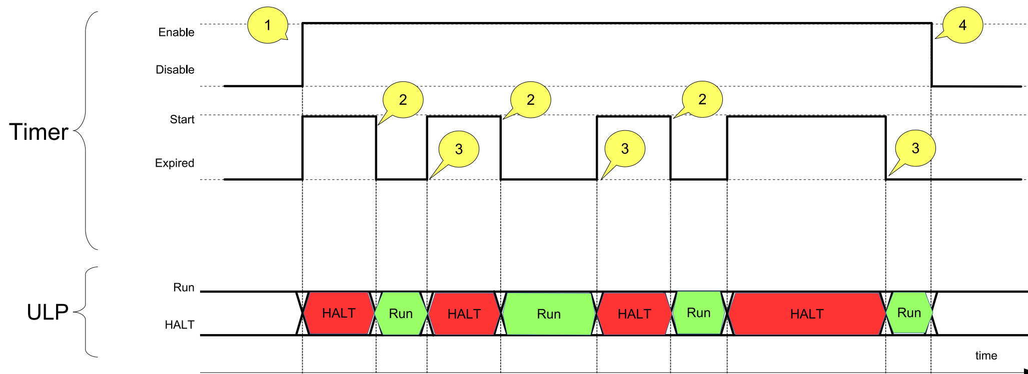


Figure 1-4. Sample of a ULP Operation Sequence

1. Enable the timer and the timer starts counting.
2. The timer expires and wakes up the ULP coprocessor. ULP coprocessor starts running and executes the program flashed in RTC slow memory.
3. ULP coprocessor goes to halt and the timer starts counting again.
   - Put ULP-RISC-V into HALT: set the register RTC_CNTL_COCPU_DONE,
   - Put ULP-FSM into HALT: execute HALT instruction.
4. Disable the timer by ULP program or by software. The system exits from monitor mode.
   - Disabled by software: clear the register RTC_CNTL_ULP_CP_SLP_TIMER_EN.
   - Disabled by RTC GPIO: clear the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA, and set the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_CLR.

Note:

- If the timer is enabled by software (RTC GPIO), it should be disabled by software (RTC GPIO).
- Before setting ULP-RISC-V to HALT, users should configure the register RTC_CNTL_COCPU_DONE first, therefore, it is recommended to end the flashed program with the following pattern:
  - Set the register RTC_CNTL_COCPU_DONE to end the operation of ULP-RISC-V and put it into halt;
  - Set the register RTC_CNTL_COCPU_SHUT_RESET_EN to reset ULP-RISC-V.

  Enough time is reserved for the ULP-RISC-V to complete the operations above before it goes to halt.

The relationship between the signals and registers is shown in Figure 1-5.

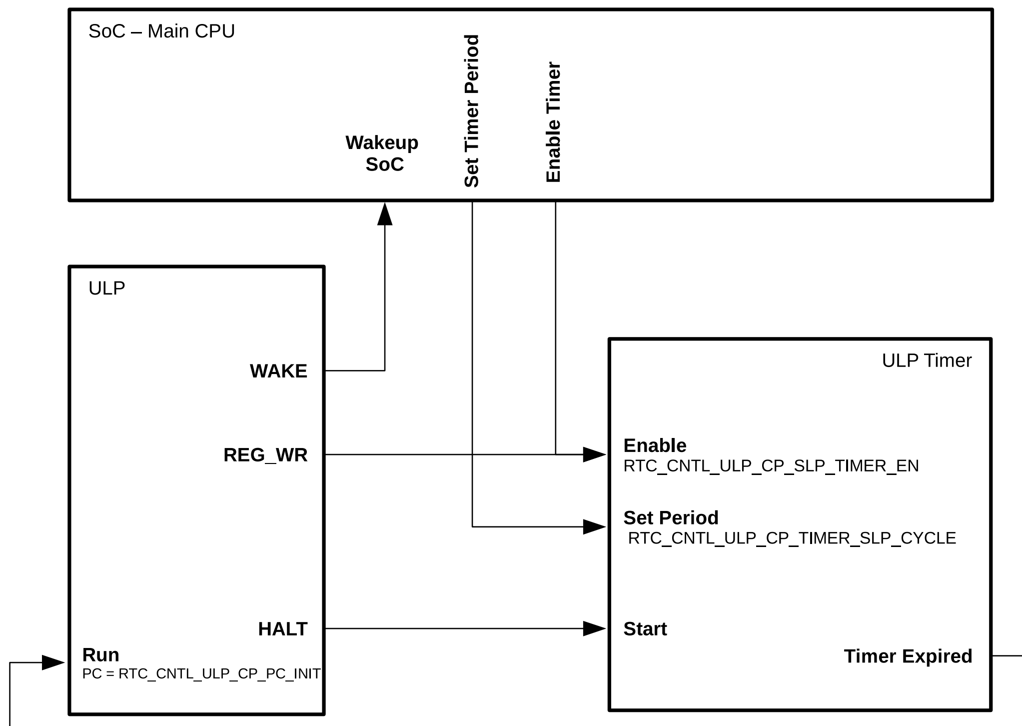


Figure 1-5. Control of ULP Program Execution

1.5 ULP-FSM

1.5.1 Features

ULP-FSM is a programmable finite state machine that can work while the main CPU is in Deep-sleep. ULP-FSM supports instructions for complex logic and arithmetic operations, and also provides dedicated instructions for RTC controllers or peripherals. ULP-FSM can access up to 8 KB of SRAM RTC slow memory (accessible by the CPU) for instructions and data. Hence, such memory is usually used to store instructions and share data between the ULP coprocessor and the CPU. ULP-FSM can be stopped by running HALT instruction.

ULP-FSM has the following features.

- Provide four 16-bit general-purpose registers (R0, R1, R2, and R3) for manipulating data and accessing memory.
- Provide one 8-bit stage count register (Stage_cnt) which can be manipulated by ALU and used in JUMP instructions.
- Support built-in instructions specially for direct control of low-power peripherals, such as SAR ADC and temperature sensor.

1.5.2 Instruction Set

ULP-FSM supports the following instructions.

- ALU: perform arithmetic and logic operations
- LD, ST, REG_RD and REG_WR: load and store data
- JUMP: jump to a certain address
- WAIT/HALT: manage program execution

- WAKE: wake up CPU or communicate with the CPU
- TSENS and ADC: take measurements

The format of ULP-FSM instructions is shown in Figure 1-6.

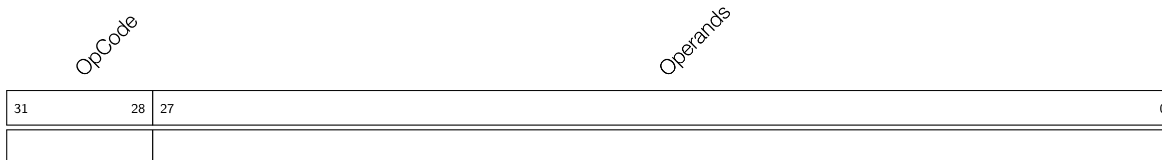


Figure 1-6. ULP-FSM Instruction Format

An instruction, which has one OpCode, can perform various operations, depending on the setting of Operands bits. A good example is the ALU instruction, which is able to perform 10 arithmetic and logic operations; or the JUMP instruction, which may be conditional or unconditional, absolute or relative.

Each instruction has a fixed width of 32 bits. A series of instructions can make a program be executed by the coprocessor. The execution flow inside the program uses 32-bit addressing. The program is stored in a dedicated region called Slow Memory, which is visible to the main CPU as one that has an address range of 0x5000_0000 to 0x5000_1FFF (8 KB).

ALU - Perform Arithmetic and Logic Operations

ALU (Arithmetic and Logic Unit) performs arithmetic and logic operations on values stored in ULP coprocessor registers, and on immediate values stored in the instruction itself. The following operations are supported.

- Arithmetic: ADD and SUB
- Logic: AND and OR
- Bit shifting: LSH and RSH
- Moving data to register: MOVE
- PC register operations - STAGE_RST, STAGE_INC, and STAGE_DEC

The ALU instruction, which has one OpCode (7), can perform various arithmetic and logic operations, depending on the setting of the instruction bits [27:21].

Operations Among Registers

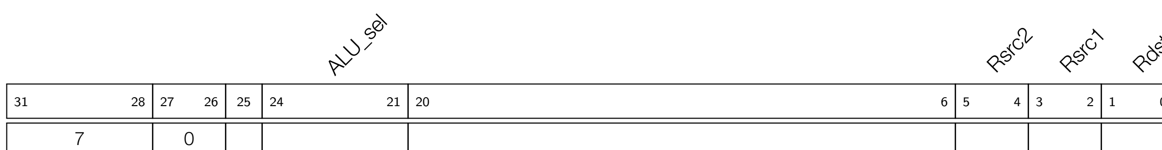


Figure 1-7. Instruction Type — ALU for Operations Among Registers

When bits [27:26] of the instruction in Figure 1-7 are set to 0, ALU performs operations on the data stored in ULP-FSM registers R[0-3]. The types of operations depend on the setting of the instruction bits ALU_sel [24:21] presented in Table 2.

Operand	**Description** - see Figure 1-7
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Rsrc2	Register R[0-3], source
ALU_sel	ALU Operation

ALU_sel	Instruction	Operation	Description
0	ADD	*Rdst* = *Rsrc1* + *Rsrc2*	Add to register
1	SUB	*Rdst* = *Rsrc1* - *Rsrc2*	Subtract from register
2	AND	*Rdst* = *Rsrc1* & *Rsrc2*	Logical AND of two operands
3	OR	*Rdst* = *Rsrc1* | *Rsrc2*	Logical OR of two operands
4	MOVE	*Rdst* = *Rsrc1*	Move to register
5	LSH	*Rdst* = *Rsrc1* << *Rsrc2*	Logical shift left
6	RSH	*Rdst* = *Rsrc1* >> *Rsrc2*	Logical shift right

Table 2: ALU Operations Among Registers

Note:

- ADD or SUB operations can be used to set or clear the overflow flag in ALU.
- All ALU operations can be used to set or clear the zero flag in ALU.

Operations with Immediate Value

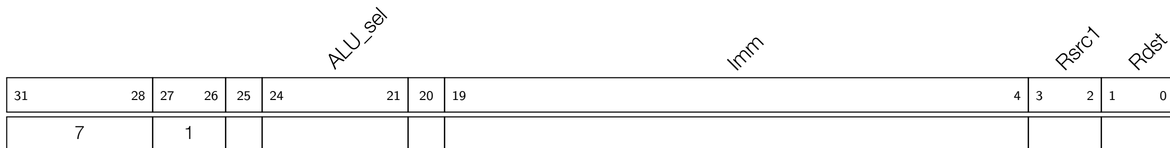


Figure 1-8. Instruction Type — ALU for Operations with Immediate Value

When bits [27:26] of the instruction in Figure 1-8 are set to 1, ALU performs operations using register R[0-3] and the immediate value stored in instruction bits [19:4]. The types of operations depend on the setting of the instruction bits ALU_sel[24:21] presented in Table 3.

Operand	**Description** - see Figure 1-8
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Imm	16-bit signed immediate value
ALU_sel	ALU Operation

ALU_sel	Instruction	Operation	Description
0	ADD	*Rdst* = *Rsrc1* + *Imm*	Add to register
1	SUB	*Rdst* = *Rsrc1* - *Imm*	Subtract from register
2	AND	*Rdst* = *Rsrc1* & *Imm*	Logical AND of two operands
3	OR	*Rdst* = *Rsrc1* | *Imm*	Logical OR of two operands
4	MOVE	*Rdst* = *Imm*	Move to register
5	LSH	*Rdst* = *Rsrc1* << *Imm*	Logical shift left
6	RSH	*Rdst* = *Rsrc1* >> *Imm*	Logical shift right

Table 3: ALU Operations with Immediate Value

Note:

- ADD or SUB operations can be used to set or clear the overflow flag in ALU.
- All ALU operations can be used to set or clear the zero flag in ALU.

Operations with Stage Count Register

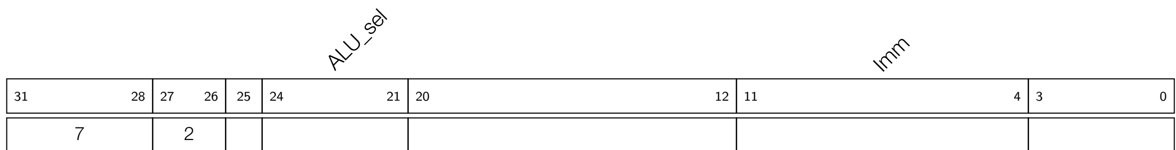


Figure 1-9. Instruction Type — ALU for Operations with Stage Count Register

ALU is also able to increment or decrement by a given value, or reset the 8-bit register Stage_cnt. To do so, bits [27:26] of instruction in Figure 1-9 should be set to 2. The type of operation depends on the setting of the instruction bits ALU_sel[24:21] presented in Table 1-9. The Stage_cnt is a separate register and is not a part of the instruction in Figure 1-9.

Operand	**Description** - see Figure 1-9
Imm	8-bit signed immediate value
ALU_sel	ALU Operation
Stage_cnt	Stage count register, a 8-bit separate register used to store variables, such as loop index

ALU_sel	Instruction	Operation	Description
0	STAGE_INC	*Stage_cnt* = *Stage_cnt* + *Imm*	Increment stage count register
1	STAGE_DEC	*Stage_cnt* = *Stage_cnt* - *Imm*	Decrement stage count register
2	STAGE_RST	*Stage_cnt* = 0	Reset stage count register

Table 4: ALU Operations with Stage Count Register

Note: This instruction is mainly used with JUMPS instruction based on the stage count register to form a stage count for-loop. For the usage, please refer to the following pseudocode:

STAGE_RST	// clear stage count register
STAGE_INC	// stage count register ++
{...}	// loop body, containing n instructions
JUMPS (step = n, cond = 0, threshold = m)	// If the value of stage count register is less than m, then jump to

STAGE_INC, otherwise jump out of the loop. By such way, a cumulative for-loop with threshold m is implemented.

ST – Store Data in Memory

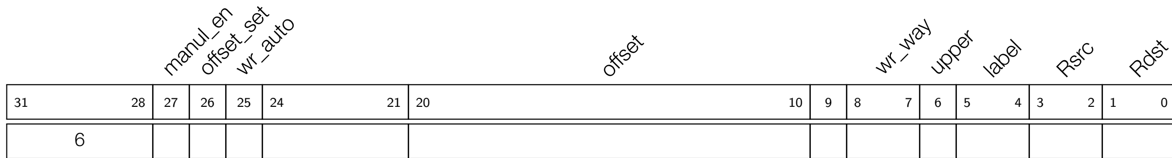


Figure 1-10. Instruction Type - ST

Operand	**Description** - see Figure 1-10
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
upper	0: write the low half-word; 1: write the high half-word
wr_way	0: write the full-word; 1: with the label; 3: without the label
offset	11-bit signed value, expressed in 32-bit words
wr_auto	Enable automatic storage mode
offset_set	Offset enable bit. 0: Do not configure the offset for automatic storage mode. 1: Configure the offset for automatic storage mode.
manul_en	Enable manual storage mode

Automatic Storage Mode

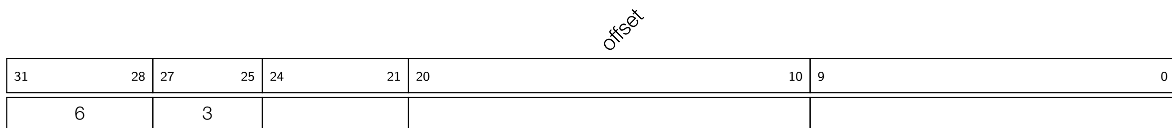


Figure 1-11. Instruction Type - Offset in Automatic Storage Mode (ST-OFFSET)

Operand	**Description** - see Figure 1-11
offset	Initial address offset, 11-bit signed value, expressed in 32-bit words

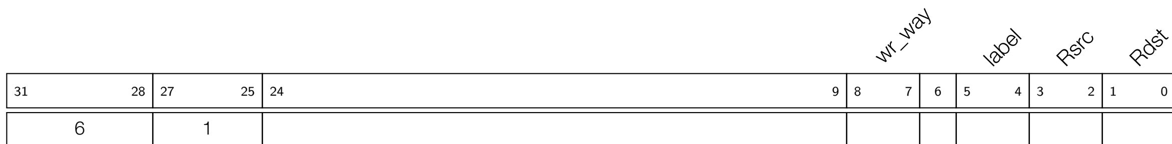


Figure 1-12. Instruction Type - Data Storage in Automatic Storage Mode (ST-AUTO-DATA)

Operand	Description - See Figure 1-12
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
wr_way	0: write the fullword; 1: with the label; 3: without the label

Description

This mode is used to access continuous addresses. Before using this mode for the first time, please configure the initial address using ST-OFFSET instruction. Executing the instruction ST-AUTO-DATA will store the 16-bit data in *Rsrc* into the memory address *Rdst* + *Offset*, see Table 5. Write_cnt here indicates the times of the instruction ST-AUTO-DATA executed.

wr_way	write_cnt	Store Data	Operation
0	*	Mem [Rdst + Offset]{31:0} ={PC[10:0], 3'b0, Label[1:0], Rsrc[15:0]}	Write full-word, including the pointer and the data
1	odd	Mem [Rdst + Offset]{15:0} = {Label[1:0],Rscr[13:0]}	Store the data with label in the low half-word
1	even	Mem [Rdst + Offset]{31:16} = {Label[1:0],Rscr[13:0]}	Store the data with label in the high half-word
3	odd	Mem [Rdst + Offset]{15:0} = Rscr[15:0]}	Store the data without label in the low half-word
3	even	Mem [Rdst + Offset]{31:16} = Rscr[15:0]	Store the data without label in the high half-word

Table 5: Data Storage Type - Automatic Storage Mode

The full-word written to RTC memory are built as follows:

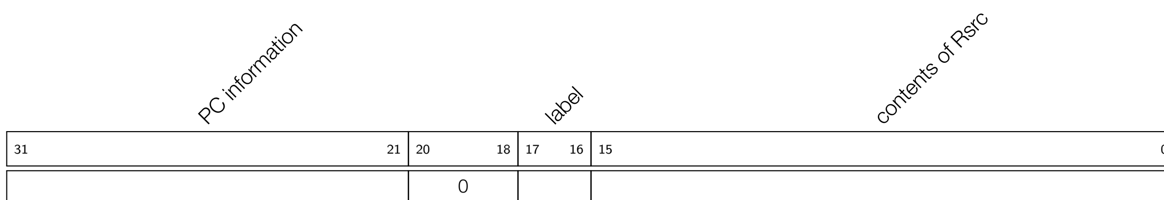


Figure 1-13. Data Structure of RTC_SLOW_MEM[Rdst + Offset]

Bits	Description - See Figure 1-13
bits [15:0]	store the content of Rsrc
bits [17:16]	data label, 2-bit user defined unsigned value
bits [20:18]	3'b0 by default
bits [31:21]	hold the PC of current instruction, expressed in 32-bit words

Note:

- When full-word is written, the offset will be automatically incremented by 1 after each ST-AUTO-DATA execution.
- When half-word is written (low half-word first), the offset will be automatically incremented by 1 after twice ST-AUTO-DATA execution.

- This instruction can only access 32-bit memory words.
- The "Mem" written is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPU.

Manual Storage Mode

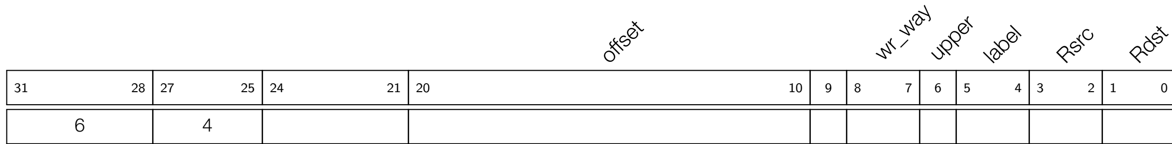


Figure 1-14. Instruction Type - Data Storage in Manual Storage Mode

Operand	**Description** - See Figure 1-14
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
upper	0: Write the low half-word; 1: write the high half-word
wr_way	0: Write the full-word; 1: with the label; 3: without the label
offset	11-bit signed value, expressed in 32-bit words

Description

Manual storage mode is mainly used for storing data into discontinuous addresses. Each instruction needs a storage address and offset. The detailed storage methods are shown in Table 6.

wr_way	upper	Data	Operation
0	*	Mem [Rdst + Offset]{31:0} ={PC[10:0], 3'b0, Label[1:0], Rsrc[15:0]}	Write full-word, including the pointer and the data
1	0	Mem [Rdst + Offset]{15:0} = {Label[1:0],Rscr[13:0]}	Store the data with label in the low half-word
1	1	Mem [Rdst + Offset]{31:16} = {Label[1:0],Rscr[13:0]}	Store the data with label in the high half-word
3	0	Mem [Rdst + Offset]{15:0} = Rsrc[15:0]	Store the data without label in the low half-word
3	1	Mem [Rdst + Offset]{31:16} = Rsrc[15:0]	Store the data without label in the high half-word

Table 6: Data Storage - Manual Storage Mode

LD – Load Data from Memory

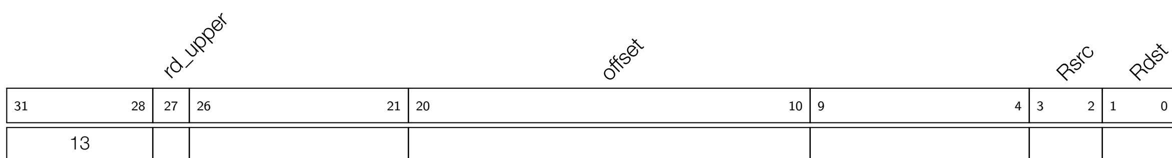


Figure 1-15. Instruction Type - LD

Operand	**Description** - see Figure 1-15
Rdst	Register R[0-3], destination
Rsrc	Register R[0-3], address of destination memory, expressed in 32-bit words
Offset	11-bit signed value, expressed in 32-bit words
rd_upper	Choose which half-word to read: 0 - read the high half-word 1 - read the low half-word

Description

This instruction loads the low or high 16-bit half-word, depending on *rd_upper*, from memory with address *Rsrc* + *offset* into the destination register *Rdst*:

Rdst[15:0] = Mem[Rsrc + Offset]

Note:

- This instruction can only access 32-bit memory words.
- The "Mem" loaded is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPU.

JUMP – Jump to an Absolute Address

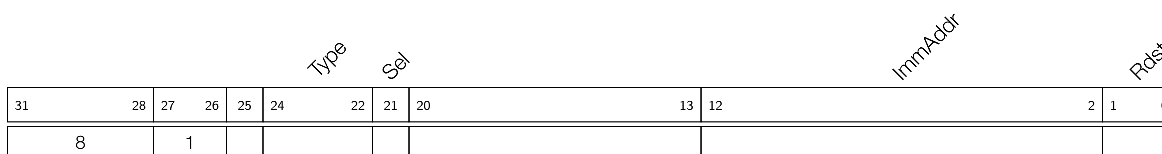


Figure 1-16. Instruction Type- JUMP

Operand	**Description** - see Figure 1-16
Rdst	Register R[0-3], containing address to jump to (expressed in 32-bit words)
ImmAddr	11-bit address, expressed in 32-bit words
Sel	Select the address to jump to: 0 - jump to the address stored in *ImmAddr* 1 - jump to the address stored in *Rdst*
Type	Jump type: 0 - make an unconditional jump 1 - jump only if the last ALU operation has set zero flag 2 - jump only if the last ALU operation has set overflow flag

Note:
All jump addresses are expressed in 32-bit words.

Description
The instruction executes a jump to a specified address. The jump can be either unconditional or based on the ALU flag.

JUMPR – Jump to a Relative Offset (Conditional upon R0)

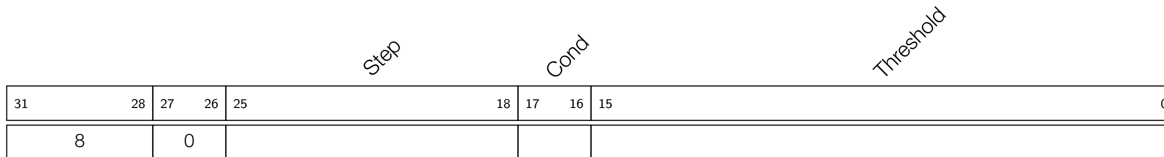


Figure 1-17. Instruction Type - JUMPR

Operand	**Description** - see Figure 1-17
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition to jump: 0 - jump if R0 < *Threshold* 1 - jump if R0 > *Threshold* 2 - jump if R0 = *Threshold*
Step	Relative shift from current position, expressed in 32-bit words: if Step[7] = 0, then PC = PC + Step[6:0] if Step[7] = 1, then PC = PC - Step[6:0]

Note:

All jump addresses are expressed in 32-bit words.

Description

The instruction executes a jump to a relative address, if the above-mentioned condition is true. The condition is the result of comparing the R0 register value and the *Threshold* value.

JUMPS – Jump to a Relative Address (Conditional upon Stage Count Register)

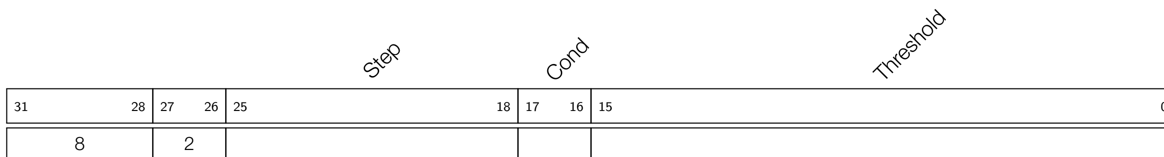


Figure 1-18. Instruction Type - JUMPS

Operand	**Description** - see Figure 1-18
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition to jump: 1X - jump if *Stage_cnt* <= *Threshold* 00 - jump if *Stage_cnt* < *Threshold* 01 - jump if *Stage_cnt* >= *Threshold*
Step	Relative shift from current position, expressed in 32-bit words: if Step[7] = 0, then PC = PC + Step[6:0] if Step[7] = 1, then PC = PC - Step[6:0]

Note:

- For more information about the stage count register, please refer to Section 1.5.2.

- All jump addresses are expresses in 32-bit words.

Description

The instruction executes a jump to a relative address if the above-mentioned condition is true. The condition itself is the result of comparing the value of *Stage_cnt* (stage count register) and the *Threshold* value.

HALT – End the Program

31 28	27 0
11	

Figure 1-19. Instruction Type- HALT

Description

The instruction ends the operation of the ULP-FSM and puts it into power-down mode.

Note:

After executing this instruction, the ULP coprocessor wakeup timer gets started.

WAKE – Wake up the Chip

31 28	27 26	25 1	0
9	0		1'b1

Figure 1-20. Instruction Type - WAKE

Description

This instruction sends an interrupt from the ULP-FSM to the RTC controller.

- If the chip is in Deep-sleep mode, and the ULP wakeup timer is enabled, the above-mentioned interrupt will wake up the chip.
- If the chip is not in Deep-sleep mode, and the ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in register RTC_CNTL_INT_ENA_REG, an RTC interrupt will be triggered.

WAIT – Wait for a Number of Cycles

31 28	27 16	15 0
4		Cycles

Figure 1-21. Instruction Type - WAIT

Operand	**Description** - see Figure 1-21
Cycles	The number of cycles to wait

Description

The instruction will delay the ULP-FSM for a given number of cycles.

TSENS – Take Measurement with Temperature Sensor

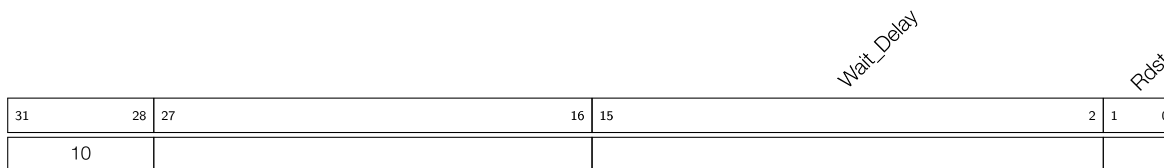


Figure 1-22. Instruction Type - TSENS

Operand	**Description** - see Figure 1-22
Rdst	Destination Register R[0-3], results will be stored in this register.
Wait_Delay	Number of cycles used to perform the measurement.

Description

Increasing the measurement cycles *Wait_Delay* helps improve the accuracy and optimize the result. The instruction performs measurement via temperature sensor and stores the result into a general purpose register.

ADC – Take Measurement with ADC

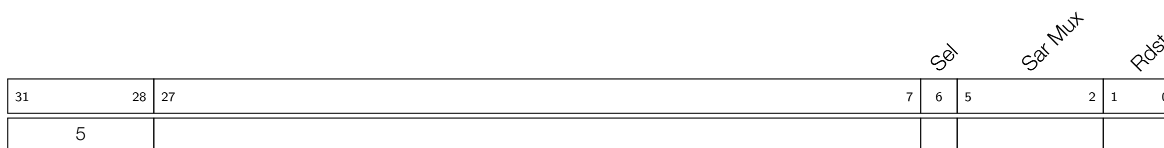


Figure 1-23. Instruction Type - ADC

Operand	**Description** - see Figure 1-23
Rdst	Destination Register R[0-3], results will be stored in this register.
Sar_Mux	Enable SAR ADC pad [Sar_Mux - 1], see Table 7.
Sel	Select ADC. 0: select SAR ADC1; 1: select SAR ADC2, see Table 7.

Table 7: Input Signals Measured Using the ADC Instruction

Pad / Signal / GPIO	*Sar_Mux*	ADC Selection *Sel*
GPIO1	1	*Sel* = 0, select SAR ADC1
GPIO2	2	
GPIO3	3	
GPIO4	4	
GPIO5	5	
GPIO6	6	
GPIO7	7	
GPIO8	8	
GPIO9	9	
GPIO10	10	
GPIO11	1	*Sel* = 1, select SAR ADC2
GPIO12	2	

Pad / Signal / GPIO	*Sar_Mux*	ADC Selection *Sel*
GPIO13	3	*Sel* = 1, select SAR ADC2
GPIO14	4	
XTAL_32k_P	5	
XTAL_32k_N	6	
DAC_1	7	
DAC_2	8	
GPIO19	9	
GPIO20	10	

REG_RD – Read from Peripheral Register

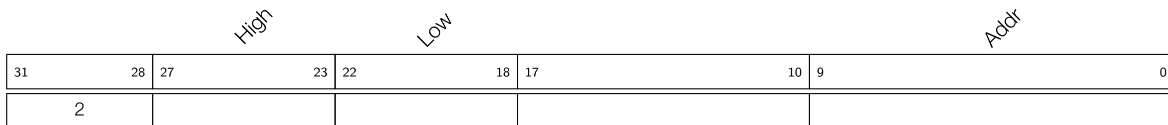


Figure 1-24. Instruction Type - REG_RD

Operand	**Description** - see Figure 1-24
Addr	Peripheral register address, in 32-bit words
Low	Register start bit number
High	Register end bit number

Description

The instruction reads up to 16 bits from a peripheral register into a general-purpose register:

R0 = REG[Addr][High:Low]

In case of more than 16 bits being requested, i.e. *High* - *Low* + 1 > 16, then the instruction will return [Low+15:Low].

Note:

- This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the register, as seen from the ULP coprocessor, can be calculated from the address of the same register on PeriBUS1 (addr_peribus1), as follows:

  addr_ulp = (addr_peribus1 - DR_REG_RTCCNTL_BASE) / 4

- The *addr_ulp* is expressed in 32-bit words (not in bytes), and value 0 maps onto the DR_REG_RTCCNTL_BASE (as seen from the main CPU). Thus, 10 bits of address cover a 4096-byte range of peripheral register space, including regions DR_REG_RTCCNTL_BASE, DR_REG_RTCIO_BASE, DR_REG_SENS_BASE, and DR_REG_RTC_I2C_BASE.

REG_WR – Write to Peripheral Register

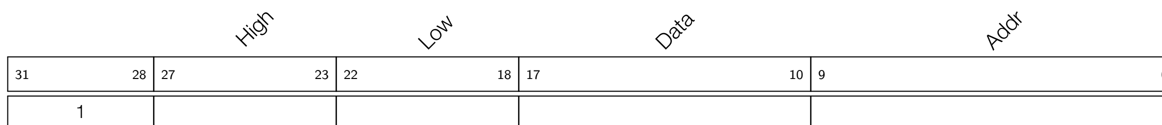


Figure 1-25. Instruction Type - REG_WR

Operand	**Description** - see Figure 1-25
Addr	Register address, expressed in 32-bit words
Data	Value to write, 8 bits
Low	Register start bit number
High	Register end bit number

Description

This instruction writes up to 8 bits from an immediate data value into a peripheral register.

REG[Addr][High:Low] = Data

If more than 8 bits are requested, i.e. *High* - *Low* + 1 > 8, then the instruction will pad with zeros the bits above the eighth bit.

Note:

See notes regarding *addr_ulp* in Section 1.5.2.

1.6 ULP-RISC-V

1.6.1 Features

- Support RV32IMC instruction set
- Thirty-two 32-bit general-purpose registers
- 32-bit multiplier and divider
- Support for interrupts

1.6.2 Multiplier and Divider

ULP-RISC-V has an independent multiplication and division unit. The efficiency of multiplication and division instructions is shown in the following table.

Table 8: Instruction Efficiency

Operation	Instruction	Execution Cycle	Instruction Description
Multiply	MUL	34	Multiply two 32-bit integers and return the lower 32-bit of the result
	MULH	66	Multiply two 32-bit signed integers and return the higher 32-bit of the result
	MULHU	66	Multiply two 32-bit unsigned integers and return the higher 32-bit of the result
	MULHSU	66	Multiply a 32-bit signed integer with a unsigned integer and return the higher 32-bit of the result
Divide	DIV	34	Divide a 32-bit integer by a 32-bit integer and return the quotient
	DIVU	34	Divide a 32-bit unsigned integer by a 32-bit unsigned integer and return the quotient
	REM	34	Divide a 32-bit signed integer by a 32-bit signed integer and return the remainder
	REMU	34	Divide a 32-bit unsigned integer by a 32-bit unsigned integer and return the remainder

1.6.3 ULP-RISC-V Interrupts

The interrupts from some sensors, software, and RTC I2C can be routed to ULP-RISC-V. To enable the interrupts, please set the register SENS_SAR_COCPU_INT_ENA_REG, see Table 9.

Enable bit	Interrupt	Description
0	TOUCH_DONE_INT	Triggered when the touch sensor completes the scan of a channel
1	TOUCH_INACTIVE_INT	Triggered when the touch pad is released
2	TOUCH_ACTIVE_INT	Triggered when the touch pad is touched
3	SARADC1_DONE_INT	Triggered when SAR ADC1 completes the conversion one time
4	SARADC2_DONE_INT	Triggered when SAR ADC2 completes the conversion one time
5	TSENS_DONE_INT	Triggered when the temperature sensor completes the dump of its data
6	RISCV_START_INT	Triggered when ULP-RISC-V powers on and starts working
7	SW_INT	Triggered by software
8	SWD_INT	Triggered by timeout of Super Watchdog (SWD)

Table 9: ULP-RISC-V Interrupt List

Note:

- Besides the above-mentioned interrupts, ULP-RISC-V can also handle the interrupt from RTC_IO by simply configuring RTC_IO as input mode. Users can configure RTCIO_GPIO_PIN*n*_INT_TYPE to select the interrupt trigger modes, but only level trigger modes are available. For more details about RTC_IO configuration, see Chapter IO MUX and GPIO Matrix.
- The interrupt from RTC_IO can be cleared by releasing RTC_IO and its source can be read from the register RTCIO_RTC_GPIO_STATUS_REG.
- The SW_INT interrupt is generated by configuring the register RTC_CNTL_COCPU_SW_INT_TRIGGER.
- For the information about RTC I2C interrupts, please refer to Section 1.7.4.

1.7 RTC I2C Controller

ULP coprocessor can use RTC I2C controller to read from or write to the external I2C slave devices.

1.7.1 Connecting RTC I2C Signals

SDA and SCL signals can be mapped onto two out of the four GPIO pins, which are identified in Table 38 in Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*, using the register RTCIO_SAR_I2C_IO_REG.

1.7.2 Configuring RTC I2C

Before the ULP coprocessor can use the I2C instruction, certain parameters of the RTC I2C need to be configured. Configuration is performed by writing certain timing parameters into the RTC I2C registers. This can be done by the program running on the main CPU, or by the ULP coprocessor itself.

1. Set the low and high SCL half-periods by configuring RTC_I2C_SCL_LOW_PERIOD_REG and RTC_I2C_SCL_HIGH_PERIOD_REG in RTC_FAST_CLK cycles (e.g. RTC_I2C_SCL_LOW_PERIOD_REG = 40, RTC_I2C_SCL_HIGH_PERIOD_REG = 40 for 100 kHz frequency).
2. Set the number of cycles between the SDA switch and the falling edge of SCL by using RTC_I2C_SDA_DUTY_REG in RTC_FAST_CLK (e.g. RTC_I2C_SDA_DUTY_REG = 16).
3. Set the waiting time after the START condition by using RTC_I2C_SCL_START_PERIOD_REG (e.g. RTC_I2C_SCL_START_PERIOD = 30).
4. Set the waiting time before the END condition by using RTC_I2C_SCL_STOP_PERIOD_REG (e.g. RTC_I2C_SCL_STOP_PERIOD = 44).
5. Set the transaction timeout by using RTC_I2C_TIME_OUT_REG (e.g. RTC_I2C_TIME_OUT_REG = 200).
6. Configure the RTC I2C controller into master mode by setting the RTC_I2C_MS_MODE bit in RTC_I2C_CTRL_REG.
7. Write the address(es) of external slave(s) to SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7). Up to eight slave addresses can be pre-programmed this way. One of these addresses can then be selected for each transaction as part of the RTC I2C instruction.

Once RTC I2C is configured, the main CPU or the ULP coprocessor can communicate with the external I2C devices.

1.7.3 Using RTC I2C

Instruction Format

The format of RTC I2C instruction is consistent with that of I2C0/I2C1, see Section 25.3.2.2 *CMD_Controller* in Chapter 25 *I2C Controller (I2C)*. The only difference is that RTC I2C provides fixed instructions for different operations, as follows:

- Command 0 ~ Command 1: specially for I2C write operation
- Command 2 ~ Command 6: specially for I2C read operation

Note: All slave addresses are expressed in 7 bits.

I2C_RD - I2C Read Workflow

Preparation for RTC I2C read:

- Configure the instruction list of RTC I2C (see Section CMD_Controller in Chapter 25 *I2C Controller (I2C)*), including instruction order, instruction code, read data number (byte_num), and other information.
- Configure the slave register address by setting the register SENS_SAR_I2C_CTRL[18:11].
- Start RTC I2C transmission by setting the registers SENS_SAR_I2C_START_FORCE and SENS_SAR_I2C_START.
- When an RTC_I2C_RX_DATA_INT interrupt is received, transfer the read data stored in RTC_I2C_RDATA to SRAM RTC slow memory, or use the data directly.

The I2C_RD instruction performs the following operations (see Figure 1-26):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 (“write”). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*.
3. Slave generates ACK.
4. Master sends slave register address.
5. Slave generates ACK.
6. Master generates a repeated START (RSTART) condition.
7. Master sends slave address, with r/w bit set to 1 (“read”).
8. Slave sends one byte of data.
9. Master checks whether the number of transmitted bytes reaches the number set by the current instruction (byte_num). If yes, master jumps out of the read instruction and sends an NACK signal. Otherwise master repeats Step 8 and waits for the slave to send the next byte.
10. Master generates a STOP condition and stops reading.

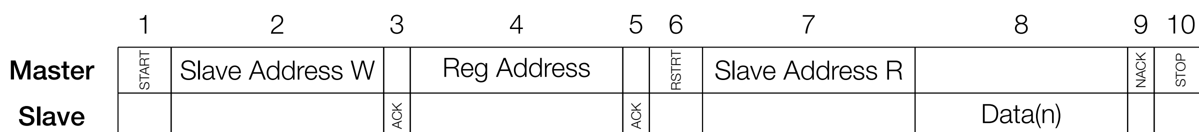


Figure 1-26. I2C Read Operation

Note:
The RTC I2C peripheral samples the SDA signals on the falling edge of SCL. If the slave changes SDA in less than 0.38 microsecond, the master may receive incorrect data.

I2C_WR - I2C Write Workflow

Preparation for RTC I2C write:

- Configure RTC I2C instruction list, including instruction order, instruction code, and the data to be written in byte (byte_num). See the configuration of I2C0/I2C1 in Section CMD_Controller in Chapter 25 *I2C Controller (I2C)*.
- Configure the slave register address by setting the register SENS_SAR_I2C_CTRL[18:11], and the data to be transmitted in SENS_SAR_I2C_CTRL[26:19].

- Set the registers SENS_SAR_I2C_START_FORCE and SENS_SAR_I2C_START to start the transmission.
- Update the next data to be transmitted in the register SENS_SAR_I2C_CTRL[26:19], each time when an RTC_I2C_TX_DATA_INT interrupt is received.

The I2C_WR instruction performs the following operations, see Figure 1-27.

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 ("write"). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*.
3. Slave generates ACK.
4. Master sends slave register address.
5. Slave generates ACK.
6. Master generates a repeated START condition (RSTART).
7. Master sends slave address, with r/w bit set to 0 ("write").
8. Master sends one byte of data.
9. Slave generates ACK. Master checks whether the number of transmitted bytes reaches the number set by the current instruction (byte_num). If yes, master jumps out of the write instruction and starts the next instruction. Otherwise the master repeats step 8 and sends the next byte.
10. Master generates a STOP condition and stops the transmission.

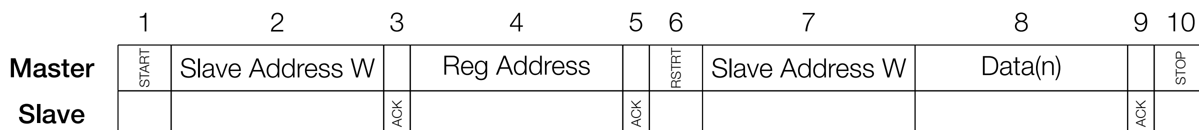


Figure 1-27. I2C Write Operation

Detecting Error Conditions

Applications can query specific bits in the RTC_I2C_INT_ST_REG register to check if the transaction is successful. To enable checking for specific communication events, their corresponding bits should be set in register RTC_I2C_INT_ENA_REG. Note that the bit map is shifted by 1. If a specific communication event is detected and its corresponding bit in register RTC_I2C_INT_ST_REG is set, the event can then be cleared using register RTC_I2C_INT_CLR_REG.

1.7.4 RTC I2C Interrupts

- RTC_I2C_SLAVE_TRAN_COMP_INT: Triggered when the slave finishes the transaction.
- RTC_I2C_ARBITRATION_LOST_INT: Triggered when the master loses control of the bus.
- RTC_I2C_MASTER_TRAN_COMP_INT: Triggered when the master completes the transaction.
- RTC_I2C_TRANS_COMPLETE_INT: Triggered when a STOP signal is detected.
- RTC_I2C_TIME_OUT_INT: Triggered by time out event.
- RTC_I2C_ACK_ERR_INT: Triggered by ACK error.

- RTC_I2C_RX_DATA_INT: Triggered when data is received.
- RTC_I2C_TX_DATA_INT: Triggered when data is transmitted.
- RTC_I2C_DETECT_START_INT: Triggered when a START signal is detected.

1.8 Base Address

1.8.1 ULP Coprocessor Base Address

Users can access ULP coprocessors with two base addresses, which can be seen in Table 10. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 10: ULP Coprocessor Base Address

Module	Bus to Access Peripheral	Base Address
ULP (ALWAYS_ON)	PeriBUS1	0x3F408000
	PeriBUS2	0x60008000
ULP (RTC_PERI)	PeriBUS1	0x3F408800
	PeriBUS2	0x60008800

Wherein:

- ULP (ALWAYS_ON) represents the registers, which will not be reset due to the power down of RTC_PERI domain. See Chapter 9 *Low-Power Management (RTC_CNTL)*.
- ULP (RTC_PERI) represents the registers in RTC_PERI domain, which will be reset due to the power down of RTC_PERI domain. See Chapter 9 *Low-Power Management (RTC_CNTL)*.

1.8.2 RTC I2C Base Address

Users can access the RTC I2C registers in RTC_PERI domain, including RTC_PERI registers and I2C registers, with two base addresses, which can be seen in Table 11. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 11: RTC I2C Base Address

Module	Bus to Access Peripheral	Base Address
RTC I2C (RTC_PERI)	PeriBUS1	0x3F408800
	PeriBUS2	0x60008800
RTC I2C (I2C)	PeriBUS1	0x3F408C00
	PeriBUS2	0x60008C00

1.9 Register Summary

The address in the following part represents the address offset (relative address) with respect to the base address, not the absolute address. For detailed information about the base address, please refer to Section 1.8.

1.9.1 ULP (ALWAYS_ON) Register Summary

Name	Description	Address	Access
ULP Timer Registers			
RTC_CNTL_ULP_CP_TIMER_REG	Configure the timer	0x00F8	varies

Name	Description	Address	Access
RTC_CNTL_ULP_CP_TIMER_1_REG	Configure sleep cycle of the timer	0x0130	R/W
ULP-FSM Register			
RTC_CNTL_ULP_CP_CTRL_REG	ULP-FSM configuration register	0x00FC	R/W
ULP-RISC-V Register			
RTC_CNTL_COCPU_CTRL_REG	ULP-RISC-V configuration register	0x0100	varies

1.9.2 ULP (RTC_PERI) Register Summary

Name	Description	Address	Access
ULP-RISC-V Registers			
SENS_ SAR_COCPU_INT_RAW_REG	Interrupt raw bit of ULP-RISC-V	0x0128	RO
SENS_SAR_COCPU_INT_ENA_REG	Interrupt enable bit of ULP-RISC-V	0x012C	R/W
SENS_SAR_COCPU_INT_ST_REG	Interrupt status bit of ULP-RISC-V	0x0130	RO
SENS_SAR_COCPU_INT_CLR_REG	Interrupt clear bit of ULP-RISC-V	0x0134	WO

1.9.3 RTC I2C (RTC_PERI) Register Summary

Name	Description	Address	Access
RTC I2C Controller Register			
SENS_SAR_I2C_CTRL_REG	Configure RTC I2C transmission	0x0058	R/W
RTC I2C Slave Address Registers			
SENS_SAR_SLAVE_ADDR1_REG	Configure slave addresses 0-1 of RTC I2C	0x0040	R/W
SENS_SAR_SLAVE_ADDR2_REG	Configure slave addresses 2-3 of RTC I2C	0x0044	R/W
SENS_SAR_SLAVE_ADDR3_REG	Configure slave addresses 4-5 of RTC I2C	0x0048	R/W
SENS_SAR_SLAVE_ADDR4_REG	Configure slave addresses 6-7 of RTC I2C	0x004C	R/W

1.9.4 RTC I2C (I2C) Register Summary

Name	Description	Address	Access
RTC I2C Signal Setting Registers			
RTC_I2C_SCL_LOW_REG	Configure the low level width of SCL	0x0000	R/W
RTC_I2C_SCL_HIGH_REG	Configure the high level width of SCL	0x0014	R/W
RTC_I2C_SDA_DUTY_REG	Configure the SDA hold time after a negative SCL edge	0x0018	R/W
RTC_I2C_SCL_START_PERIOD_REG	Configure the delay between the SDA and SCL negative edge for a start condition	0x001C	R/W
RTC_I2C_SCL_STOP_PERIOD_REG	Configure the delay between SDA and SCL positive edge for a stop condition	0x0020	R/W
RTC I2C Control Registers			
RTC_I2C_CTRL_REG	Transmission setting	0x0004	R/W
RTC_I2C_STATUS_REG	RTC I2C status	0x0008	RO
RTC_I2C_TO_REG	Configure RTC I2C timeout	0x000C	R/W
RTC_I2C_SLAVE_ADDR_REG	Configure slave address	0x0010	R/W
RTC I2C Interrupt Registers			

Name	Description	Address	Access
RTC_I2C_INT_CLR_REG	Clear RTC I2C interrupt	0x0024	WO
RTC_I2C_INT_RAW_REG	RTC I2C raw interrupt	0x0028	RO
RTC_I2C_INT_ST_REG	RTC I2C interrupt status	0x002C	RO
RTC_I2C_INT_ENA_REG	Enable RTC I2C interrupt	0x0030	R/W
RTC I2C Status Register			
RTC_I2C_DATA_REG	RTC I2C read data	0x0034	varies
RTC I2C Command Registers			
RTC_I2C_CMD0_REG	RTC I2C Command 0	0x0038	varies
RTC_I2C_CMD1_REG	RTC I2C Command 1	0x003C	varies
RTC_I2C_CMD2_REG	RTC I2C Command 2	0x0040	varies
RTC_I2C_CMD3_REG	RTC I2C Command 3	0x0044	varies
RTC_I2C_CMD4_REG	RTC I2C Command 4	0x0048	varies
RTC_I2C_CMD5_REG	RTC I2C Command 5	0x004C	varies
RTC_I2C_CMD6_REG	RTC I2C Command 6	0x0050	varies
RTC_I2C_CMD7_REG	RTC I2C Command 7	0x0054	varies
RTC_I2C_CMD8_REG	RTC I2C Command 8	0x0058	varies
RTC_I2C_CMD9_REG	RTC I2C Command 9	0x005C	varies
RTC_I2C_CMD10_REG	RTC I2C Command 10	0x0060	varies
RTC_I2C_CMD11_REG	RTC I2C Command 11	0x0064	varies
RTC_I2C_CMD12_REG	RTC I2C Command 12	0x0068	varies
RTC_I2C_CMD13_REG	RTC I2C Command 13	0x006C	varies
RTC_I2C_CMD14_REG	RTC I2C Command 14	0x0070	varies
RTC_I2C_CMD15_REG	RTC I2C Command 15	0x0074	varies
Version register			
RTC_I2C_DATE_REG	Version control register	0x00FC	R/W

1.10 Registers

The address in the following part represents the address offset (relative address) with respect to the base address, not the absolute address. For detailed information about the base address, please refer to Section 1.8.

1.10.1 ULP (ALWAYS_ON) Registers

Register 1.1: RTC_CNTL_ULP_CP_TIMER_REG (0x00F8)

RTC_CNTL_ULP_CP_SLP_TIMER_EN	RTC_CNTL_ULP_CP_GPIO_WAKEUP_CLR	RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA	(reserved)	RTC_CNTL_ULP_CP_PC_INIT	
31	30	29	28 ... 11	10 ... 0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RTC_CNTL_ULP_CP_PC_INIT ULP coprocessor PC initial address. (R/W)

RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA Enable the option of ULP coprocessor woken up by RTC GPIO. (R/W)

RTC_CNTL_ULP_CP_GPIO_WAKEUP_CLR Disable the option of ULP coprocessor woken up by RTC GPIO. (WO)

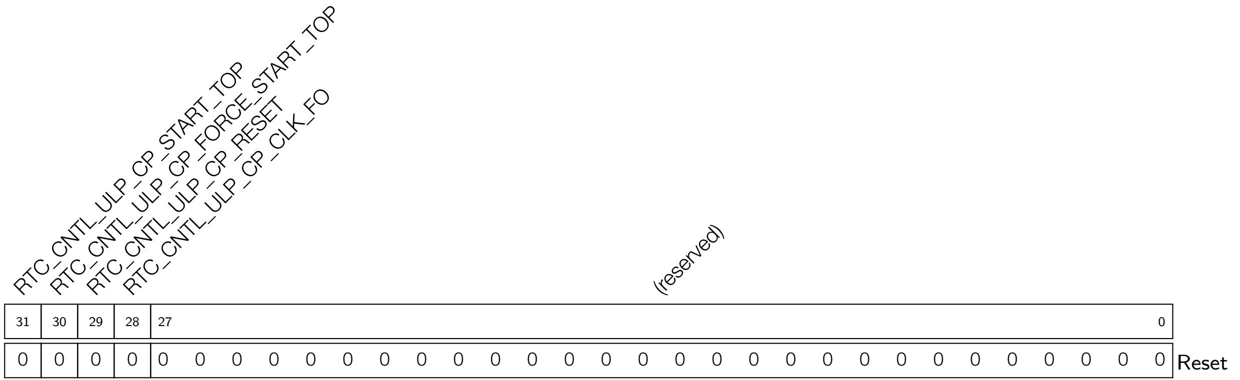
RTC_CNTL_ULP_CP_SLP_TIMER_EN ULP coprocessor timer enable bit. 0: Disable hardware timer; 1: Enable hardware timer. (R/W)

Register 1.2: RTC_CNTL_ULP_CP_TIMER_1_REG (0x0130)

RTC_CNTL_ULP_CP_TIMER_SLP_CYCLE	(reserved)	
31 ... 8	7 ... 0	
200	0 0 0 0 0 0 0 0	Reset

RTC_CNTL_ULP_CP_TIMER_SLP_CYCLE Set sleep cycles for ULP coprocessor timer. (R/W)

Register 1.3: RTC_CNTL_ULP_CP_CTRL_REG (0x00FC)



RTC_CNTL_ULP_CP_CLK_FO ULP-FSM clock force on. (R/W)

RTC_CNTL_ULP_CP_RESET ULP-FSM clock software reset. (R/W)

RTC_CNTL_ULP_CP_FORCE_START_TOP Write 1 to start ULP-FSM by software. (R/W)

RTC_CNTL_ULP_CP_START_TOP Write 1 to start ULP-FSM. (R/W)

Register 1.4: RTC_CNTL_COCPU_CTRL_REG (0x0100)

(reserved)	RTC_CNTL_COCPU_SW_INT_TRIGGER	RTC_CNTL_COCPU_DONE	RTC_CNTL_COCPU_DONE_FORCE	RTC_CNTL_COCPU_SEL	RTC_CNTL_COCPU_SHUT_RESET_EN	RTC_CNTL_COCPU_SHUT_2_CLK_DIS	RTC_CNTL_COCPU_SHUT	RTC_CNTL_COCPU_START_2_INTR_EN	RTC_CNTL_COCPU_START_2_RESET_DIS	RTC_CNTL_COCPU_CLK_FO	
31 27	26	25	24	23	22	21 14	13	12 7	6 1	0	
0 0 0 0 0	0	0	0	1	0	40	0	16	8	0	Reset

RTC_CNTL_COCPU_CLK_FO ULP-RISC-V clock force on. (R/W)

RTC_CNTL_COCPU_START_2_RESET_DIS Time from ULP-RISC-V startup to pull down reset. (R/W)

RTC_CNTL_COCPU_START_2_INTR_EN Time from ULP-RISC-V startup to send out RISCV_START_INT interrupt. (R/W)

RTC_CNTL_COCPU_SHUT Shut down ULP-RISC-V. (R/W)

RTC_CNTL_COCPU_SHUT_2_CLK_DIS Time from shut down ULP-RISC-V to disable clock. (R/W)

RTC_CNTL_COCPU_SHUT_RESET_EN This bit is used to reset ULP-RISC-V. (R/W)

RTC_CNTL_COCPU_SEL 0: select ULP-RISC-V; 1: select ULP-FSM. (R/W)

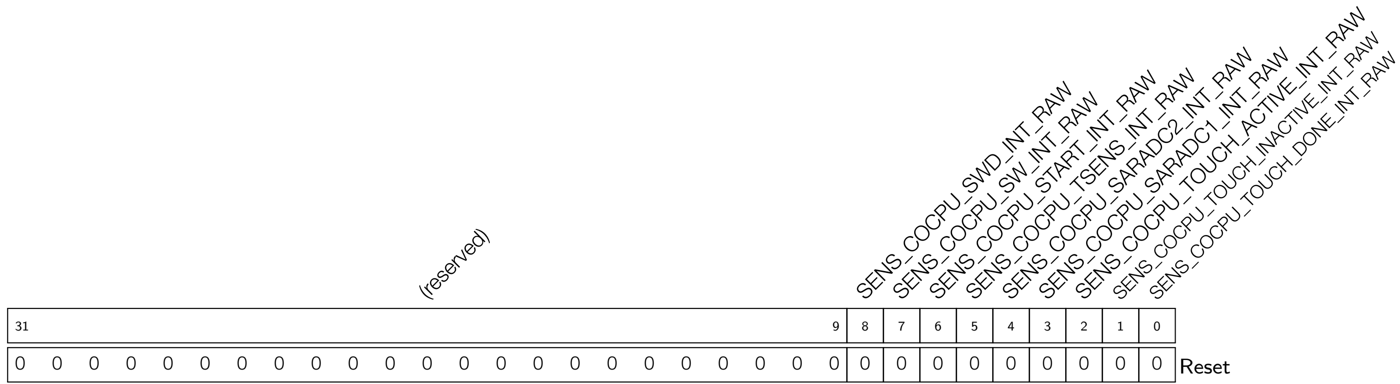
RTC_CNTL_COCPU_DONE_FORCE 0: select ULP-FSM DONE signal; 1: select ULP-RISC-V DONE signal. (R/W)

RTC_CNTL_COCPU_DONE DONE signal. Write 1 to this bit, ULP-RISC-V will go to HALT and the timer starts counting. (R/W)

RTC_CNTL_COCPU_SW_INT_TRIGGER Trigger ULP-RISC-V register interrupt. (WO)

1.10.2 ULP (RTC_PERI) Registers

Register 1.5: SENS_ SAR_COCPU_INT_RAW_REG (0x0128)



SENS_COCPU_TOUCH_DONE_INT_RAW TOUCH_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_TOUCH_INACTIVE_INT_RAW TOUCH_INACTIVE_INT interrupt raw bit. (RO)

SENS_COCPU_TOUCH_ACTIVE_INT_RAW TOUCH_ACTIVE_INT interrupt raw bit. (RO)

SENS_COCPU_SARADC1_INT_RAW SARADC1_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_SARADC2_INT_RAW SARADC2_DONE_INT interrupt raw bit. (RO)

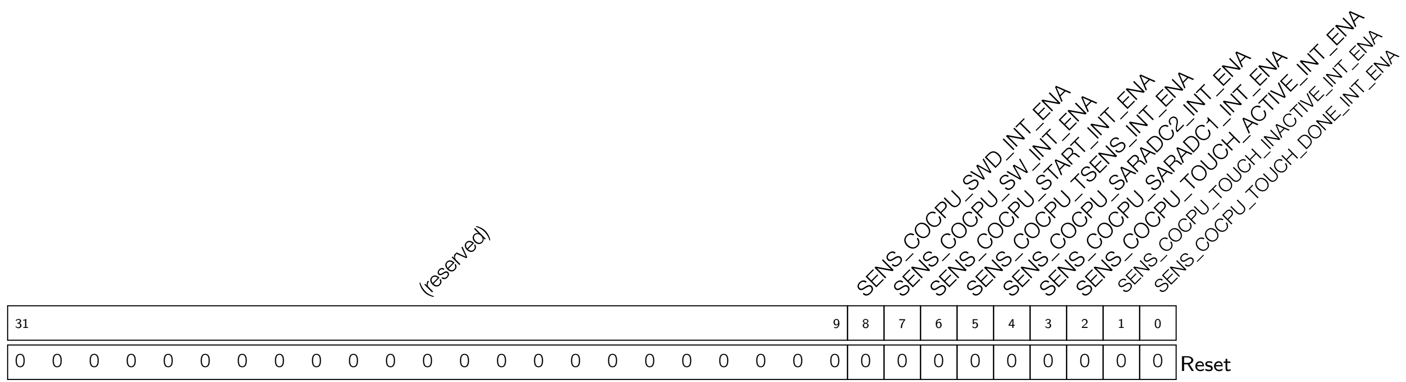
SENS_COCPU_TSENS_INT_RAW TSENS_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_START_INT_RAW RISCV_START_INT interrupt raw bit. (RO)

SENS_COCPU_SW_INT_RAW SW_INT interrupt raw bit. (RO)

SENS_COCPU_SWD_INT_RAW SWD_INT interrupt raw bit. (RO)

Register 1.6: SENS_SAR_COCPU_INT_ENA_REG (0x012C)



SENS_COCPU_TOUCH_DONE_INT_ENA TOUCH_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_TOUCH_INACTIVE_INT_ENA TOUCH_INACTIVE_INT interrupt enable bit. (R/W)

SENS_COCPU_TOUCH_ACTIVE_INT_ENA TOUCH_ACTIVE_INT interrupt enable bit. (R/W)

SENS_COCPU_SARADC1_INT_ENA SARADC1_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_SARADC2_INT_ENA SARADC2_DONE_INT interrupt enable bit. (R/W)

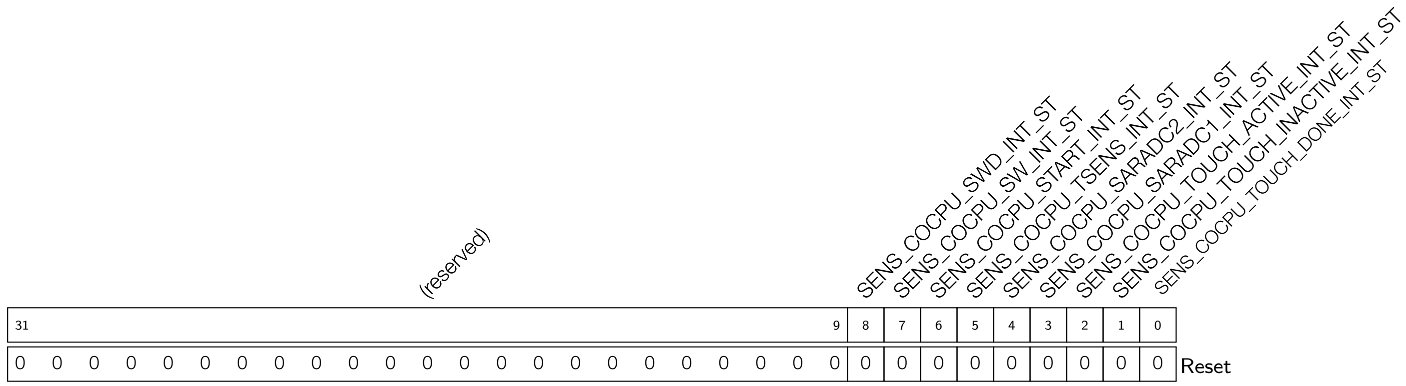
SENS_COCPU_TSENS_INT_ENA TSENS_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_START_INT_ENA RISCV_START_INT interrupt enable bit. (R/W)

SENS_COCPU_SW_INT_ENA SW_INT interrupt enable bit. (R/W)

SENS_COCPU_SWD_INT_ENA SWD_INT interrupt enable bit. (R/W)

Register 1.7: SENS_SAR_COCPU_INT_ST_REG (0x0130)



SENS_COCPU_TOUCH_DONE_INT_ST TOUCH_DONE_INT interrupt status bit. (RO)

SENS_COCPU_TOUCH_INACTIVE_INT_ST TOUCH_INACTIVE_INT interrupt status bit. (RO)

SENS_COCPU_TOUCH_ACTIVE_INT_ST TOUCH_ACTIVE_INT interrupt status bit. (RO)

SENS_COCPU_SARADC1_INT_ST SARADC1_DONE_INT interrupt status bit. (RO)

SENS_COCPU_SARADC2_INT_ST SARADC2_DONE_INT interrupt status bit. (RO)

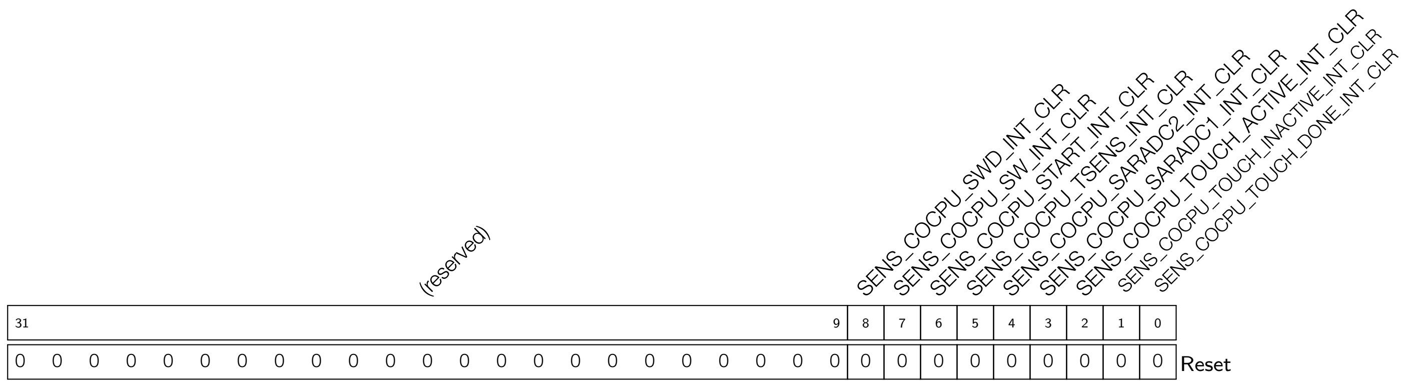
SENS_COCPU_TSENS_INT_ST TSENS_DONE_INT interrupt status bit. (RO)

SENS_COCPU_START_INT_ST RISCV_START_INT interrupt status bit. (RO)

SENS_COCPU_SW_INT_ST SW_INT interrupt status bit. (RO)

SENS_COCPU_SWD_INT_ST SWD_INT interrupt status bit. (RO)

Register 1.8: SENS_SAR_COCPU_INT_CLR_REG (0x0134)



SENS_COCPU_TOUCH_DONE_INT_CLR TOUCH_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_TOUCH_INACTIVE_INT_CLR TOUCH_INACTIVE_INT interrupt clear bit. (WO)

SENS_COCPU_TOUCH_ACTIVE_INT_CLR TOUCH_ACTIVE_INT interrupt clear bit. (WO)

SENS_COCPU_SARADC1_INT_CLR SARADC1_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_SARADC2_INT_CLR SARADC2_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_TSENS_INT_CLR TSENS_DONE_INT interrupt clear bit. (WO)

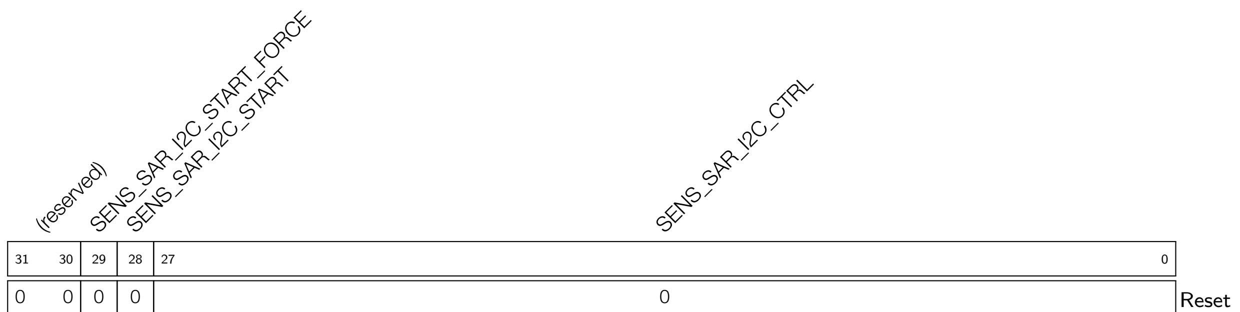
SENS_COCPU_START_INT_CLR RISCV_START_INT interrupt clear bit. (WO)

SENS_COCPU_SW_INT_CLR SW_INT interrupt clear bit. (WO)

SENS_COCPU_SWD_INT_CLR SWD_INT interrupt clear bit. (WO)

1.10.3 RTC I2C (RTC_PERI) Registers

Register 1.9: SENS_SAR_I2C_CTRL_REG (0x0058)

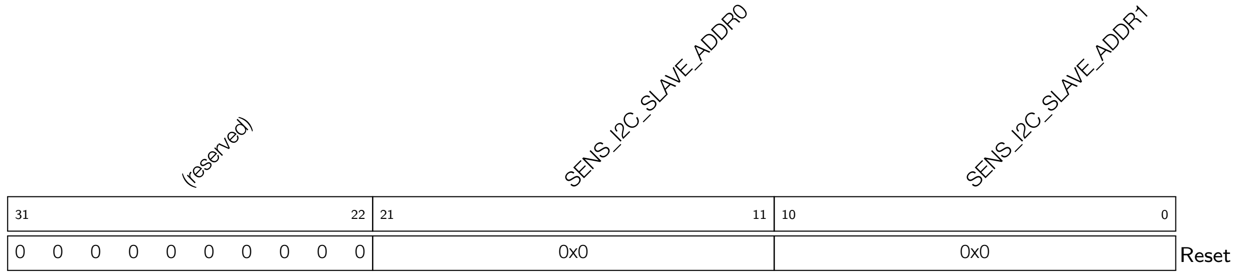


SENS_SAR_I2C_CTRL RTC I2C control data; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

SENS_SAR_I2C_START Start RTC I2C; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

SENS_SAR_I2C_START_FORCE 0: RTC I2C started by FSM; 1: RTC I2C started by software. (R/W)

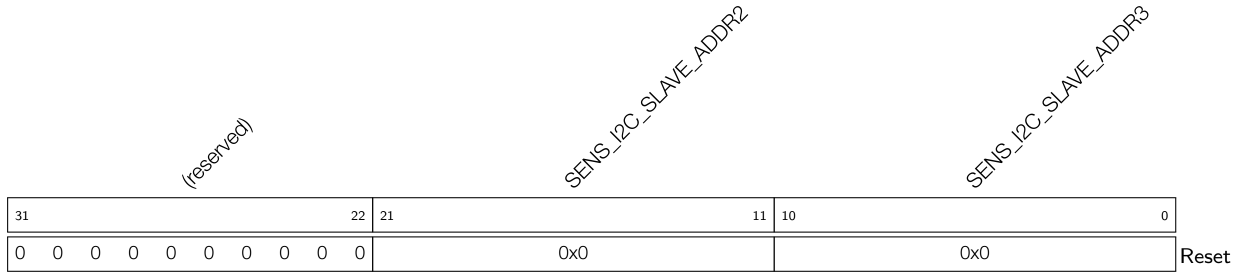
Register 1.10: SENS_SAR_SLAVE_ADDR1_REG (0x0040)



SENS_I2C_SLAVE_ADDR1 RTC I2C slave address 1. (R/W)

SENS_I2C_SLAVE_ADDR0 RTC I2C slave address 0. (R/W)

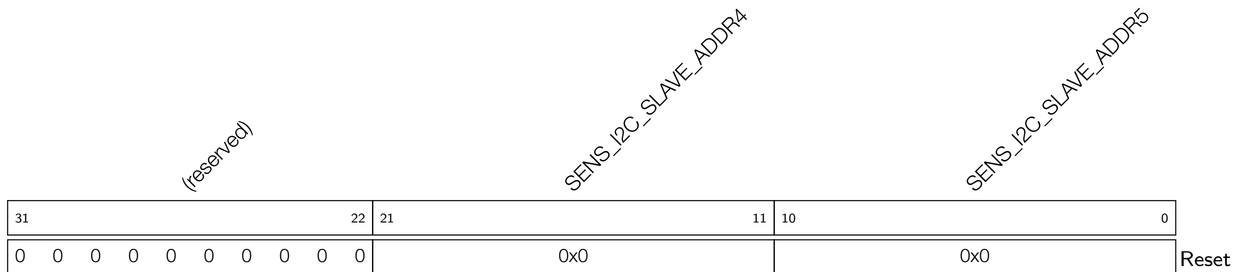
Register 1.11: SENS_SAR_SLAVE_ADDR2_REG (0x0044)



SENS_I2C_SLAVE_ADDR3 RTC I2C slave address 3. (R/W)

SENS_I2C_SLAVE_ADDR2 RTC I2C slave address 2. (R/W)

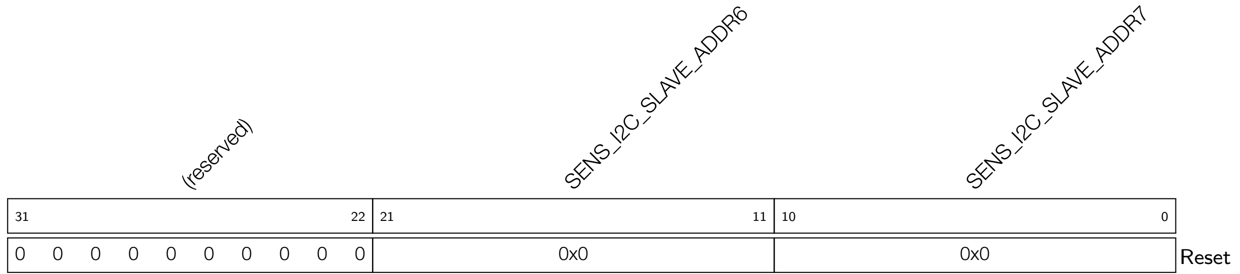
Register 1.12: SENS_SAR_SLAVE_ADDR3_REG (0x0048)



SENS_I2C_SLAVE_ADDR5 RTC I2C slave address 5. (R/W)

SENS_I2C_SLAVE_ADDR4 RTC I2C slave address 4. (R/W)

Register 1.13: SENS_SAR_SLAVE_ADDR4_REG (0x004C)

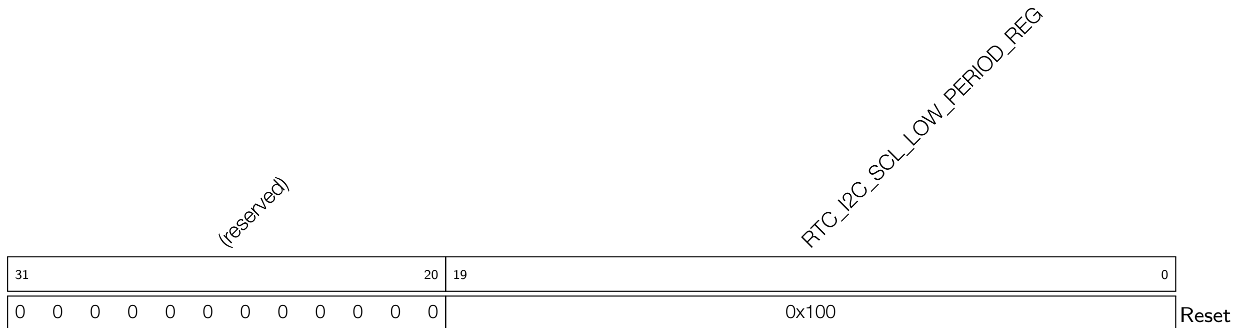


SENS_I2C_SLAVE_ADDR7 RTC I2C slave address 7. (R/W)

SENS_I2C_SLAVE_ADDR6 RTC I2C slave address 6. (R/W)

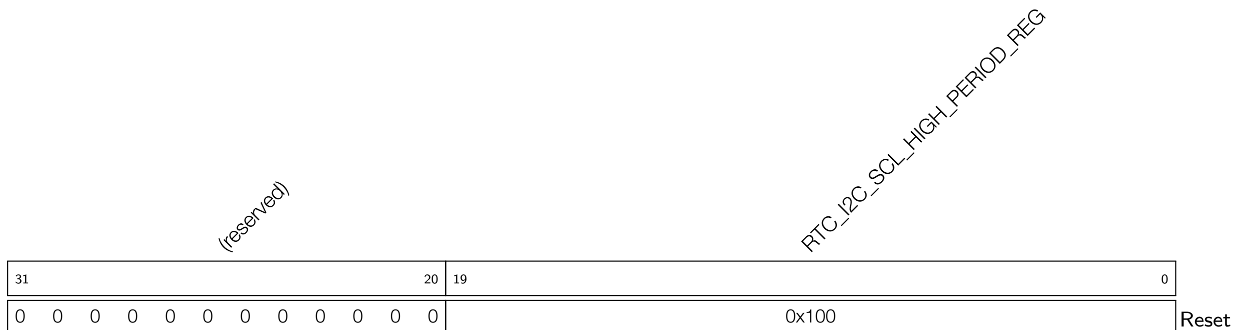
1.10.4 RTC I2C (I2C) Registers

Register 1.14: RTC_I2C_SCL_LOW_REG (0x0000)



RTC_I2C_SCL_LOW_PERIOD_REG This register is used to configure how many clock cycles SCL remains low. (R/W)

Register 1.15: RTC_I2C_SCL_HIGH_REG (0x0014)

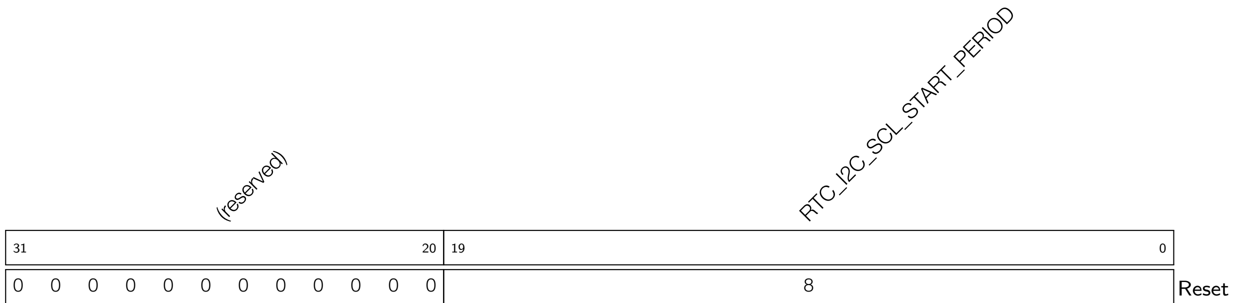


RTC_I2C_SCL_HIGH_PERIOD_REG This register is used to configure how many cycles SCL remains high. (R/W)

Register 1.16: RTC_I2C_SDA_DUTY_REG (0x0018)

RTC_I2C_SDA_DUTY_NUM The number of clock cycles between the SDA switch and the falling edge of SCL. (R/W)

Register 1.17: RTC_I2C_SCL_START_PERIOD_REG (0x001C)



RTC_I2C_SCL_START_PERIOD Number of clock cycles to wait after generating a start condition. (R/W)

Register 1.18: RTC_I2C_SCL_STOP_PERIOD_REG (0x0020)

RTC_I2C_SCL_STOP_PERIOD Number of clock cycles to wait before generating a stop condition. (R/W)

Register 1.19: RTC_I2C_CTRL_REG (0x0004)

31	30	29	28 ... 6	5	4	3	2	1	0	
(reserved)	RTC_I2C_RESET	RTC_I2C_CTRL_CLK_GATE_EN	(reserved)	RTC_I2C_RX_LSB_FIRST	RTC_I2C_TX_LSB_FIRST	RTC_I2C_TRANS_START	RTC_I2C_MS_MODE	RTC_I2C_SCL_FORCE_OUT	RTC_I2C_SDA_FORCE_OUT	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

RTC_I2C_SDA_FORCE_OUT SDA output mode. 0: open drain; 1: push pull. (R/W)

RTC_I2C_SCL_FORCE_OUT SCL output mode. 0: open drain; 1: push pull. (R/W)

RTC_I2C_MS_MODE Set this bit to configure RTC I2C as a master. (R/W)

RTC_I2C_TRANS_START Set this bit to 1, RTC I2C starts sending data. (R/W)

RTC_I2C_TX_LSB_FIRST This bit is used to control the sending mode. 0: send data from the most significant bit; 1: send data from the least significant bit. (R/W)

RTC_I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. 0: receive data from the most significant bit; 1: receive data from the least significant bit. (R/W)

RTC_I2C_CTRL_CLK_GATE_EN RTC I2C controller clock gate. (R/W)

RTC_I2C_RESET RTC I2C software reset. (R/W)

Register 1.20: RTC_I2C_STATUS_REG (0x0008)

(reserved)	RTC_I2C_OP_CNT	RTC_I2C_BYTE_TRANS	RTC_I2C_SLAVE_ADDRESSED	RTC_I2C_BUS_BUSY	RTC_I2C_ARB_LOST	RTC_I2C_SLAVE_RW	RTC_I2C_ACK_REC	
31 ... 8	7 ... 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	Reset

RTC_I2C_ACK_REC The received ACK value. 0: ACK; 1: NACK. (RO)

RTC_I2C_SLAVE_RW 0: master writes to slave; 1: master reads from slave. (RO)

RTC_I2C_ARB_LOST When the RTC I2C loses control of SCL line, the register changes to 1. (RO)

RTC_I2C_BUS_BUSY 0: RTC I2C bus is in idle state; 1: RTC I2C bus is busy transferring data. (RO)

RTC_I2C_SLAVE_ADDRESSED When the address sent by the master matches the address of the slave, then this bit will be set. (RO)

RTC_I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

RTC_I2C_OP_CNT Indicate which operation is working. (RO)

Register 1.21: RTC_I2C_TO_REG (0x000C)

(reserved)	RTC_I2C_TIME_OUT_REG	
31 ... 20	19 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0	0x10000	Reset

RTC_I2C_TIME_OUT_REG Timeout threshold. (R/W)

Register 1.22: RTC_I2C_SLAVE_ADDR_REG (0x0010)

31	30 … 0	
RTC_I2C_ADDR_10BIT_EN	(reserved)	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_I2C_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode. (R/W)

Register 1.23: RTC_I2C_INT_CLR_REG (0x0024)

31 … 9	8	7	6	5	4	3	2	1	0	
(reserved)	RTC_I2C_DETECT_START_INT_CLR	RTC_I2C_TX_DATA_INT_CLR	RTC_I2C_RX_DATA_INT_CLR	RTC_I2C_ACK_ERR_INT_CLR	RTC_I2C_TIME_OUT_INT_CLR	RTC_I2C_TRANS_COMPLETE_INT_CLR	RTC_I2C_MASTER_TRAN_COMP_INT_CLR	RTC_I2C_ARBITRATION_LOST_INT_CLR	RTC_I2C_SLAVE_TRAN_COMP_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

RTC_I2C_SLAVE_TRAN_COMP_INT_CLR RTC_I2C_SLAVE_TRAN_COMP_INT interrupt clear bit. (WO)

RTC_I2C_ARBITRATION_LOST_INT_CLR RTC_I2C_ARBITRATION_LOST_INT interrupt clear bit. (WO)

RTC_I2C_MASTER_TRAN_COMP_INT_CLR RTC_I2C_MASTER_TRAN_COMP_INT interrupt clear bit. (WO)

RTC_I2C_TRANS_COMPLETE_INT_CLR RTC_I2C_TRANS_COMPLETE_INT interrupt clear bit. (WO)

RTC_I2C_TIME_OUT_INT_CLR RTC_I2C_TIME_OUT_INT interrupt clear bit. (WO)

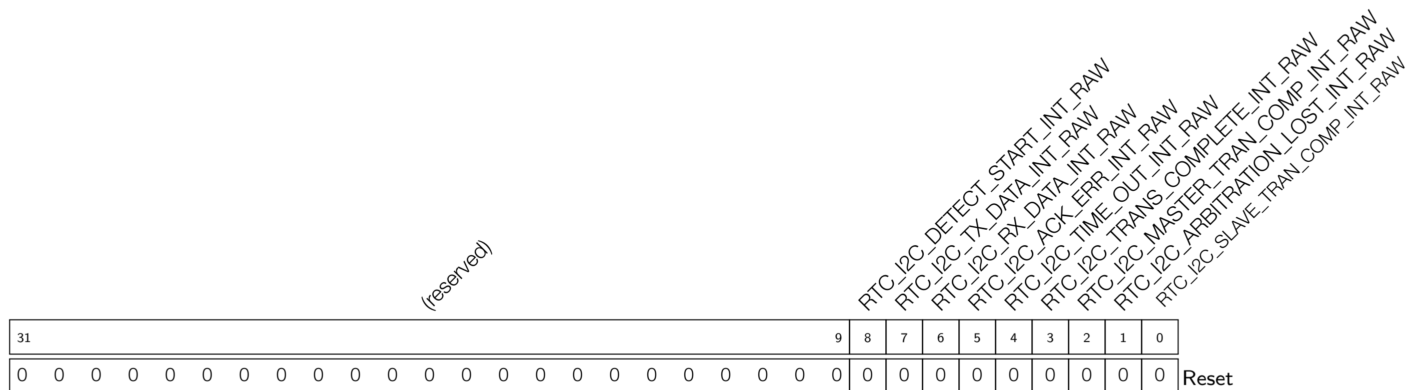
RTC_I2C_ACK_ERR_INT_CLR RTC_I2C_ACK_ERR_INT interrupt clear bit. (WO)

RTC_I2C_RX_DATA_INT_CLR RTC_I2C_RX_DATA_INT interrupt clear bit. (WO)

RTC_I2C_TX_DATA_INT_CLR RTC_I2C_TX_DATA_INT interrupt clear bit. (WO)

RTC_I2C_DETECT_START_INT_CLR RTC_I2C_DETECT_START_INT interrupt clear bit. (WO)

Register 1.24: RTC_I2C_INT_RAW_REG (0x0028)



RTC_I2C_SLAVE_TRAN_COMP_INT_RAW RTC_I2C_SLAVE_TRAN_COMP_INT interrupt raw bit. (RO)

RTC_I2C_ARBITRATION_LOST_INT_RAW RTC_I2C_ARBITRATION_LOST_INT interrupt raw bit. (RO)

RTC_I2C_MASTER_TRAN_COMP_INT_RAW RTC_I2C_MASTER_TRAN_COMP_INT interrupt raw bit. (RO)

RTC_I2C_TRANS_COMPLETE_INT_RAW RTC_I2C_TRANS_COMPLETE_INT interrupt raw bit. (RO)

RTC_I2C_TIME_OUT_INT_RAW RTC_I2C_TIME_OUT_INT interrupt raw bit. (RO)

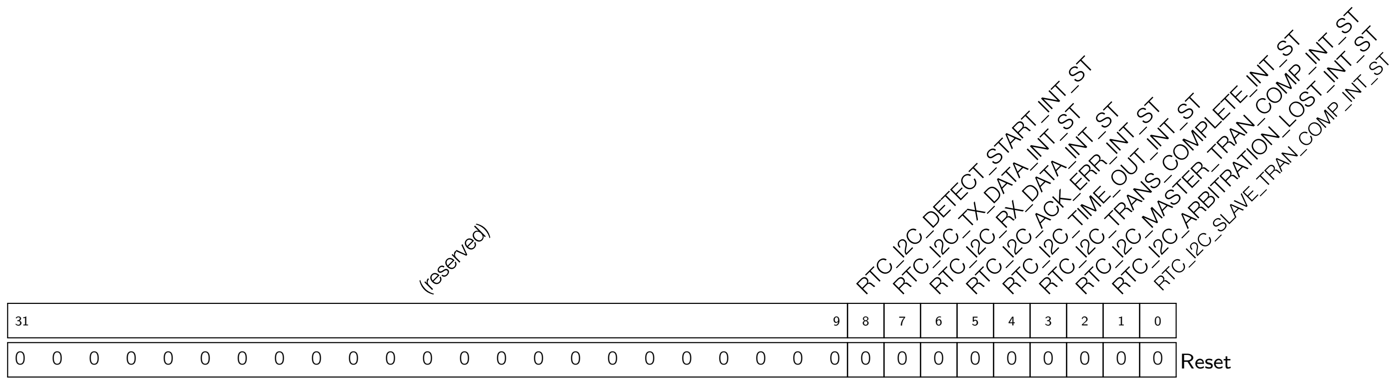
RTC_I2C_ACK_ERR_INT_RAW RTC_I2C_ACK_ERR_INT interrupt raw bit. (RO)

RTC_I2C_RX_DATA_INT_RAW RTC_I2C_RX_DATA_INT interrupt raw bit. (RO)

RTC_I2C_TX_DATA_INT_RAW RTC_I2C_TX_DATA_INT interrupt raw bit. (RO)

RTC_I2C_DETECT_START_INT_RAW RTC_I2C_DETECT_START_INT interrupt raw bit. (RO)

Register 1.25: RTC_I2C_INT_ST_REG (0x002C)



RTC_I2C_SLAVE_TRAN_COMP_INT_ST RTC_I2C_SLAVE_TRAN_COMP_INT interrupt status bit. (RO)

RTC_I2C_ARBITRATION_LOST_INT_ST RTC_I2C_ARBITRATION_LOST_INT interrupt status bit. (RO)

RTC_I2C_MASTER_TRAN_COMP_INT_ST RTC_I2C_MASTER_TRAN_COMP_INT interrupt status bit. (RO)

RTC_I2C_TRANS_COMPLETE_INT_ST RTC_I2C_TRANS_COMPLETE_INT interrupt status bit. (RO)

RTC_I2C_TIME_OUT_INT_ST RTC_I2C_TIME_OUT_INT interrupt status bit. (RO)

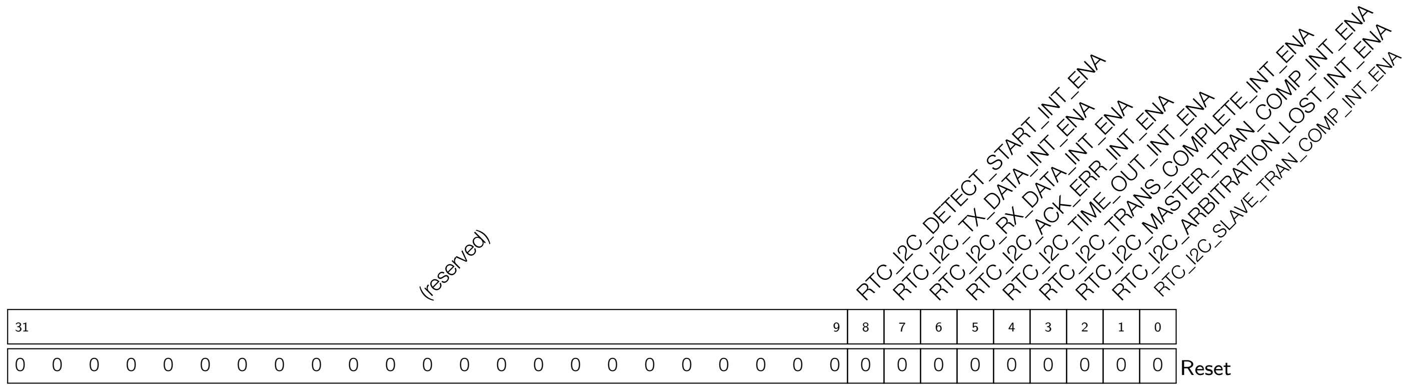
RTC_I2C_ACK_ERR_INT_ST RTC_I2C_ACK_ERR_INT interrupt status bit. (RO)

RTC_I2C_RX_DATA_INT_ST RTC_I2C_RX_DATA_INT interrupt status bit. (RO)

RTC_I2C_TX_DATA_INT_ST RTC_I2C_TX_DATA_INT interrupt status bit. (RO)

RTC_I2C_DETECT_START_INT_ST RTC_I2C_DETECT_START_INT interrupt status bit. (RO)

Register 1.26: RTC_I2C_INT_ENA_REG (0x0030)



RTC_I2C_SLAVE_TRAN_COMP_INT_ENA RTC_I2C_SLAVE_TRAN_COMP_INT interrupt enable bit. (R/W)

RTC_I2C_ARBITRATION_LOST_INT_ENA RTC_I2C_ARBITRATION_LOST_INT interrupt enable bit. (R/W)

RTC_I2C_MASTER_TRAN_COMP_INT_ENA RTC_I2C_MASTER_TRAN_COMP_INT interrupt enable bit. (R/W)

RTC_I2C_TRANS_COMPLETE_INT_ENA RTC_I2C_TRANS_COMPLETE_INT interrupt enable bit. (R/W)

RTC_I2C_TIME_OUT_INT_ENA RTC_I2C_TIME_OUT_INT interrupt enable bit. (R/W)

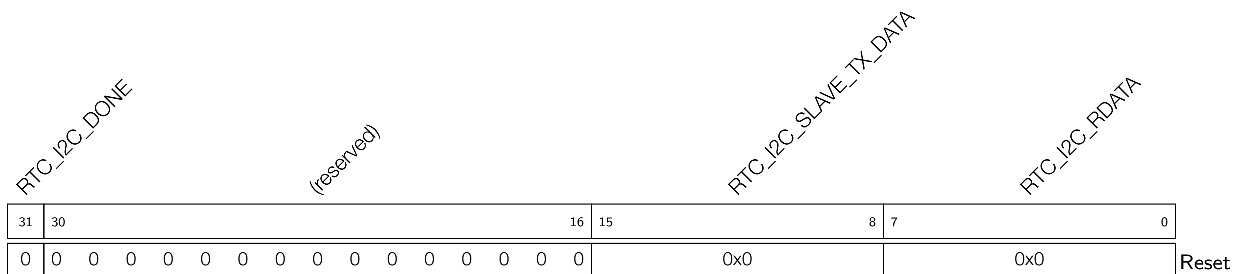
RTC_I2C_ACK_ERR_INT_ENA RTC_I2C_ACK_ERR_INT interrupt enable bit. (R/W)

RTC_I2C_RX_DATA_INT_ENA RTC_I2C_RX_DATA_INT interrupt enable bit. (R/W)

RTC_I2C_TX_DATA_INT_ENA RTC_I2C_TX_DATA_INT interrupt enable bit. (R/W)

RTC_I2C_DETECT_START_INT_ENA RTC_I2C_DETECT_START_INT interrupt enable bit. (R/W)

Register 1.27: RTC_I2C_DATA_REG (0x0034)

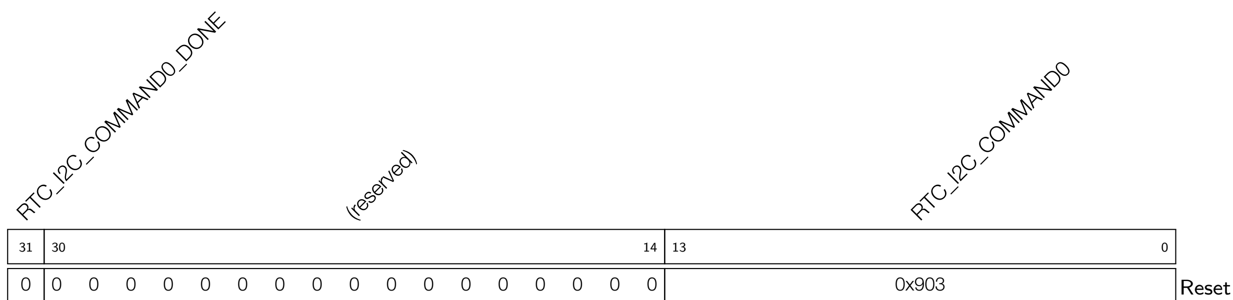


RTC_I2C_RDATA Data received. (RO)

RTC_I2C_SLAVE_TX_DATA The data sent by slave. (R/W)

RTC_I2C_DONE RTC I2C transmission is done. (RO)

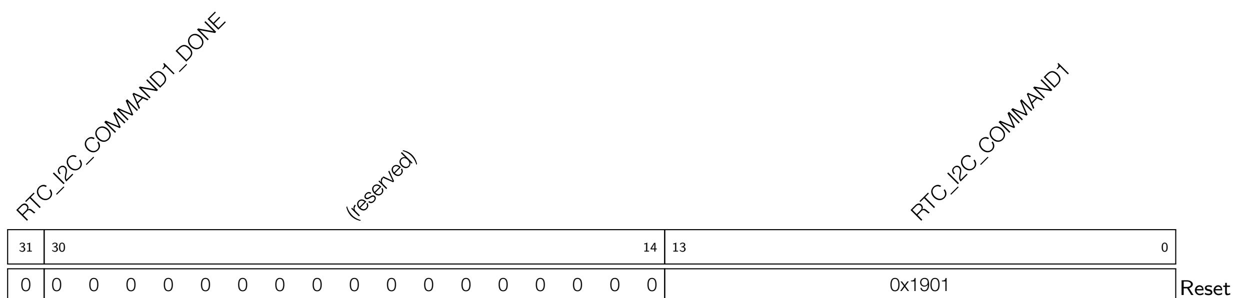
Register 1.28: RTC_I2C_CMD0_REG (0x0038)



RTC_I2C_COMMAND0 Content of command 0. For more information, please refer to the register I2C_COMD0_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND0_DONE When command 0 is done, this bit changes to 1. (RO)

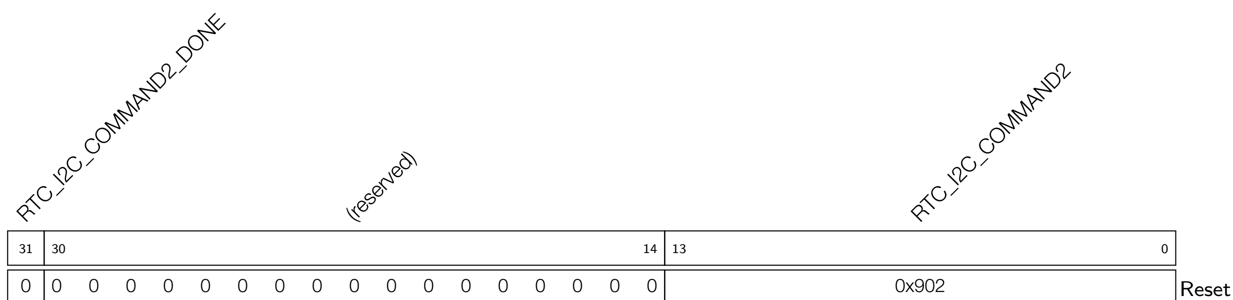
Register 1.29: RTC_I2C_CMD1_REG (0x003C)



RTC_I2C_COMMAND1 Content of command 1. For more information, please refer to the register I2C_COMD1_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND1_DONE When command 1 is done, this bit changes to 1. (RO)

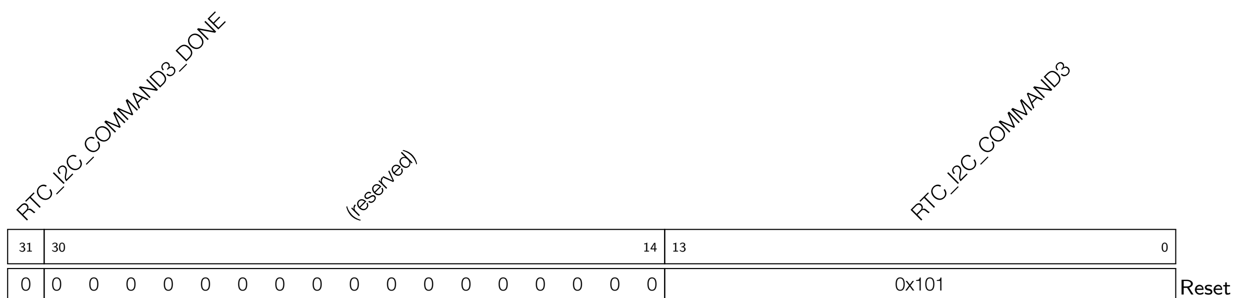
Register 1.30: RTC_I2C_CMD2_REG (0x0040)



RTC_I2C_COMMAND2 Content of command 2. For more information, please refer to the register I2C_COMD2_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND2_DONE When command 2 is done, this bit changes to 1. (RO)

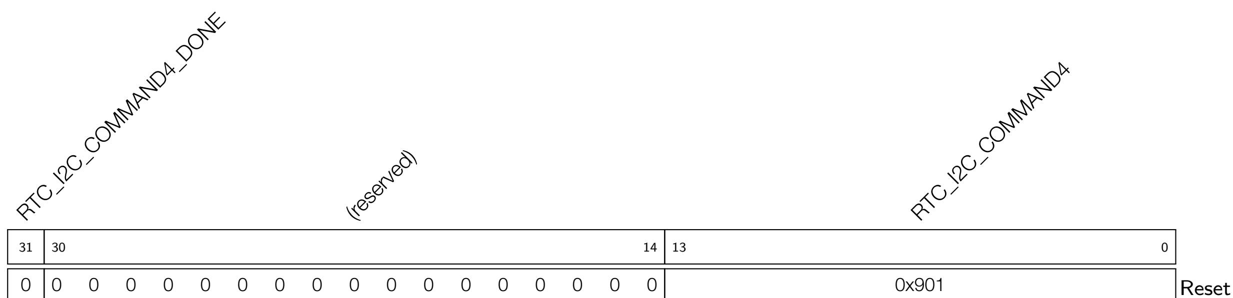
Register 1.31: RTC_I2C_CMD3_REG (0x0044)



RTC_I2C_COMMAND3 Content of command 3. For more information, please refer to the register I2C_COMD3_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND3_DONE When command 3 is done, this bit changes to 1. (RO)

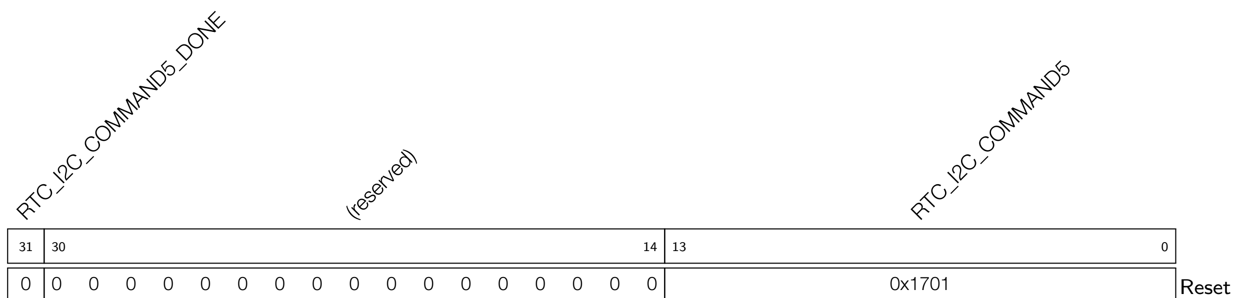
Register 1.32: RTC_I2C_CMD4_REG (0x0048)



RTC_I2C_COMMAND4 Content of command 4. For more information, please refer to the register I2C_COMD4_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND4_DONE When command 4 is done, this bit changes to 1. (RO)

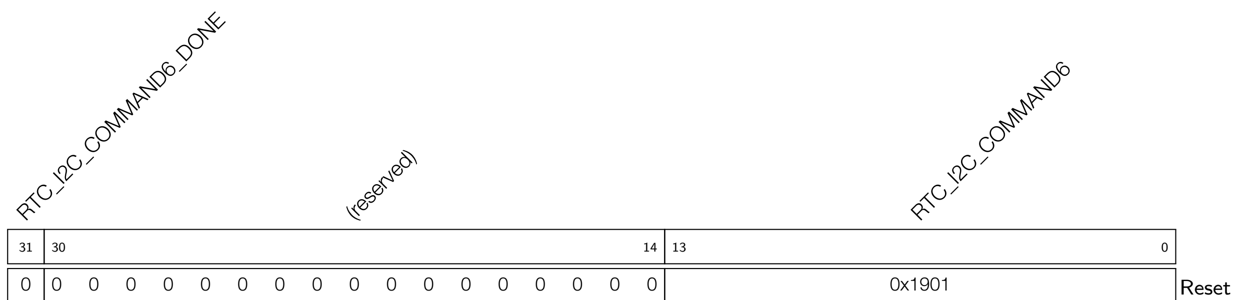
Register 1.33: RTC_I2C_CMD5_REG (0x004C)



RTC_I2C_COMMAND5 Content of command 5. For more information, please refer to the register I2C_COMD5_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND5_DONE When command 5 is done, this bit changes to 1. (RO)

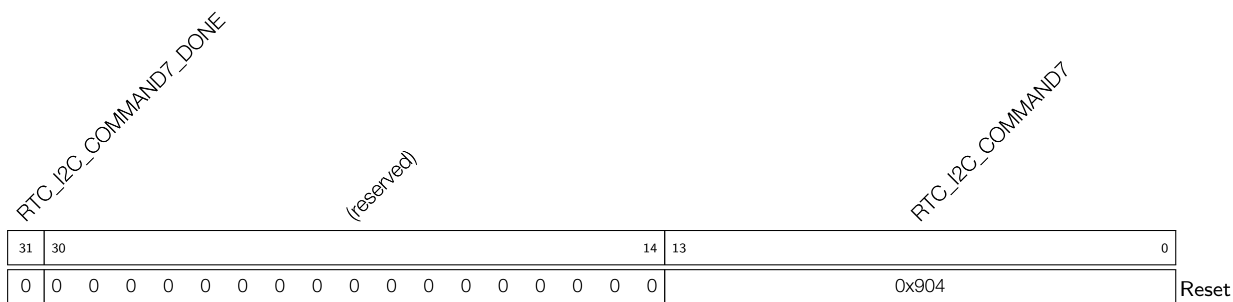
Register 1.34: RTC_I2C_CMD6_REG (0x0050)



RTC_I2C_COMMAND6 Content of command 6. For more information, please refer to the register I2C_COMD6_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND6_DONE When command 6 is done, this bit changes to 1. (RO)

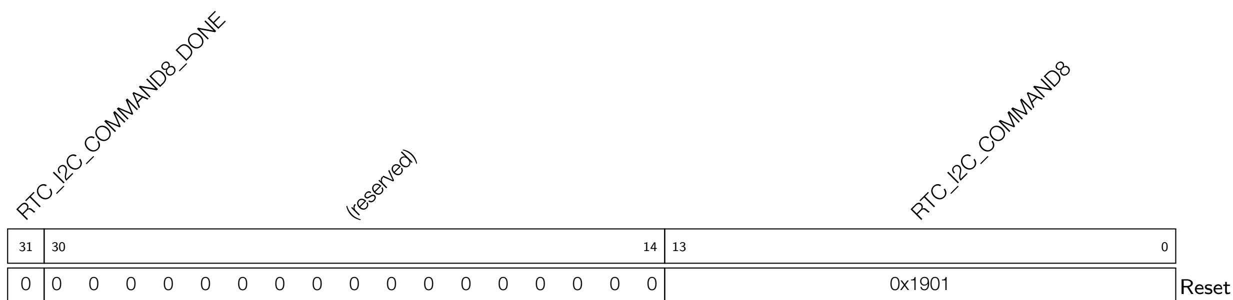
Register 1.35: RTC_I2C_CMD7_REG (0x0054)



RTC_I2C_COMMAND7 Content of command 7. For more information, please refer to the register I2C_COMD7_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND7_DONE When command 7 is done, this bit changes to 1. (RO)

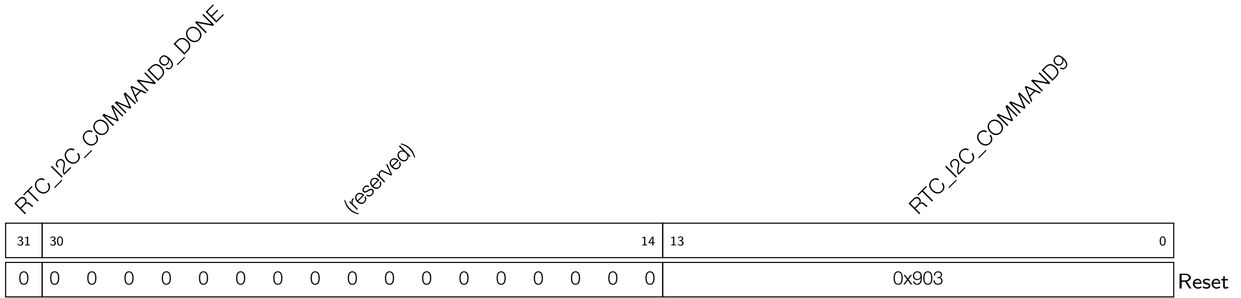
Register 1.36: RTC_I2C_CMD8_REG (0x0058)



RTC_I2C_COMMAND8 Content of command 8. For more information, please refer to the register I2C_COMD8_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND8_DONE When command 8 is done, this bit changes to 1. (RO)

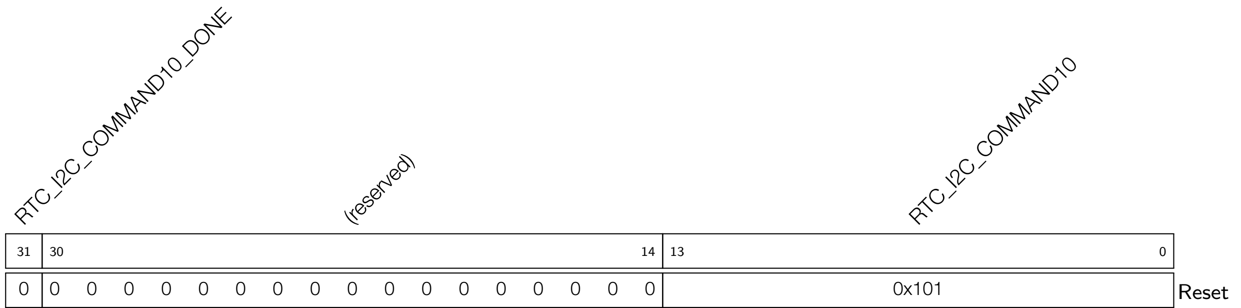
Register 1.37: RTC_I2C_CMD9_REG (0x005C)



RTC_I2C_COMMAND9 Content of command 9. For more information, please refer to the register I2C_COMD9_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND9_DONE When command 9 is done, this bit changes to 1. (RO)

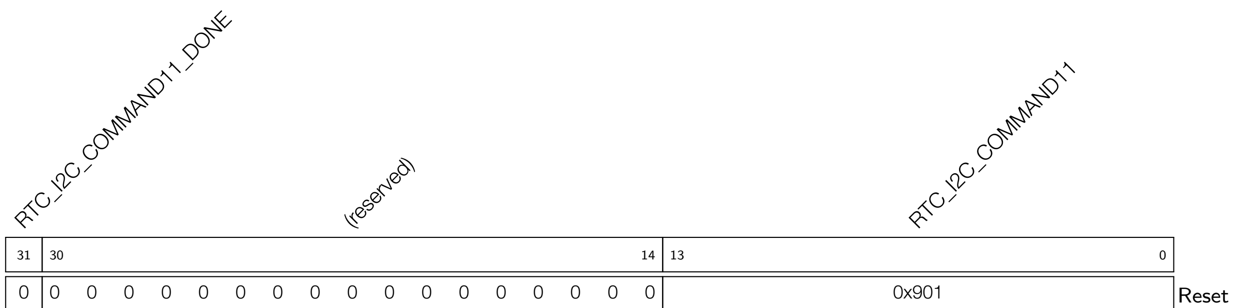
Register 1.38: RTC_I2C_CMD10_REG (0x0060)



RTC_I2C_COMMAND10 Content of command 10. For more information, please refer to the register I2C_COMD10_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND10_DONE When command 10 is done, this bit changes to 1. (RO)

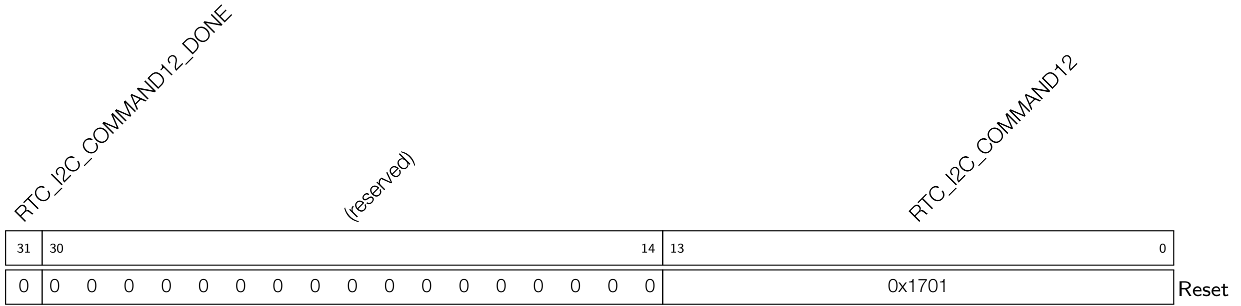
Register 1.39: RTC_I2C_CMD11_REG (0x0064)



RTC_I2C_COMMAND11 Content of command 11. For more information, please refer to the register I2C_COMD11_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND11_DONE When command 11 is done, this bit changes to 1. (RO)

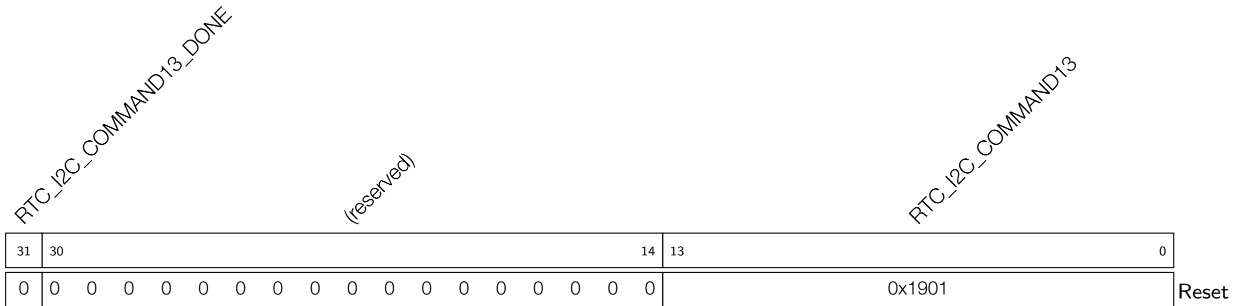
Register 1.40: RTC_I2C_CMD12_REG (0x0068)



RTC_I2C_COMMAND12 Content of command 12. For more information, please refer to the register I2C_COMD12_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND12_DONE When command 12 is done, this bit changes to 1. (RO)

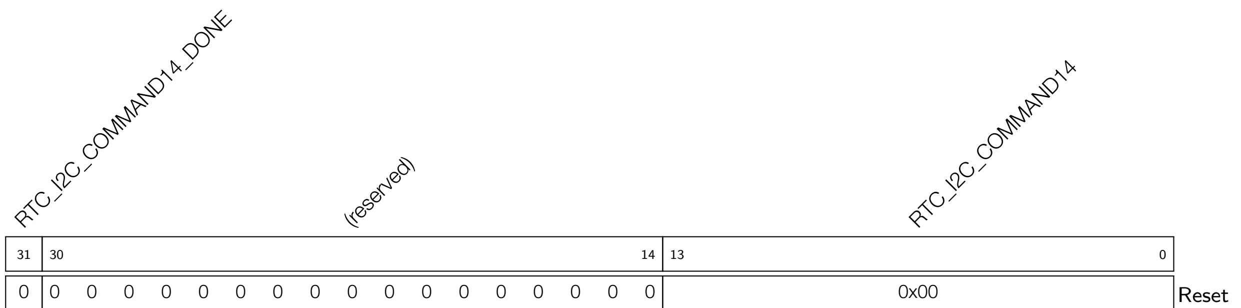
Register 1.41: RTC_I2C_CMD13_REG (0x006C)



RTC_I2C_COMMAND13 Content of command 13. For more information, please refer to the register I2C_COMD13_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND13_DONE When command 13 is done, this bit changes to 1. (RO)

Register 1.42: RTC_I2C_CMD14_REG (0x0070)



RTC_I2C_COMMAND14 Content of command 14. For more information, please refer to the register I2C_COMD14_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND14_DONE When command 14 is done, this bit changes to 1. (RO)

Register 1.43: RTC_I2C_CMD15_REG (0x0074)

RTC_I2C_COMMAND15_DONE	(reserved)	RTC_I2C_COMMAND15	
31	30 ... 14	13 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

RTC_I2C_COMMAND15 Content of command 15. For more information, please refer to the register I2C_COMD15_REG in Chapter I2C Controller. (R/W)

RTC_I2C_COMMAND15_DONE When command 15 is done, this bit changes to 1. (RO)

Register 1.44: RTC_I2C_DATE_REG (0x00FC)

(reserved)	RTC_I2C_DATE	
31 ... 28	27 ... 0	
0 0 0 0	0x1905310	Reset

RTC_I2C_DATE Version control register (R/W)

2. DMA Controller (DMA)

2.1 Overview

Direct Memory Access (DMA) is a feature that allows peripheral-to-memory and memory-to-memory data transfer at a high speed. The CPU is not involved in the DMA transfer, and therefore it becomes more efficient.

ESP32-S2 has three types of DMA, namely Internal DMA, EDMA and Copy DMA. Internal DMA can only access internal RAM and is used for data transfer between internal RAM and peripherals. EDMA can access both internal RAM and external RAM and is used for data transfer between internal RAM, external RAM and peripherals. Copy DMA can only access internal RAM and is used for data transfer from one location in internal RAM to another.

Eight peripherals on ESP32-S2 have DMA features. As shown in Figure 2-1, UART0 and UART1 share one Internal DMA; SPI3 and ADC Controller share one Internal DMA; AES Accelerator and SHA Accelerator share one EDMA (i.e. Crypto DMA); SPI2 and I2S0 have their individual EDMA. Besides, the CPU Peripheral module on ESP32-S2 also has one Copy DMA.

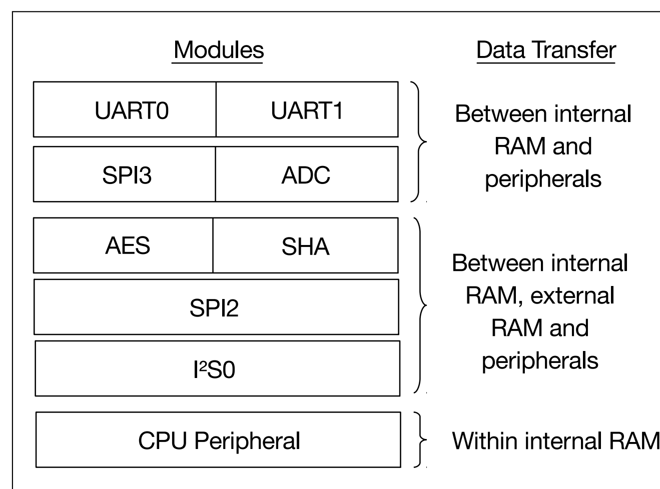


Figure 2-1. Modules with DMA and Supported Data Transfers

2.2 Features

The DMA controller has the following features:

- AHB bus architecture
- Half-duplex and full-duplex mode
- Programmable length of data to be transferred in bytes
- INCR burst transfer when accessing internal RAM
- Access to an address space of 320 KB at most in internal RAM
- Access to an address space of 10.5 MB at most in external RAM
- High-speed data transfer using DMA

2.3 Functional Description

In ESP32-S2, all modules that need high-speed data transfer support DMA. The DMA controller and CPU data bus have access to the same address space in internal RAM and external RAM. DMA controllers for different modules vary in functions according to needs, but their architecture is identical.

2.3.1 DMA Engine Architecture

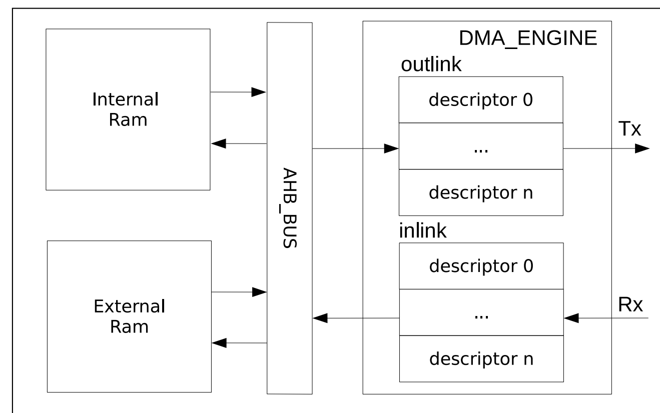


Figure 2-2. DMA Engine Architecture

A DMA engine reads/writes data to/from external RAM or internal RAM via the AHB_BUS. Figure 2-2 shows the basic architecture of a DMA engine. For how to access RAM, please see Chapter 3 *System and Memory*. Software can use the DMA engine through linked lists. The DMA_ENGINE transmits data in corresponding RAM according to the outlink (i.e. a linked list of transmit descriptors), and stores received data into specific address space in RAM according to the inlink (i.e. a linked list of receive descriptors).

2.3.2 Linked List

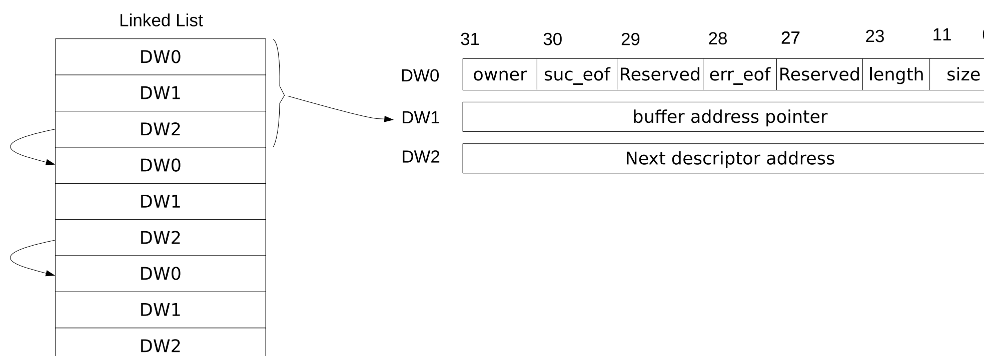


Figure 2-3. Structure of a Linked List

Figure 2-3 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of three words. Linked lists should be in internal RAM for the DMA engine to be able to use them. The meaning of each field is as follows:

- Owner (DW0) [31]: Specifies who is allowed to access the buffer that this descriptor points to.
  1'b0: CPU can access the buffer;
  1'b1: The DMA controller can access the buffer.
  When the DMA controller stops using the buffer, this bit is cleared by hardware. You can set *PERI*_IN_LOOP_TEST bit to disable automatic clearing by hardware. When software loads a linked list, this

bit should be set to 1.
Note: *PERI* refers to modules that support DMA transfers, e.g. I2S, SPI, UHCI, etc.

- suc_eof (DW0) [30]: Specifies whether this descriptor is the last descriptor in the list.
  1'b0: This descriptor is not the last one;
  1'b1: This descriptor is the last one.
  Software clears suc_eof bit in receive descriptors. When a packet has been received, this bit in the last receive descriptor is set by hardware, and this bit in the last transmit descriptor is set by software.
- Reserved (DW0) [29]: Reserved.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  This bit is used only when UART DMA receives data. When an error is detected in the received packet, this bit in the receive descriptor is set to 1 by hardware.
- Reserved (DW0) [27:24]: Reserved.
- Length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many bytes have been stored into the buffer.
- Size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
  When the DMA controller accesses external RAM, this field must be a multiple of 16/32/64 bytes. Please see more details in Section 2.3.8 *Accessing External RAM*.
- Buffer address pointer (DW1): Pointer to the buffer.
  When the DMA controller accesses external RAM, the destination address must be aligned with *PERI*_EXT_MEM_BK_SIZE field. Please see more details in Section 2.3.8 *Accessing External RAM*.
- Next descriptor address (DW2): Pointer to the next descriptor. If the current descriptor is the last one (suc_eof = 1), this value is 0. This field can only point to internal RAM.

If the length of data received is smaller than the size of the buffer, the DMA controller will not use available space of the buffer in the next transaction.

2.3.3 Enabling DMA

Software uses the DMA controller through linked lists. When the DMA controller receives data, software loads an inlink, configures *PERI*_INLINK_ADDR field with address of the first receive descriptor, and sets *PERI*_INLINK_START bit to enable DMA. When the DMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures *PERI*_OUTLINK_ADDR field with address of the first transmit descriptor, and sets *PERI*_OUTLINK_START bit to enable DMA. *PERI*_INLINK_START bit and *PERI*_OUTLINK_START bit are cleared automatically by hardware.

The DMA controller can be restarted. If you are not sure whether the loaded linked list has been used up or not and want to load a new linked list, you can use this Restart function without affecting the loaded linked list. When using the Restart function, software needs to rewrite address of the first descriptor in the new list to DW2 of the last descriptor in the loaded list, loads the new list as shown in Figure 2-4, and set *PERI*_INLINK_RESTART bit or *PERI*_OUTLINK_RESTART bit (these two bits are cleared automatically by hardware). By doing so, hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

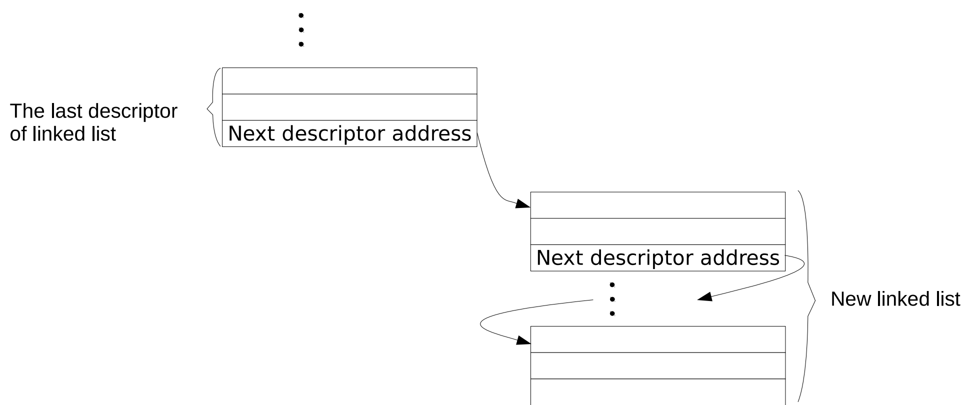


Figure 2-4. Relationship among Linked Lists

2.3.4 Linked List reading process

Once configured and enabled by software, the DMA controller starts to read the linked list from internal RAM. *PERI*_IN_DSCR_ERR_INT_ENA bit or *PERI*_OUT_DSCR_ERR_INT_ENA bit can be set to enable descriptor error interrupt. If the buffer address pointer (DW1) does not point to 0x3FFB0000 ~ 0x3FFFFFFF when the DMA controller accesses internal RAM, or does not point to 0x3F500000 ~ 0x3FF7FFFF when the DMA controller accesses external RAM, a descriptor error interrupt is generated.

Note: The third word (DW2) in a descriptor can only point to internal RAM; it points to the next descriptor to use and descriptors must be in internal memory.

2.3.5 EOF

The DMA controller uses EOF (end of file) flags to indicate the completion of data transfer.

Before the DMA controller transmits data, *PERI*_OUT_TOTAL_EOF_INT_ENA bit should be set. If data in the buffer pointed by the last descriptor has been transmitted, a *PERI*_OUT_TOTAL_EOF_INT interrupt is generated.

Before the DMA controller receives data, *PERI*_IN_SUC_EOF_INT_ENA bit should be set. If data has been received successfully, a *PERI*_IN_SUC_EOF_INT interrupt is generated. In addition to *PERI*_IN_SUC_EOF_INT interrupt, UART DMA also supports UHCI_IN_ERR_EOF_INT. This interrupt is enabled by setting UHCI_IN_ERR_EOF_INT_ENA bit, and it indicates that a data packet has been received with errors.

When a *PERI*_OUT_TOTAL_EOF_INT or a *PERI*_IN_SUC_EOF_INT interrupt is detected, software can record the value of field *PERI*_OUTLINK_DSCR_ADDR or *PERI*_INLINK_DSCR_ADDR, i.e. address of the last descriptor (right shifted 2 bits). Therefore, software can tell which descriptors have been used and reclaim them.

Note: In this chapter, EOF of transmit descriptors refers to suc_eof, EOF of receive descriptors refers to both suc_eof and err_eof.

2.3.6 Internal DMA

Internal DMA is used by UART0, UART1, SPI3, and ADC Controller. It can only access 0x3FFB0000 ~ 0x3FFFFFFF in internal RAM. For Internal DMA, size, length, and buffer address pointer in linked list descriptors are not necessarily word-aligned. In other words, Internal DMA can read data of specified length from any starting address in the accessible address range, or write data of the specified length to any contiguous addresses in the accessible address range.

2.3.7 EDMA

EDMA is used by I2S0, SPI2, AES Accelerator, and SHA Accelerator. It can access both internal RAM and external RAM.

2.3.7.1 Accessing Internal RAM

Like Internal DMA, EDMA can access 0x3FFB0000 ~ 0x3FFFFFFF in internal RAM. Size, length, and buffer address pointer in linked list descriptors are also not necessarily word-aligned. Please note that if EDMA receives more than one data frame with null character (null character indicates empty data, which will not be written into memory by EDMA; such a character is generated together with EOF), as shown in Figure 2-5 there must be at least three bytes between EOF in the first data frame and EOF in the second data frame. In this case, EOF could be either suc_eof or err_eof.

EDMA can send data in burst mode to improve data transfer efficiency. To enable burst mode, please set *PERI*_OUT_DATA_BURST_EN bit.

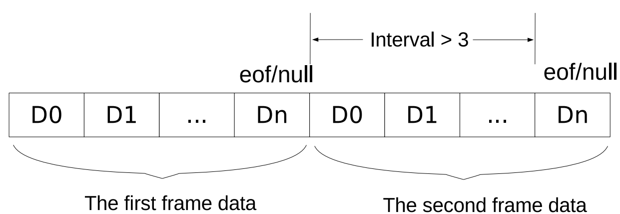


Figure 2-5. EDMA Receiving Data Frames in Internal

2.3.8 Accessing External RAM

EDMA can access 0x3F500000 ~ 0x3FF7FFFF in external RAM. The block size of EDMA can be 16 bytes, 32 bytes or 64 bytes. That is, the amount of data to transfer at a time can be 16 bytes, 32 bytes or 64 bytes. To configure the block size, please set *PERI*_EXT_MEM_BK_SIZE bit. Note that all EDMA should have the same block size.

When EDMA accesses external RAM, fields in linked list descriptors should be aligned with block size. Specifically, size and buffer address pointer in receive descriptors should be 16-byte, 32-byte or 64-byte aligned. For data frame whose length is not a multiple of 16 bytes, 32 bytes, or 64 bytes, EDMA adds padding bytes to the end. After *PERI*_DMA_IN_SUC_EOF_INT is detected, software can check the amount of valid data by reading length field in receive descriptors. Size, length and buffer address pointer in transmit descriptors are not necessarily aligned with block size. Table 16 illustrates the value of *PERI*_EXT_MEM_BK_SIZE bit when fields in linked list descriptors are 16-byte, 32-byte and 64-byte aligned respectively.

Table 16: Relationship Between Configuration Register, Block Size and Alignment

*PERI*_EXT_MEM_BK_SIZE	Block Size	Alignment
0	16 bytes	16-byte aligned
1	32 bytes	32-byte aligned
2	64 bytes	64-byte aligned

2.4 Copy DMA Controller

Copy DMA is used for data transfer from one location in internal RAM to another. Figure 2-6 shows the architecture of a Copy DMA engine. Unlike Internal DMA and EDMA, Copy DMA first reads data to be transferred from internal RAM, stores the data into the DMA FIFO via an outlink, and then writes the data to the target internal RAM via an inlink.

Copy DMA should be configured by software as follows:

1. Set CP_DMA_IN_RST, CP_DMA_OUT_RST, CP_DMA_FIFO_RST and CP_DMA_CMDFIFO_RST bit first to 1 and then to 0, to reset Copy DMA state machine and FIFO pointer;
2. Load an outlink, and configure CP_DMA_OUTLINK_ADDR with address of the first transmit descriptor;
3. Load an inlink, and configure CP_DMA_INLINK_ADDR with address of the first receive descriptor;
4. Set CP_DMA_OUTLINK_START to enable DMA transmission;
5. Set CP_DMA_INLINK_START to enable DMA reception.

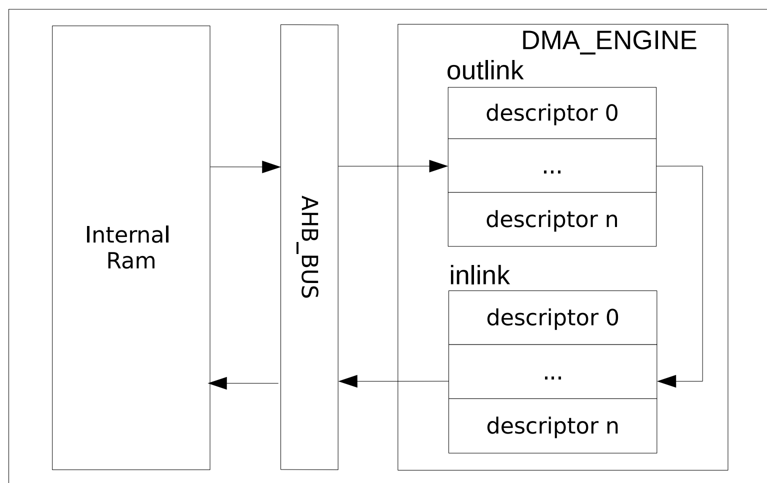


Figure 2-6. Copy DMA Engine Architecture

2.5 UART DMA (UDMA) Controller

ESP32-S2 has two UART controllers. They share one UDMA controller. UHCI_UART_CE specifies which UART controller gets access to UDMA.

Figure 2-7 shows how data is transferred using UDMA. Before UDMA receives data, software prepares an inlink. UHCI_INLINK_ADDR points to the first receive descriptor in the inlink. After UHCI_INLINK_START is set, UHCI sends data that UART has received to the Decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of UDMA.

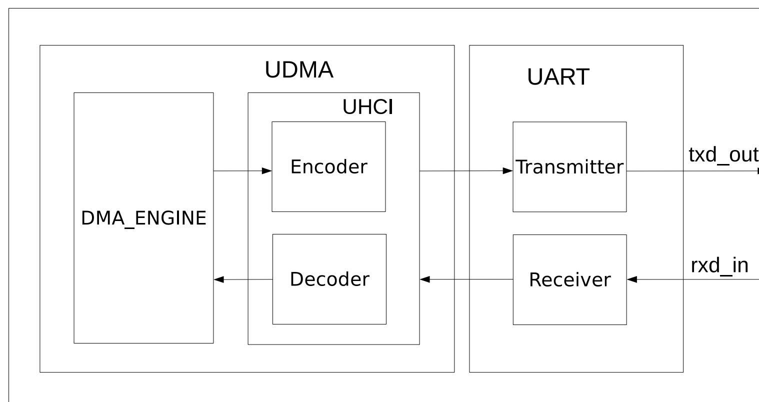


Figure 2-7. Data Transfer in UDMA Mode

Before UDMA sends data, software prepares an outlink and data to be sent. UHCI_OUTLINK_ADDR points to the first transmit descriptor in the outlink. After UHCI_OUTLINK_START is set, UDMA reads data from the RAM pointed by outlink. The data is then encoded by the Encoder, and sent sequentially by the UART transmitter.

Data packets of UDMA have separators at the beginning and the end, with data bits in the middle. The encoder inserts separators in front of and after data bits, and replaces data bits identical to separators with special characters. The decoder removes separators in front of and after data bits, and replaces special characters with separators. There can be more than one continuous separator at the beginning and the end of a data packet. The separator is configured by UHCI_SEPER_CHAR, 0xC0 by default. The special character is configured by UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). When all data has been sent, a UHCI_OUT_TOTAL_EOF_INT interrupt is generated. When all data has been received, a UHCI_IN_SUC_EOF_INT is generated.

2.6 SPI DMA Controller

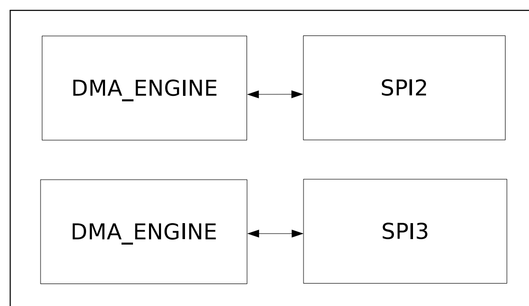


Figure 2-8. SPI DMA

As shown in Figure 2-8, SPI2 and SPI3 have separate DMA controllers.

SPI DMA receives and transmits data through descriptors at least one byte at a time. The transmission of data can be done in bursts.

SPI_OUTLINK_START bit of SPI_DMA_OUT_LINK_REG register and SPI_INLINK_START bit of SPI_DMA_IN_LINK_REG register are used to enable the DMA engine and are cleared by hardware. When SPI_OUTLINK_START is set to 1, the DMA engine loads an outlink and prepares data to be transferred; when SPI_INLINK_START is set to 1, the DMA engine loads an inlink and prepares to receive data.

When receiving data, SPI DMA should be configured by software as follows:

1. Set SPI_IN_RST, SPI_AHBM_FIFO_RST and SPI_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an inlink, and configure SPI_INLINK_ADDR with address of the first receive descriptor;
3. Set SPI_INLINK_START to enable DMA reception.

When transmitting data, SPI DMA should be configured by software as follows:

1. Set SPI_OUT_RST, SPI_AHBM_FIFO_RST and SPI_AHBM_RST bit first to 1 and then to 0, to reset RMA state machine and FIFO pointer;
2. Load an outlink, and configure SPI_OUTLINK_ADDR with address of the first transmit descriptor;
3. Set SPI_OUTLINK_START to enable DMA transmission.

Note: When SPI DMA transfers data between internal RAM and external RAM, SPI_MEM_TRANS_EN should be set.

SPI DMA also supports data transfer in segments. For more details, please refer to Chapter 24 *SPI Controller (SPI)*.

2.7 I2S DMA Controller

ESP32-S2 I2S has an individual DMA. I2S_DSCR_EN bit of I2S_FIFO_CONF_REG register is used to enable DMA transfer of I2S. I2S DMA receives and transmits data through linked lists. The transmission of data can be done in bursts. I2S_RX_EOF_NUM[31:0] bit of I2S_RXEOF_NUM_REG register is used to configure how many words of data to be received at a time.

I2S_OUTLINK_START bit of I2S_OUT_LINK_REG and I2S_INLINK_START bit of I2S_IN_LINK_REG register are used to enable the DMA engine and are cleared by hardware. When I2S_OUTLINK_START bit is set to 1,the DMA engine loads an outlink and prepares data to be transferred; when I2S_INLINK_START is set to 1,the DMA engine loads an inlink and prepares to receive data.

When receiving data, I2S DMA should be configured by software as follows:

1. Set I2S_IN_RST, I2S_AHBM_FIFO_RST and I2S_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an inlink, and configure I2S_INLINK_ADDR with address of the first receive descriptor;
3. Set I2S_INLINK_START to enable DMA reception.

When transmitting data, I2S DMA should be configured by software as follows:

1. Set I2S_OUT_RST, I2S_AHBM_FIFO_RST and I2S_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an outlink, and configure I2S_OUTLINK_ADDR with address of the first transmit descriptor;
3. Set I2S_OUTLINK_START to enable DMA transmission.

Note: When I2S DMA transfers data between internal RAM and external RAM using I2S DMA, I2S_MEM_TRANS_EN should be set.

For how data is stored in I2S mode, please refer to description about I2S mode in Chapter 26 *I2S Controller (I2S)*; for how data is stored in LED mode, please refer to Section 26.10 *LCD Master Transmitting Mode* in Chapter 26 *I2S Controller (I2S)*.

For more information about I2S DMA interrupts, please refer to Section 26.12.2 *DMA Interrupts* in Chapter 26 *I2S Controller (I2S)*.

2.8 Crypto DMA

AES Accelerator and SHA Accelerator (cryptographic hardware accelerators) on ESP32-S2 share one dedicated EDMA, which is called as Crypto DMA. If CRYPTO_DMA_AES_SHA_SELECT is set, Crypto DMA is used by SHA Accelerator. If CRYPTO_DMA_AES_SHA_SELECT is cleared, Crypto DMA is used by AES Accelerator.

When receiving data, Crypto DMA should be configured by software as follows:

1. Set CRYPTO_DMA_IN_RST, CRYPTO_DMA_AHBM_FIFO_RST and CRYPTO_DMA_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an inlink, and configure CRYPTO_DMA_INLINK_ADDR with address of the first receive descriptor;
3. Set CRYPTO_DMA_INLINK_START to enable DMA reception.

When transmitting data, Crypto DMA should be configured by software as follows:

1. Set CRYPTO_DMA_OUT_RST, CRYPTO_DMA_AHBM_FIFO_RST and CRYPTO_DMA_AHBM_RST first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an outlink, and configure CRYPTO_DMA_OUTLINK_ADDR with address of the first transmit descriptor;
3. Set CRYPTO_DMA_OUTLINK_START to enable DMA transmission.

Note: When Crypto DMA is used for data transfer between internal RAM and external RAM, please set CRYPTO_DMA_MEM_TRANS_EN.

2.9 Copy DMA Interrupts

- CP_DMA_OUT_TOTAL_EOF_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent.
- CP_DMA_IN_DSCR_EMPTY_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received.
- CP_DMA_OUT_DSCR_ERR_INT: Triggered when an error is detected in a transmit descriptor.
- CP_DMA_IN_DSCR_ERR_INT: Triggered when an error is detected in a receive descriptor.
- CP_DMA_OUT_EOF_INT: Triggered when EOF in a transmit descriptor is 1 and data corresponding to this descriptor has been sent.
- CP_DMA_OUT_DONE_INT: Triggered when all data corresponding to a transmit descriptor has been sent.
- CP_DMA_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- CP_DMA_IN_DONE_INT: Triggered when all data corresponding to a receive descriptor has been received.

2.10 Crypto DMA Interrupts

- CRYPTO_DMA_INFIFO_FULL_WM_INT: Triggered when the amount of data in DMA RX FIFO is larger than the threshold (the value of CRYPTO_DMA_INFIFO_FULL_THRS).

- CRYPTO_DMA_OUT_TOTAL_EOF_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent.
- CRYPTO_DMA_IN_DSCR_EMPTY_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received.
- CRYPTO_DMA_OUT_DSCR_ERR_INT: Triggered when an error is detected in a receive descriptor.
- CRYPTO_DMA_IN_DSCR_ERR_INT: Triggered when an error is detected in a transmit descriptor.
- CRYPTO_DMA_OUT_EOF_INT: Triggered when EOF in a transmit descriptor is 1 and all data corresponding to this descriptor has been sent.
- CRYPTO_DMA_OUT_DONE_INT: Triggered when all data corresponding to a transmit descriptor has been sent.
- CRYPTO_DMA_IN_ERR_EOF_INT: Reserved.
- CRYPTO_DMA_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- CRYPTO_DMA_IN_DONE_INT: Triggered when all data corresponding to a receive descriptor has been sent.

2.11 Base Address

Users can access Copy DMA and Crypto DMA respectively with two base addresses, which can be seen in table 17. Details about UART DMA, SPI DMA and I2S DMA are included in individual chapters. For more information about accessing peripherals from different buses, please see Chapter 3 *System and Memory*.

Table 17: Copy DMA and Crypto DMA Base Address

Module	Bus to Access Peripheral	Base Address
CP DMA	PeriBUS1	0x3F4C3000
CRYPTO DMA	PeriBUS1	0x3F43F000
	PeriBUS2	0x6003F000

2.12 Register Summary

The addresses in the following table are relative to the Copy DMA base addresses provided in Section 2.11.

Name	Description	Address	Access
Interrupt Registers			
CP_DMA_INT_RAW_REG	Raw interrupt status	0x0000	RO
CP_DMA_INT_ST_REG	Masked interrupt status	0x0004	RO
CP_DMA_INT_ENA_REG	Interrupt enable bits	0x0008	R/W
CP_DMA_INT_CLR_REG	Interrupt clear bits	0x000C	WO
Configuration Registers			
CP_DMA_OUT_LINK_REG	Link descriptor address and control	0x0010	varies
CP_DMA_IN_LINK_REG	Link descriptor address and control	0x0014	varies
CP_DMA_CONF_REG	Copy DMA configuration register	0x003C	R/W
Status Registers			
CP_DMA_OUT_EOF_DES_ADDR_REG	Transmit descriptor address when EOF occurs	0x0018	RO

Name	Description	Address	Access
CP_DMA_IN_EOF_DES_ADDR_REG	Receive descriptor address when EOF occurs	0x001C	RO
CP_DMA_OUT_EOF_BFR_DES_ADDR_REG	Transmit descriptor address before the last transmit descriptor	0x0020	RO
CP_DMA_INLINK_DSCR_REG	Address of current receive descriptor	0x0024	RO
CP_DMA_INLINK_DSCR_BF0_REG	Address of last receive descriptor	0x0028	RO
CP_DMA_OUTLINK_DSCR_REG	Address of current transmit descriptor	0x0030	RO
CP_DMA_OUTLINK_DSCR_BF0_REG	Address of last transmit descriptor	0x0034	RO
CP_DMA_IN_ST_REG	Status register of receiving data	0x0040	RO
CP_DMA_OUT_ST_REG	Status register of transmitting data	0x0044	RO
CP_DMA_DATE_REG	Copy DMA version control register	0x00FC	R/W

The addresses in the following table are relative to the Crypto DMA base addresses provided in Section 2.11.

Name	Description	Address	Access
Configuration Registers			
CRYPTO_DMA_CONF0_REG	DMA configuration register	0x0000	R/W
CRYPTO_DMA_OUT_LINK_REG	Link descriptor address and control	0x0024	varies
CRYPTO_DMA_IN_LINK_REG	Link descriptor address and control	0x0028	varies
CRYPTO_DMA_CONF1_REG	DMA configuration register	0x002C	R/W
CRYPTO_DMA_AHB_TEST_REG	AHB test register	0x0048	R/W
CRYPTO_DMA_AES_SHA_SELECT_REG	AES/SHA select register	0x0064	R/W
CRYPTO_DMA_PD_CONF_REG	Power control register	0x0068	R/W
CRYPTO_DMA_DATE_REG	Crypto DMA version control register	0x00FC	R/W
Interrupt Registers			
CRYPTO_DMA_INT_RAW_REG	Raw interrupt status	0x0004	RO
CRYPTO_DMA_INT_ST_REG	Masked interrupt status	0x0008	RO
CRYPTO_DMA_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
CRYPTO_DMA_INT_CLR_REG	Interrupt clear bits	0x0010	WO
Status Registers			
CRYPTO_DMA_OUT_STATUS_REG	TX FIFO status register	0x0014	RO
CRYPTO_DMA_IN_STATUS_REG	RX FIFO status register	0x001C	RO
CRYPTO_DMA_STATE0_REG	Status register of receiving data	0x0030	RO
CRYPTO_DMA_STATE1_REG	Status register of transmitting data	0x0034	RO
CRYPTO_DMA_OUT_EOF_DES_ADDR_REG	Transmit descriptor address when EOF occurs	0x0038	RO
CRYPTO_DMA_IN_SUC_EOF_DES_ADDR_REG	Receive descriptor address when EOF occurs	0x003C	RO
CRYPTO_DMA_IN_ERR_EOF_DES_ADDR_REG	Receive descriptor address when errors occur	0x0040	RO
CRYPTO_DMA_OUT_EOF_BFR_DES_ADDR_REG	Transmit descriptor address before the last transmit descriptor	0x0044	RO
CRYPTO_DMA_IN_DSCR_REG	Address of current receive descriptor	0x004C	RO

Name	Description	Address	Access
CRYPTO_DMA_IN_DSCR_BF0_REG	Address of last receive descriptor	0x0050	RO
CRYPTO_DMA_OUT_DSCR_REG	Address of current transmit descriptor	0x0058	RO
CRYPTO_DMA_OUT_DSCR_BF0_REG	Address of last transmit descriptor	0x005C	RO

2.13 Registers

Register 2.1: CP_DMA_INT_RAW_REG (0x0000)

31–8	7	6	5	4	3	2	1	0	
(reserved)	CP_DMA_OUT_TOTAL_EOF_INT_RAW	CP_DMA_IN_DSCR_EMPTY_INT_RAW	CP_DMA_OUT_DSCR_ERR_INT_RAW	CP_DMA_IN_DSCR_ERR_INT_RAW	CP_DMA_OUT_EOF_INT_RAW	CP_DMA_OUT_DONE_INT_RAW	CP_DMA_IN_SUC_EOF_INT_RAW	CP_DMA_IN_DONE_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

CP_DMA_IN_DONE_INT_RAW This is the interrupt raw bit. Triggered when the last data of frame is received or the receive buffer is full indicated by receive descriptor. (RO)

CP_DMA_IN_SUC_EOF_INT_RAW This is the interrupt raw bit. Triggered when the last data of one frame is received. (RO)

CP_DMA_OUT_DONE_INT_RAW This is the interrupt raw bit. Triggered when all data indicated by one transmit descriptor has been pushed into TX FIFO. (RO)

CP_DMA_OUT_EOF_INT_RAW This is the interrupt raw bit. Triggered when the last data with EOF flag has been pushed into TX FIFO. (RO)

CP_DMA_IN_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when detecting receive descriptor error, including owner error, the second and third word error of receive descriptor. (RO)

CP_DMA_OUT_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when detecting transmit descriptor error, including owner error, the second and third word error of transmit descriptor. (RO)

CP_DMA_IN_DSCR_EMPTY_INT_RAW This is the interrupt raw bit. Triggered when receiving data is completed and no more receive descriptor. (RO)

CP_DMA_OUT_TOTAL_EOF_INT_RAW This is the interrupt raw bit. Triggered when data corresponding to all transmit descriptors and the last descriptor with valid EOF is transmitted out. (RO)

Register 2.2: CP_DMA_INT_ST_REG (0x0004)

31–8	7	6	5	4	3	2	1	0	
(reserved)	CP_DMA_OUT_TOTAL_EOF_INT_ST	CP_DMA_IN_DSCR_EMPTY_INT_ST	CP_DMA_OUT_DSCR_ERR_INT_ST	CP_DMA_IN_DSCR_ERR_INT_ST	CP_DMA_OUT_EOF_INT_ST	CP_DMA_OUT_DONE_INT_ST	CP_DMA_IN_SUC_EOF_INT_ST	CP_DMA_IN_DONE_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

CP_DMA_IN_DONE_INT_ST This is the masked interrupt bit for CP_DMA_IN_DONE_INT interrupt when CP_DMA_IN_DONE_INT_ENA is set. (RO)

CP_DMA_IN_SUC_EOF_INT_ST This is the masked interrupt bit for CP_DMA_IN_SUC_EOF_INT interrupt when CP_DMA_IN_SUC_EOF_INT_ENA is set. (RO)

CP_DMA_OUT_DONE_INT_ST This is the masked interrupt bit for CP_DMA_OUT_DONE_INT interrupt when CP_DMA_OUT_DONE_INT_ENA is set. (RO)

CP_DMA_OUT_EOF_INT_ST This is the masked interrupt bit for CP_DMA_OUT_EOF_INT interrupt when CP_DMA_OUT_EOF_INT_ENA is set. (RO)

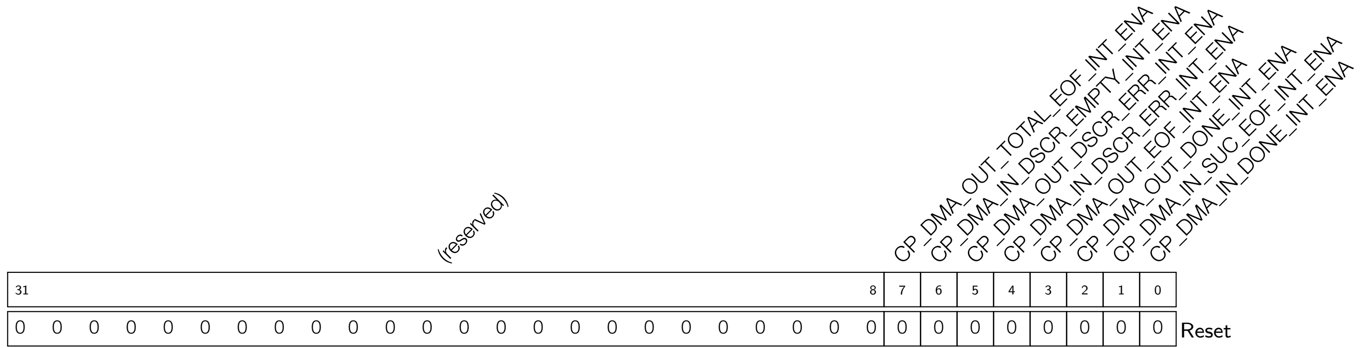
CP_DMA_IN_DSCR_ERR_INT_ST This is the masked interrupt bit for CP_DMA_IN_DSCR_ERR_INT interrupt when CP_DMA_IN_DSCR_ERR_INT_ENA is set. (RO)

CP_DMA_OUT_DSCR_ERR_INT_ST This is the masked interrupt bit for CP_DMA_OUT_DSCR_ERR_INT interrupt when CP_DMA_OUT_DSCR_ERR_INT_ENA is set. (RO)

CP_DMA_IN_DSCR_EMPTY_INT_ST This is the masked interrupt bit for CP_DMA_IN_DSCR_EMPTY_INT interrupt when CP_DMA_IN_DSCR_EMPTY_INT_ENA is set. (RO)

CP_DMA_OUT_TOTAL_EOF_INT_ST This is the masked interrupt bit for CP_DMA_OUT_TOTAL_EOF_INT interrupt when CP_DMA_OUT_TOTAL_EOF_INT_ENA is set. (RO)

Register 2.3: CP_DMA_INT_ENA_REG (0x0008)



CP_DMA_IN_DONE_INT_ENA This is the interrupt enable bit for CP_DMA_IN_DONE_INT interrupt. (R/W)

CP_DMA_IN_SUC_EOF_INT_ENA This is the interrupt enable bit for CP_DMA_IN_SUC_EOF_INT interrupt. (R/W)

CP_DMA_OUT_DONE_INT_ENA This is the interrupt enable bit for CP_DMA_OUT_DONE_INT interrupt. (R/W)

CP_DMA_OUT_EOF_INT_ENA This is the interrupt enable bit for CP_DMA_OUT_EOF_INT interrupt. (R/W)

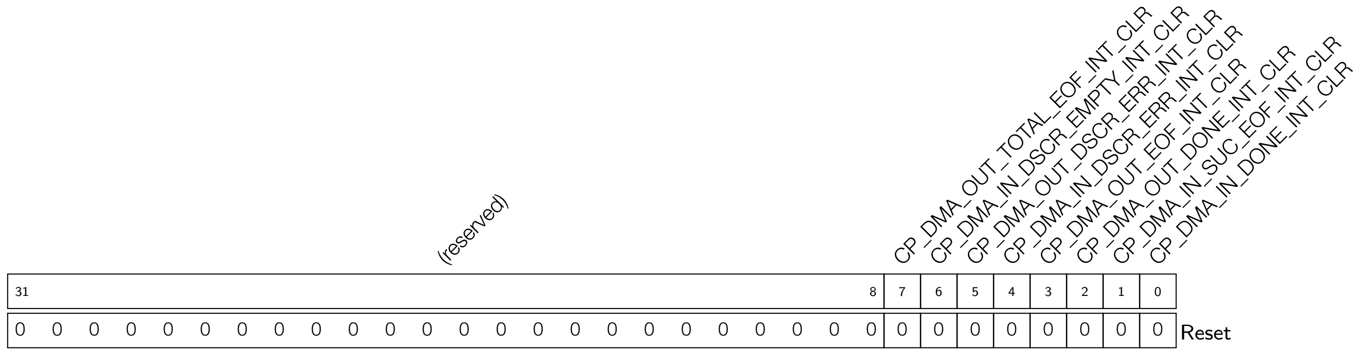
CP_DMA_IN_DSCR_ERR_INT_ENA This is the interrupt enable bit for CP_DMA_IN_DSCR_ERR_INT interrupt. (R/W)

CP_DMA_OUT_DSCR_ERR_INT_ENA This is the interrupt enable bit for CP_DMA_OUT_DSCR_ERR_INT interrupt. (R/W)

CP_DMA_IN_DSCR_EMPTY_INT_ENA This is the interrupt enable bit for CP_DMA_IN_DSCR_EMPTY_INT interrupt. (R/W)

CP_DMA_OUT_TOTAL_EOF_INT_ENA This is the interrupt enable bit for CP_DMA_OUT_TOTAL_EOF_INT interrupt. (R/W)

Register 2.4: CP_DMA_INT_CLR_REG (0x000C)



CP_DMA_IN_DONE_INT_CLR Set this bit to clear CP_DMA_IN_DONE_INT INTERRUPT. (WO)

CP_DMA_IN_SUC_EOF_INT_CLR Set this bit to clear CP_DMA_IN_SUC_EOF_INT interrupt. (WO)

CP_DMA_OUT_DONE_INT_CLR Set this bit to clear CP_DMA_OUT_DONE_INT interrupt. (WO)

CP_DMA_OUT_EOF_INT_CLR Set this bit to clear CP_DMA_OUT_EOF_INT interrupt. (WO)

CP_DMA_IN_DSCR_ERR_INT_CLR Set this bit to clear CP_DMA_IN_DSCR_ERR_INT interrupt. (WO)

CP_DMA_OUT_DSCR_ERR_INT_CLR Set this bit to clear CP_DMA_OUT_DSCR_ERR_INT interrupt. (WO)

CP_DMA_IN_DSCR_EMPTY_INT_CLR Set this bit to clear CP_DMA_IN_DSCR_EMPTY_INT interrupt. (WO)

CP_DMA_OUT_TOTAL_EOF_INT_CLR Set this bit to clear CP_DMA_OUT_TOTAL_EOF_INT interrupt. (WO)

Register 2.5: CP_DMA_OUT_LINK_REG (0x0010)

CP_DMA_OUTLINK_ADDR This register is used to specify the least significant 20 bits of the first transmit descriptor's address. (R/W)

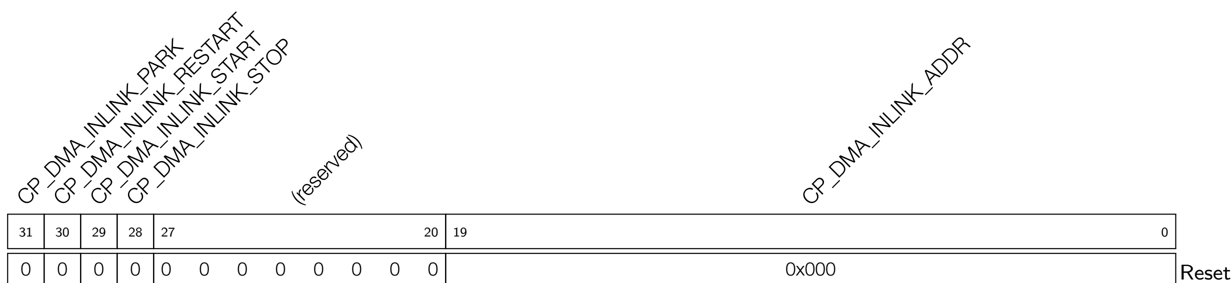
CP_DMA_OUTLINK_STOP Set this bit to stop DMA from reading transmit descriptors after finishing the current data transaction. (R/W)

CP_DMA_OUTLINK_START Set this bit to start a new transmit descriptor. (R/W)

CP_DMA_OUTLINK_RESTART Set this bit to restart the transmit descriptor from the last address. (R/W)

CP_DMA_OUTLINK_PARK 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

Register 2.6: CP_DMA_IN_LINK_REG (0x0014)



CP_DMA_INLINK_ADDR This register is used to specify the least significant 20 bits of the first receive descriptor's address. (R/W)

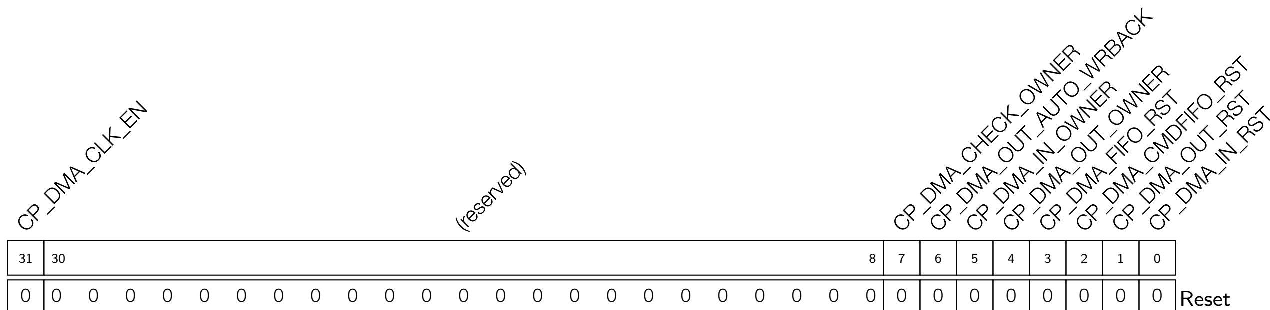
CP_DMA_INLINK_STOP Set this bit to stop DMA from reading receive descriptors after finishing the current data transaction. (R/W)

CP_DMA_INLINK_START Set this bit to enable DMA to read receive descriptors. (R/W)

CP_DMA_INLINK_RESTART Set this bit to restart new receive descriptors. (R/W)

CP_DMA_INLINK_PARK 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

Register 2.7: CP_DMA_CONF_REG (0x003C)



CP_DMA_IN_RST Set this bit to reset in DMA FSM. (R/W)

CP_DMA_OUT_RST Set this bit to reset out DMA FSM. (R/W)

CP_DMA_CMDFIFO_RST Set this bit to reset in_cmd FIFO and out_cmd FIFO. (R/W)

CP_DMA_FIFO_RST Set this bit to reset data in RX FIFO. (R/W)

CP_DMA_OUT_OWNER This is used to configure the owner bit in transmit descriptor. This is effective only when you set CP_DMA_OUT_AUTO_WRBACK. (R/W)

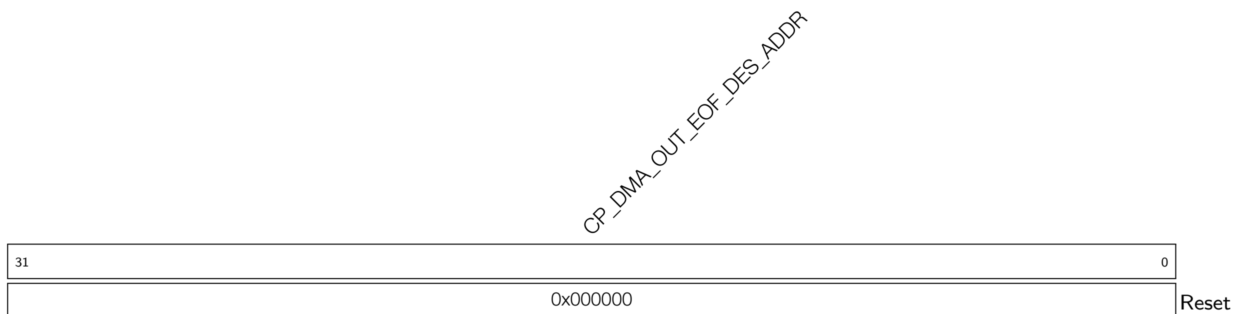
CP_DMA_IN_OWNER This is used to configure the owner bit in receive descriptor. (R/W)

CP_DMA_OUT_AUTO_WRBACK This bit is used to write back out descriptor when hardware has already used this descriptor. (R/W)

CP_DMA_CHECK_OWNER Set this bit to enable owner bit check in descriptor. (R/W)

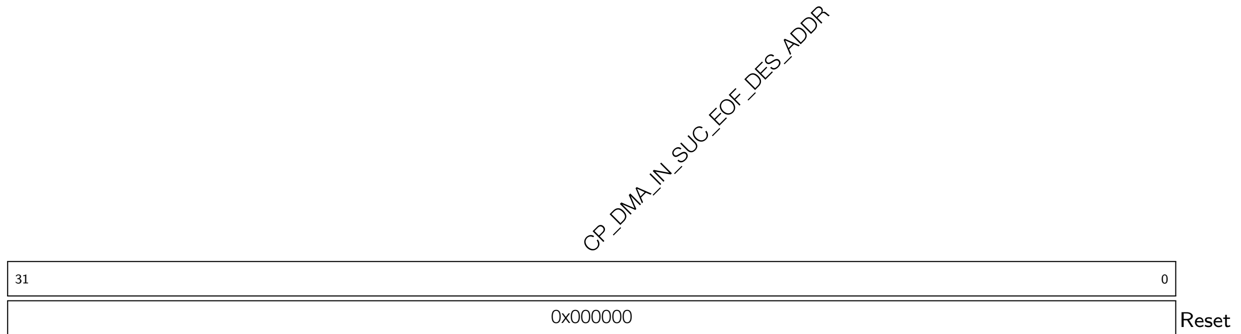
CP_DMA_CLK_EN 1’b1: Force clock on for register. 1’b0: Support clock only when application writes registers. (R/W)

Register 2.8: CP_DMA_OUT_EOF_DES_ADDR_REG (0x0018)



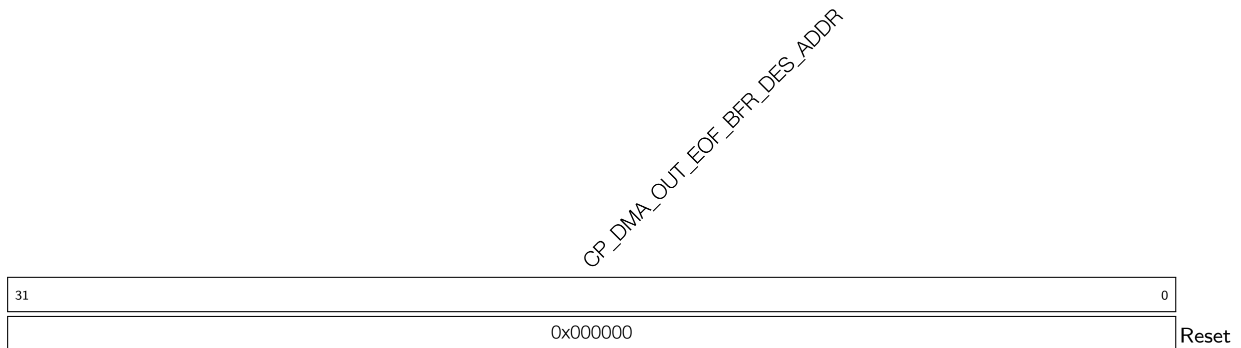
CP_DMA_OUT_EOF_DES_ADDR This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.9: CP_DMA_IN_EOF_DES_ADDR_REG (0x001C)



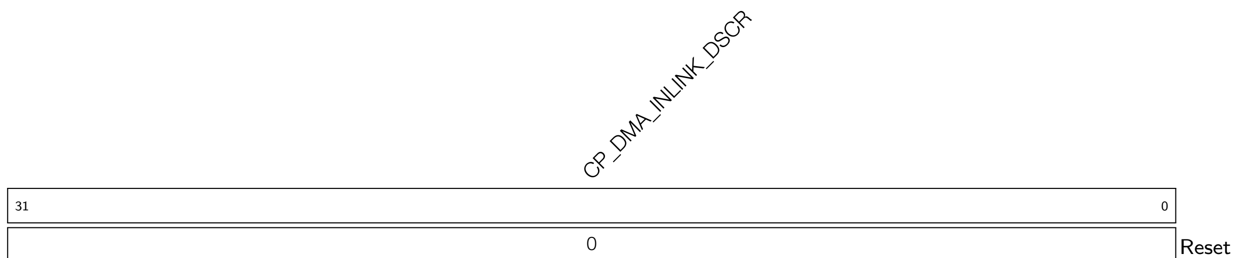
CP_DMA_IN_SUC_EOF_DES_ADDR This register stores the address of the receive descriptor when received successful EOF. (RO)

Register 2.10: CP_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x0020)



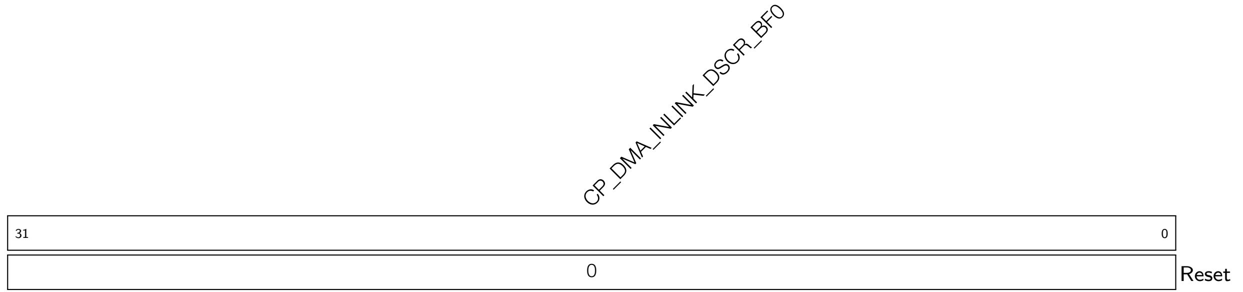
CP_DMA_OUT_EOF_BFR_DES_ADDR This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 2.11: CP_DMA_INLINK_DSCR_REG (0x0024)



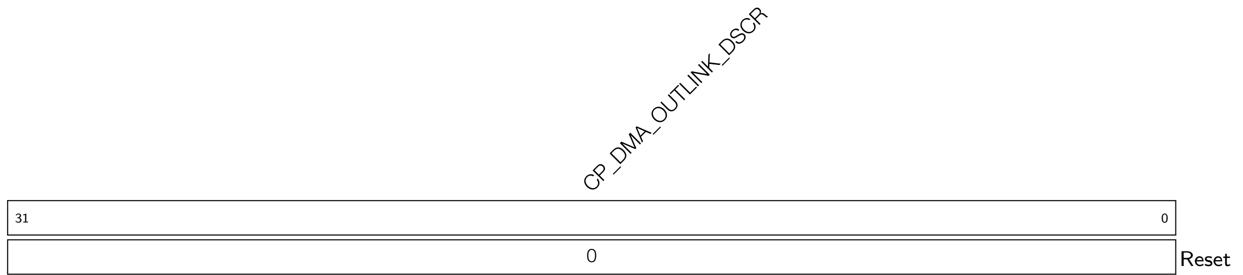
CP_DMA_INLINK_DSCR The address of the current receive descriptor x. (RO)

Register 2.12: CP_DMA_INLINK_DSCR_BF0_REG (0x0028)



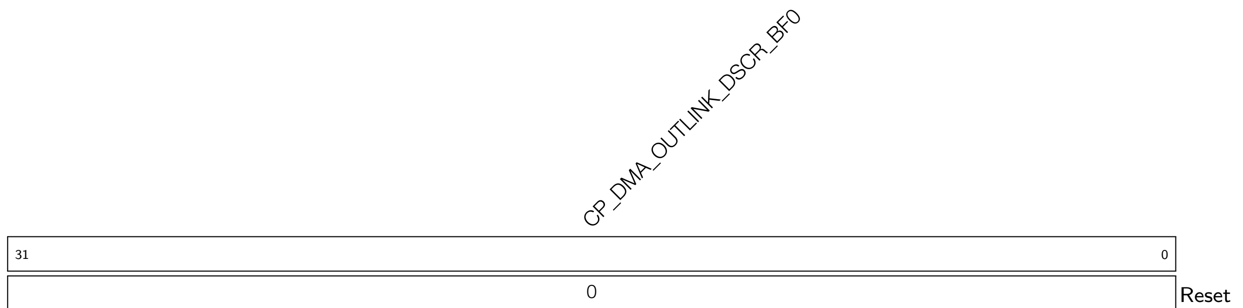
CP_DMA_INLINK_DSCR_BF0 The address of the last receive descriptor x-1. (RO)

Register 2.13: CP_DMA_OUTLINK_DSCR_REG (0x0030)



CP_DMA_OUTLINK_DSCR The address of the current transmit descriptor y. (RO)

Register 2.14: CP_DMA_OUTLINK_DSCR_BF0_REG (0x0034)



CP_DMA_OUTLINK_DSCR_BF0 The address of the last transmit descriptor y-1. (RO)

Register 2.15: CP_DMA_IN_ST_REG (0x0040)

(reserved)	CP_DMA_FIFO_EMPTY	CP_DMA_IN_STATE	CP_DMA_IN_DSCR_STATE	CP_DMA_INLINK_DSCR_ADDR	
31 … 24	23	22 … 20	19 … 18	17 … 0	
0 0 0 0 0 0 0 0	0	0	0	0	Reset

CP_DMA_INLINK_DSCR_ADDR This register stores the current receive descriptor's address. (RO)

CP_DMA_IN_DSCR_STATE Reserved. (RO)

CP_DMA_IN_STATE Reserved. (RO)

CP_DMA_FIFO_EMPTY Copy DMA FIFO empty signal. (RO)

Register 2.16: CP_DMA_OUT_ST_REG (0x0044)

(reserved)	CP_DMA_FIFO_FULL	CP_DMA_OUT_STATE	CP_DMA_OUT_DSCR_STATE	CP_DMA_OUTLINK_DSCR_ADDR	
31 … 24	23	22 … 20	19 … 18	17 … 0	
0 0 0 0 0 0 0 0	0	0	0	0	Reset

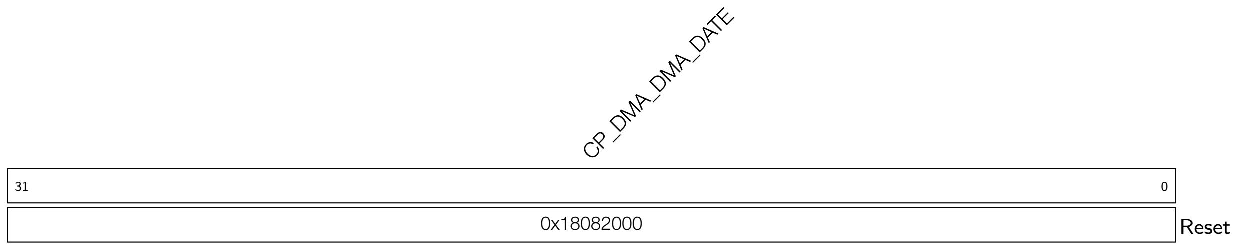
CP_DMA_OUTLINK_DSCR_ADDR This register stores the current transmit descriptor's address. (RO)

CP_DMA_OUT_DSCR_STATE Reserved. (RO)

CP_DMA_OUT_STATE Reserved. (RO)

CP_DMA_FIFO_FULL Copy DMA FIFO full signal. (RO)

Register 2.17: CP_DMA_DATE_REG (0x00FC)



CP_DMA_DMA_DATE This is the version control register. (R/W)

Register 2.18: CRYPTO_DMA_CONF0_REG (0x0000)

Bits	31–13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	(reserved)	CRYPTO_DMA_MEM_TRANS_EN	CRYPTO_DMA_OUT_DATA_BURST_EN	CRYPTO_DMA_INDSCR_BURST_EN	CRYPTO_DMA_OUTDSCR_BURST_EN	CRYPTO_DMA_OUT_EOF_MODE	CRYPTO_DMA_OUT_NO_RESTART_CLR	CRYPTO_DMA_OUT_AUTO_WRBACK	CRYPTO_DMA_OUT_LOOP_TEST	CRYPTO_DMA_IN_LOOP_TEST	CRYPTO_DMA_AHBM_RST	CRYPTO_DMA_AHBM_FIFO_RST	CRYPTO_DMA_OUT_RST	CRYPTO_DMA_IN_RST
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	1	0	0	0	0	0	0	0	0

CRYPTO_DMA_IN_RST This bit is used to reset crypto DMA in FSM and RX FIFO pointer. (R/W)

CRYPTO_DMA_OUT_RST This bit is used to reset crypto DMA out FSM and TX FIFO pointer. (R/W)

CRYPTO_DMA_AHBM_FIFO_RST This bit is used to reset crypto DMA AHB master FIFO pointer. (R/W)

CRYPTO_DMA_AHBM_RST Reset crypto DMA AHB master. (R/W)

CRYPTO_DMA_IN_LOOP_TEST Reserved (R/W)

CRYPTO_DMA_OUT_LOOP_TEST Reserved (R/W)

CRYPTO_DMA_OUT_AUTO_WRBACK Set this bit to enable automatic outlink-writeback when all the data in TX Buffer has been transmitted. (R/W)

CRYPTO_DMA_OUT_NO_RESTART_CLR Reserved (R/W)

CRYPTO_DMA_OUT_EOF_MODE Out EOF flag generation mode of TX FIFO. 1: EOF flag of TX is generated when the last data with EOF would be transmitted has been popped from FIFO of Crypto DMA; 0: EOF flag is generated when the last data with EOF would be transmitted has been pushed into FIFO of Crypto DMA. (R/W)

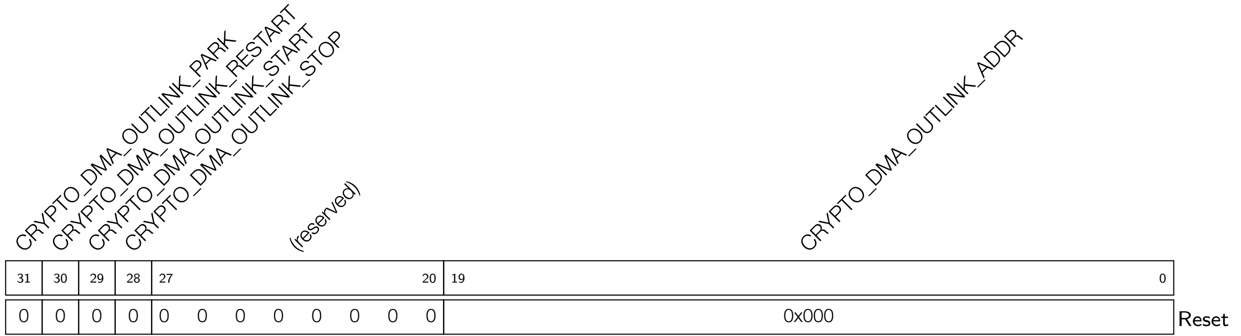
CRYPTO_DMA_OUTDSCR_BURST_EN Set this bit to enable INCR burst transfer when TX FIFO reads descriptor from internal RAM. (R/W)

CRYPTO_DMA_INDSCR_BURST_EN Set this bit to enable INCR burst transfer when RX FIFO reads descriptor from internal RAM. (R/W)

CRYPTO_DMA_OUT_DATA_BURST_EN Set this bit to enable INCR burst transfer when TX FIFO reads data from internal RAM. (R/W)

CRYPTO_DMA_MEM_TRANS_EN Set this bit to enable automatic transmitting data from memory to memory via DMA. (R/W)

Register 2.19: CRYPTO_DMA_OUT_LINK_REG (0x0024)



CRYPTO_DMA_OUTLINK_ADDR This register stores the 20 least significant bits of the first transmit descriptor's address. (R/W)

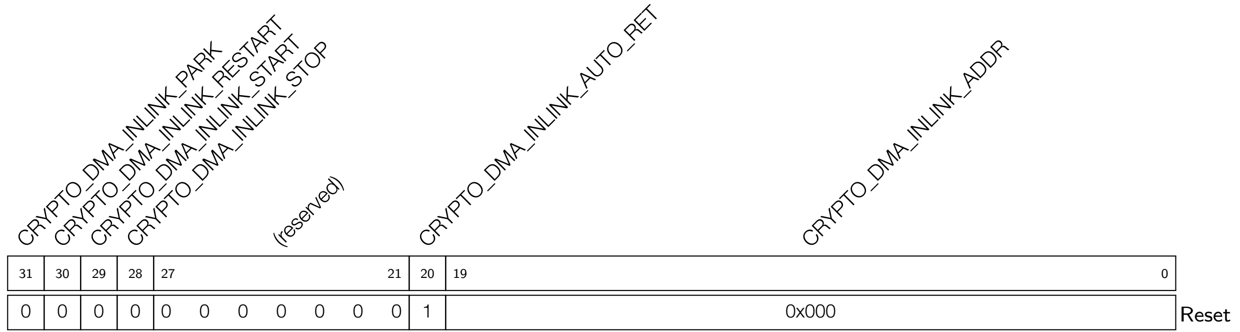
CRYPTO_DMA_OUTLINK_STOP Set this bit to stop DMA from reading transmit descriptors after finishing the current data transaction. (R/W)

CRYPTO_DMA_OUTLINK_START Set this bit to enable DMA to read transmit descriptors. (R/W)

CRYPTO_DMA_OUTLINK_RESTART Set this bit to restart a new outlink from the last address. (R/W)

CRYPTO_DMA_OUTLINK_PARK 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

Register 2.20: CRYPTO_DMA_IN_LINK_REG (0x0028)



CRYPTO_DMA_INLINK_ADDR This register stores the 20 least significant bits of the first receive descriptor's address. (R/W)

CRYPTO_DMA_INLINK_AUTO_RET Reserved (R/W)

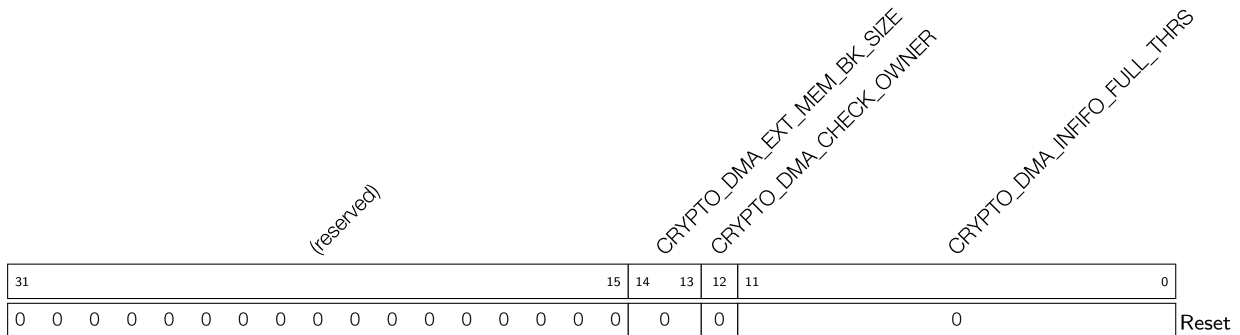
CRYPTO_DMA_INLINK_STOP Set this bit to stop DMA from reading receive descriptors after finishing the current data transaction. (R/W)

CRYPTO_DMA_INLINK_START Set this bit to enable DMA to read receive descriptors. (R/W)

CRYPTO_DMA_INLINK_RESTART Set this bit to mount a new receive descriptor. (R/W)

CRYPTO_DMA_INLINK_PARK 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

Register 2.21: CRYPTO_DMA_CONF1_REG (0x002C)

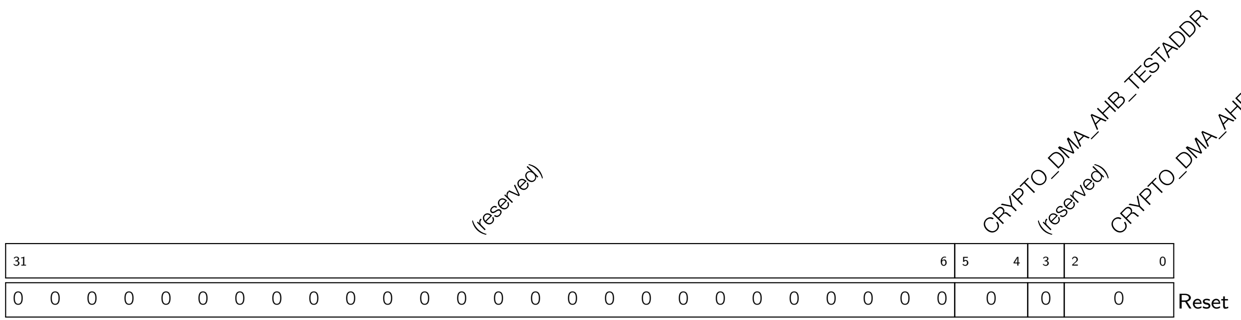


CRYPTO_DMA_INFIFO_FULL_THRS This register is used to generate the CRYPTO_DMA_INFIFO_FULL_WM_INT interrupt when the byte number is up to the value of the register. (R/W)

CRYPTO_DMA_CHECK_OWNER Set this bit to enable checking the owner attribute of the link descriptor. (R/W)

CRYPTO_DMA_EXT_MEM_BK_SIZE DMA access external memory block size. 0: 16 bytes; 1: 32 bytes; 2:64 bytes; 3:Reserved. (R/W)

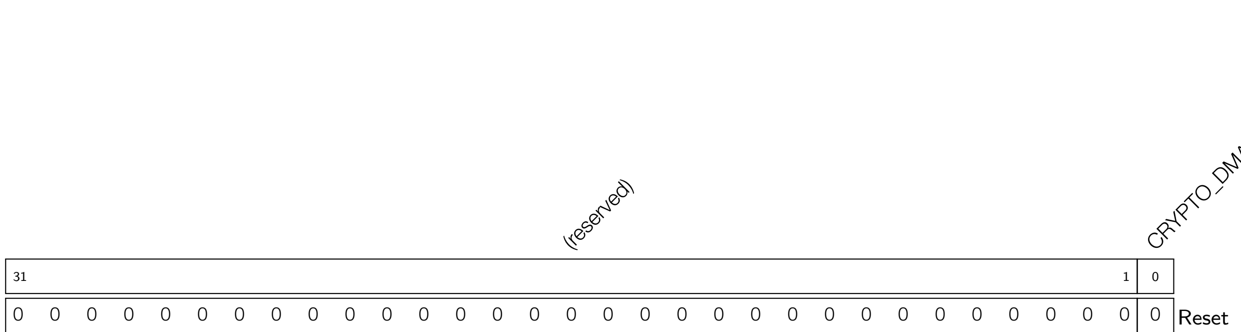
Register 2.22: CRYPTO_DMA_AHB_TEST_REG (0x0048)



CRYPTO_DMA_AHB_TESTMODE Reserved (R/W)

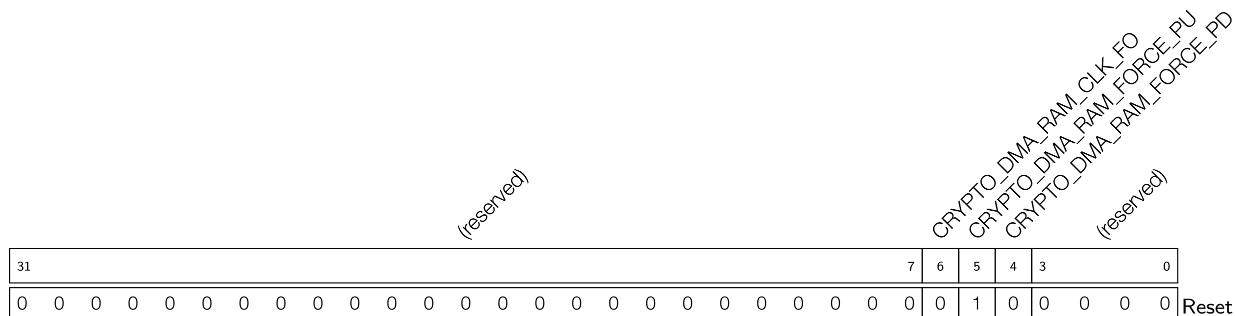
CRYPTO_DMA_AHB_TESTADDR Reserved (R/W)

Register 2.23: CRYPTO_DMA_AES_SHA_SELECT_REG (0x0064)



CRYPTO_DMA_AES_SHA_SELECT Select one between AES and SHA to use DMA. 0: AES. 1:SHA. (R/W)

Register 2.24: CRYPTO_DMA_PD_CONF_REG (0x0068)

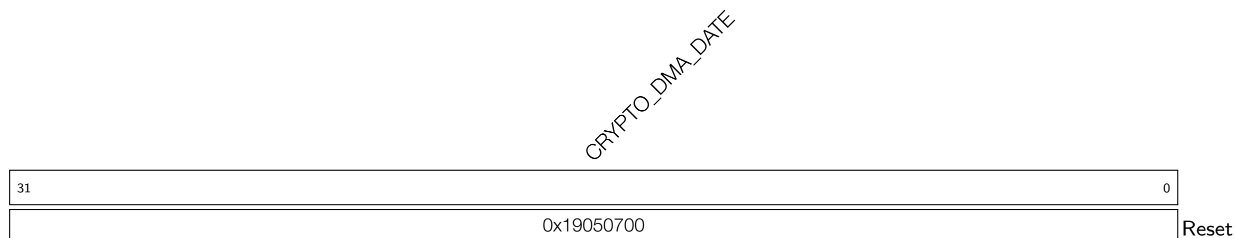


CRYPTO_DMA_RAM_FORCE_PD Force power down signal to RAM. 0: force RAM power up; 1: only when CRYPTO_DMA_RAM_FORCE_PU is 0, power down RAM. (R/W)

CRYPTO_DMA_RAM_FORCE_PU Force power up signal to RAM. 0: only when CRYPTO_DMA_RAM_FORCE_PD is 1, power down RAM; 1: force RAM power up. (R/W)

CRYPTO_DMA_RAM_CLK_FO 1: Force to open the clock and bypass the gate-clock when accessing the RAM in DMA. 0: A gate-clock will be used when accessing the RAM in DMA. (R/W)

Register 2.25: CRYPTO_DMA_DATE_REG (0x00FC)



CRYPTO_DMA_DATE This is the version control register. (R/W)

Register 2.26: CRYPTO_DMA_INT_RAW_REG (0x0004)

31–10	9	8	7	6	5	4	3	2	1	0	
(reserved)	CRYPTO_DMA_INFIFO_FULL_WM_INT_RAW	CRYPTO_DMA_OUT_TOTAL_EOF_INT_RAW	CRYPTO_DMA_IN_DSCR_EMPTY_INT_RAW	CRYPTO_DMA_OUT_DSCR_ERR_INT_RAW	CRYPTO_DMA_IN_DSCR_ERR_INT_RAW	CRYPTO_DMA_OUT_EOF_INT_RAW	CRYPTO_DMA_OUT_DONE_INT_RAW	CRYPTO_DMA_IN_ERR_EOF_INT_RAW	CRYPTO_DMA_IN_SUC_EOF_INT_RAW	CRYPTO_DMA_IN_DONE_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	Reset

CRYPTO_DMA_IN_DONE_INT_RAW The raw interrupt status. Set when the last data of one frame is received or the receive buffer is full indicated by receive descriptor. (RO)

CRYPTO_DMA_IN_SUC_EOF_INT_RAW The raw interrupt status. Set when the last data of one frame is received by Crypto DMA RX FIFO. (RO)

CRYPTO_DMA_IN_ERR_EOF_INT_RAW Reserved (RO)

CRYPTO_DMA_OUT_DONE_INT_RAW The raw interrupt status. Set when all data indicated by one transmit descriptor has been pushed into TX FIFO. (RO)

CRYPTO_DMA_OUT_EOF_INT_RAW The raw interrupt status. Set when Out EOF flag is generated. (RO)

CRYPTO_DMA_IN_DSCR_ERR_INT_RAW The raw interrupt status. Set when detecting receive descriptor error, including owner error, the second and third word error of receive descriptor. (RO)

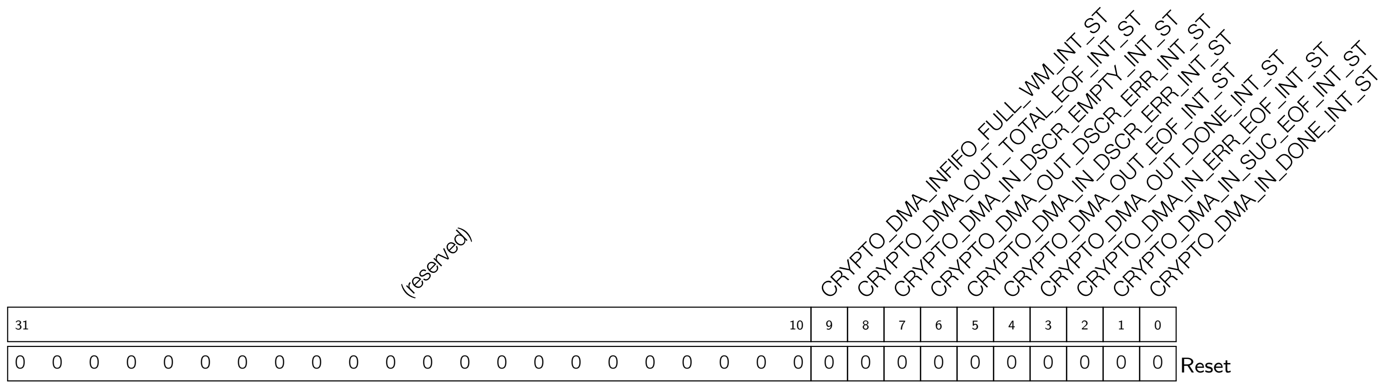
CRYPTO_DMA_OUT_DSCR_ERR_INT_RAW The raw interrupt status. Set when detecting transmit descriptor error, including owner error, the second and third word error of transmit descriptor. (RO)

CRYPTO_DMA_IN_DSCR_EMPTY_INT_RAW The raw interrupt status. Set when receiving data is completed and no more receive descriptor. (RO)

CRYPTO_DMA_OUT_TOTAL_EOF_INT_RAW The raw interrupt status. Set when data corresponding to all transmit descriptor and the last descriptor with valid EOF is transmitted out. (RO)

CRYPTO_DMA_INFIFO_FULL_WM_INT_RAW The raw interrupt status. Set when received data byte number is up to threshold configured by CRYPTO_DMA_INFIFO_FULL_THRS in RX FIFO. (RO)

Register 2.27: CRYPTO_DMA_INT_ST_REG (0x0008)



CRYPTO_DMA_IN_DONE_INT_ST The masked interrupt status bit for the CRYPTO_DMA_IN_DONE_INT interrupt. (RO)

CRYPTO_DMA_IN_SUC_EOF_INT_ST The masked interrupt status bit for the CRYPTO_DMA_IN_SUC_EOF_INT interrupt. (RO)

CRYPTO_DMA_IN_ERR_EOF_INT_ST Reserved (RO)

CRYPTO_DMA_OUT_DONE_INT_ST The masked interrupt status bit for the CRYPTO_DMA_OUT_DONE_INT interrupt. (RO)

CRYPTO_DMA_OUT_EOF_INT_ST The masked interrupt status bit for the CRYPTO_DMA_OUT_EOF_INT interrupt. (RO)

CRYPTO_DMA_IN_DSCR_ERR_INT_ST The masked interrupt status bit for the CRYPTO_DMA_IN_DSCR_ERR_INT interrupt. (RO)

CRYPTO_DMA_OUT_DSCR_ERR_INT_ST The masked interrupt status bit for the CRYPTO_DMA_OUT_DSCR_ERR_INT interrupt. (RO)

CRYPTO_DMA_IN_DSCR_EMPTY_INT_ST The masked interrupt status bit for the CRYPTO_DMA_IN_DSCR_EMPTY_INT interrupt. (RO)

CRYPTO_DMA_OUT_TOTAL_EOF_INT_ST The masked interrupt status bit for the CRYPTO_DMA_OUT_TOTAL_EOF_INT interrupt. (RO)

CRYPTO_DMA_INFIFO_FULL_WM_INT_ST The masked interrupt status bit for the CRYPTO_DMA_INFIFO_FULL_WM_INT interrupt. (RO)

Register 2.28: CRYPTO_DMA_INT_ENA_REG (0x000C)

31–10	9	8	7	6	5	4	3	2	1	0	
(reserved)	CRYPTO_DMA_INFIFO_FULL_WM_INT_ENA	CRYPTO_DMA_OUT_TOTAL_EOF_INT_ENA	CRYPTO_DMA_IN_DSCR_EMPTY_INT_ENA	CRYPTO_DMA_OUT_DSCR_ERR_INT_ENA	CRYPTO_DMA_IN_DSCR_ERR_INT_ENA	CRYPTO_DMA_OUT_EOF_INT_ENA	CRYPTO_DMA_OUT_DONE_INT_ENA	CRYPTO_DMA_IN_ERR_EOF_INT_ENA	CRYPTO_DMA_IN_SUC_EOF_INT_ENA	CRYPTO_DMA_IN_DONE_INT_ENA	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	Reset

CRYPTO_DMA_IN_DONE_INT_ENA The interrupt enable bit for the CRYPTO_DMA_IN_DONE_INT interrupt. (R/W)

CRYPTO_DMA_IN_SUC_EOF_INT_ENA The interrupt enable bit for the CRYPTO_DMA_IN_SUC_EOF_INT interrupt. (R/W)

CRYPTO_DMA_IN_ERR_EOF_INT_ENA Reserved (R/W)

CRYPTO_DMA_OUT_DONE_INT_ENA The interrupt enable bit for the CRYPTO_DMA_OUT_DONE_INT interrupt. (R/W)

CRYPTO_DMA_OUT_EOF_INT_ENA The interrupt enable bit for the CRYPTO_DMA_OUT_EOF_INT interrupt. (R/W)

CRYPTO_DMA_IN_DSCR_ERR_INT_ENA The interrupt enable bit for the CRYPTO_DMA_IN_DSCR_ERR_INT interrupt. (R/W)

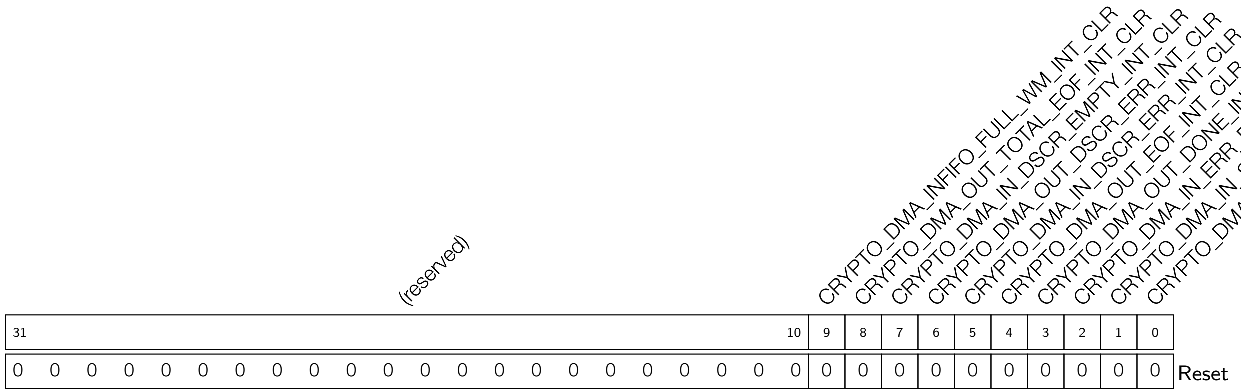
CRYPTO_DMA_OUT_DSCR_ERR_INT_ENA The interrupt enable bit for the CRYPTO_DMA_OUT_DSCR_ERR_INT interrupt. (R/W)

CRYPTO_DMA_IN_DSCR_EMPTY_INT_ENA The interrupt enable bit for the CRYPTO_DMA_IN_DSCR_EMPTY_INT interrupt. (R/W)

CRYPTO_DMA_OUT_TOTAL_EOF_INT_ENA The interrupt enable bit for the CRYPTO_DMA_OUT_TOTAL_EOF_INT interrupt. (R/W)

CRYPTO_DMA_INFIFO_FULL_WM_INT_ENA The interrupt enable bit for the CRYPTO_DMA_INFIFO_FULL_WM_INT interrupt. (R/W)

Register 2.29: CRYPTO_DMA_INT_CLR_REG (0x0010)



CRYPTO_DMA_IN_DONE_INT_CLR Set this bit to clear the CRYPTO_DMA_IN_DONE_INT interrupt. (WO)

CRYPTO_DMA_IN_SUC_EOF_INT_CLR Set this bit to clear the CRYPTO_DMA_IN_SUC_EOF_INT interrupt. (WO)

CRYPTO_DMA_IN_ERR_EOF_INT_CLR Reserved (WO)

CRYPTO_DMA_OUT_DONE_INT_CLR Set this bit to clear the CRYPTO_DMA_OUT_DONE_INT interrupt. (WO)

CRYPTO_DMA_OUT_EOF_INT_CLR Set this bit to clear the CRYPTO_DMA_OUT_EOF_INT interrupt. (WO)

CRYPTO_DMA_IN_DSCR_ERR_INT_CLR Set this bit to clear the CRYPTO_DMA_IN_DSCR_ERR_INT interrupt. (WO)

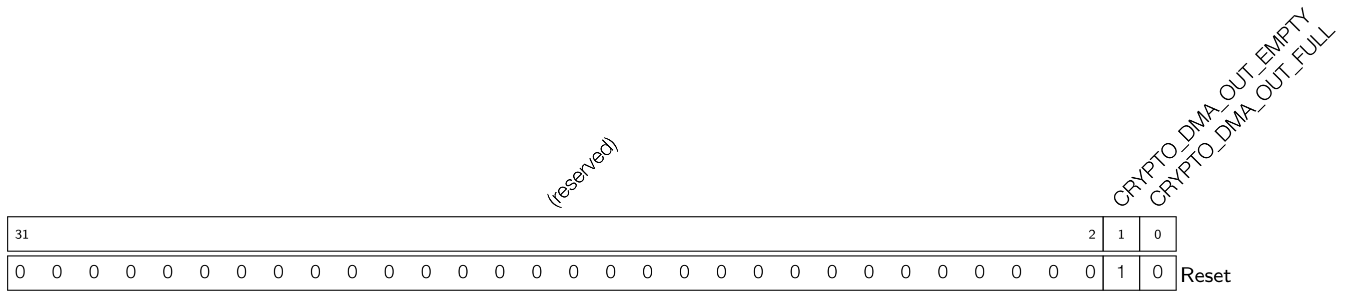
CRYPTO_DMA_OUT_DSCR_ERR_INT_CLR Set this bit to clear the CRYPTO_DMA_OUT_DSCR_ERR_INT interrupt. (WO)

CRYPTO_DMA_IN_DSCR_EMPTY_INT_CLR Set this bit to clear the CRYPTO_DMA_IN_DSCR_EMPTY_INT interrupt. (WO)

CRYPTO_DMA_OUT_TOTAL_EOF_INT_CLR Set this bit to clear the CRYPTO_DMA_OUT_TOTAL_EOF_INT interrupt. (WO)

CRYPTO_DMA_INFIFO_FULL_WM_INT_CLR Set this bit to clear the CRYPTO_DMA_INFIFO_FULL_WM_INT interrupt. (WO)

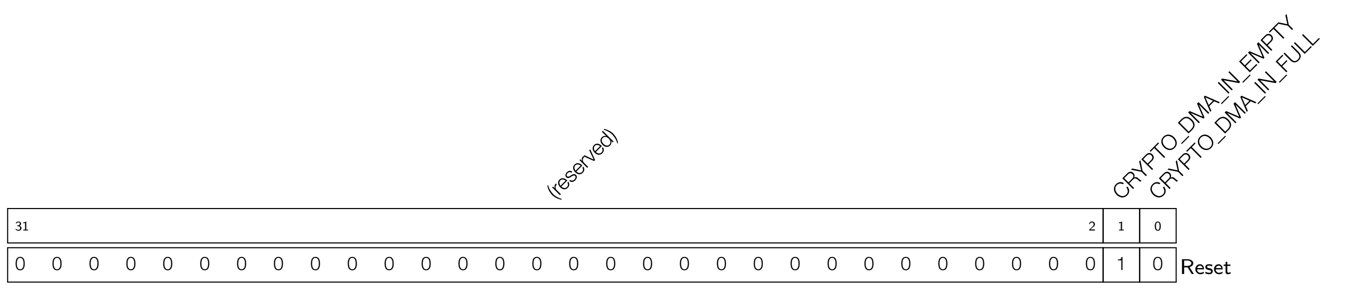
Register 2.30: CRYPTO_DMA_OUT_STATUS_REG (0x0014)



CRYPTO_DMA_OUT_FULL 1: DMA TX FIFO is full. (RO)

CRYPTO_DMA_OUT_EMPTY 1: DMA TX FIFO is empty. (RO)

Register 2.31: CRYPTO_DMA_IN_STATUS_REG (0x001C)



CRYPTO_DMA_IN_FULL 1: DMA RX FIFO is full. (RO)

CRYPTO_DMA_IN_EMPTY 1: DMA RX FIFO is empty. (RO)

Register 2.32: CRYPTO_DMA_STATE0_REG (0x0030)

(reserved)	CRYPTO_DMA_INFIFO_CNT_DEBUG	CRYPTO_DMA_IN_STATE	CRYPTO_DMA_IN_DSCR_STATE	CRYPTO_DMA_INLINK_DSCR_ADDR	
31 27	26 23	22 20	19 18	17 0	
0 0 0 0 0	0	0	0	0	Reset

CRYPTO_DMA_INLINK_DSCR_ADDR This register stores the current receive descriptor's address. (RO)

CRYPTO_DMA_IN_DSCR_STATE Reserved (RO)

CRYPTO_DMA_IN_STATE Reserved (RO)

CRYPTO_DMA_INFIFO_CNT_DEBUG This register stores the byte number of the data in the receive descriptor's FIFO. (RO)

Register 2.33: CRYPTO_DMA_STATE1_REG (0x0034)

(reserved)	CRYPTO_DMA_OUTFIFO_CNT	CRYPTO_DMA_OUT_STATE	CRYPTO_DMA_OUT_DSCR_STATE	CRYPTO_DMA_OUTLINK_DSCR_ADDR	
31 28	27 23	22 20	19 18	17 0	
0 0 0 0	0	0	0	0	Reset

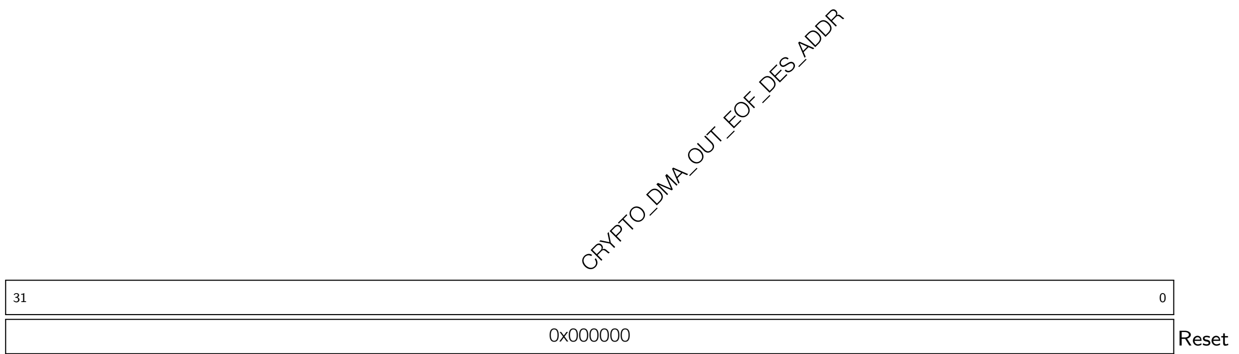
CRYPTO_DMA_OUTLINK_DSCR_ADDR This register stores the current transmit descriptor's address. (RO)

CRYPTO_DMA_OUT_DSCR_STATE Reserved (RO)

CRYPTO_DMA_OUT_STATE Reserved (RO)

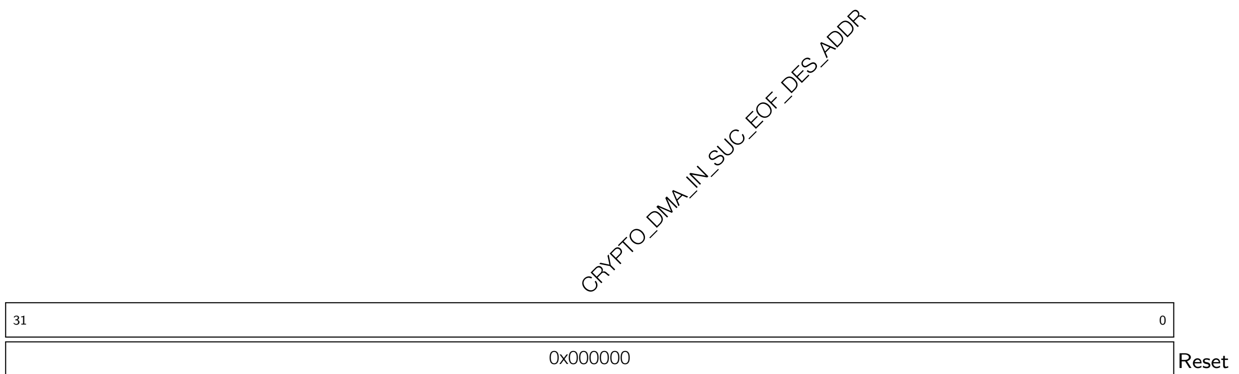
CRYPTO_DMA_OUTFIFO_CNT This register stores the byte number of the data in the transmit descriptor's FIFO. (RO)

Register 2.34: CRYPTO_DMA_OUT_EOF_DES_ADDR_REG (0x0038)



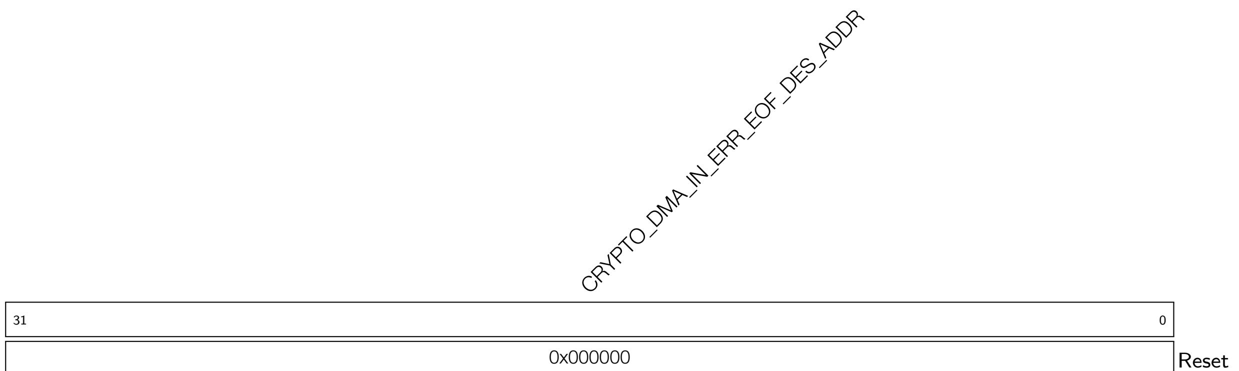
CRYPTO_DMA_OUT_EOF_DES_ADDR This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.35: CRYPTO_DMA_IN_SUC_EOF_DES_ADDR_REG (0x003C)



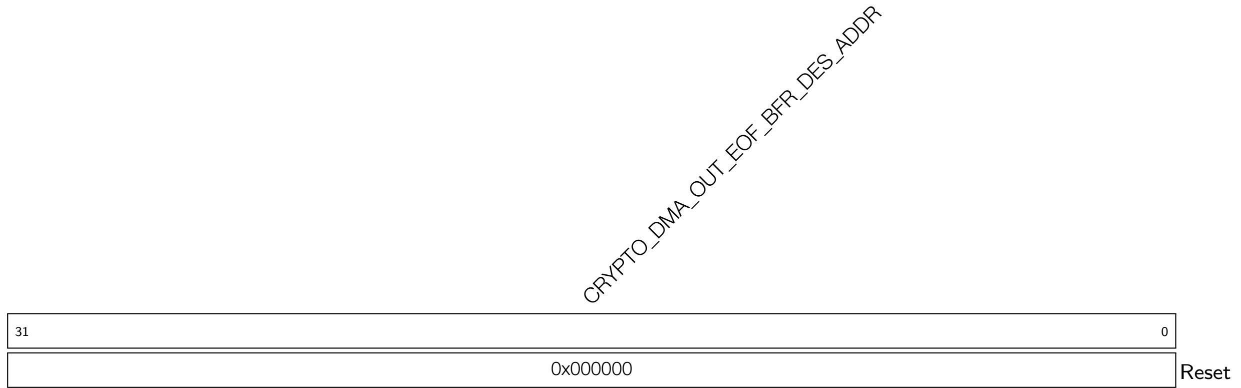
CRYPTO_DMA_IN_SUC_EOF_DES_ADDR This register stores the address of the receive descriptor when received successful EOF. (RO)

Register 2.36: CRYPTO_DMA_IN_ERR_EOF_DES_ADDR_REG (0x0040)



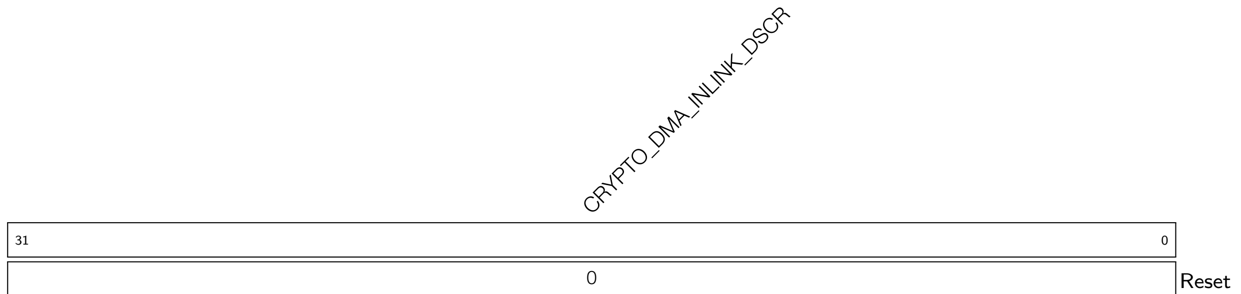
CRYPTO_DMA_IN_ERR_EOF_DES_ADDR This register stores the address of the receive descriptor when there are some errors in this descriptor. (RO)

Register 2.37: CRYPTO_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x0044)



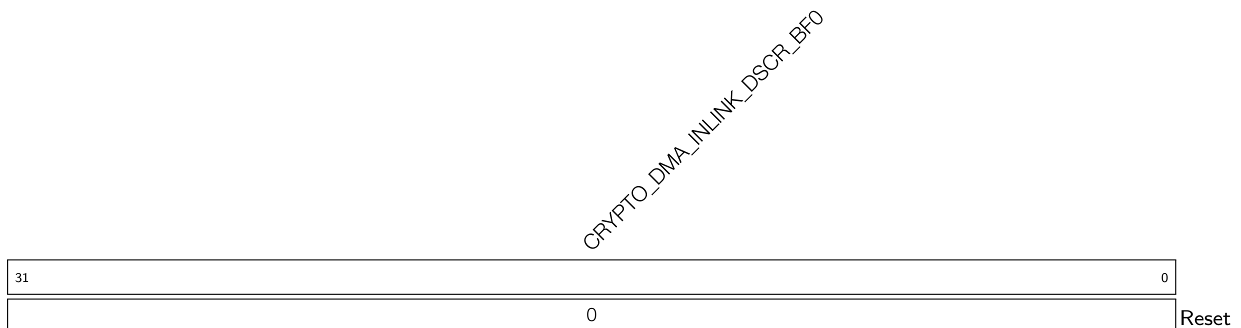
CRYPTO_DMA_OUT_EOF_BFR_DES_ADDR This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 2.38: CRYPTO_DMA_IN_DSCR_REG (0x004C)



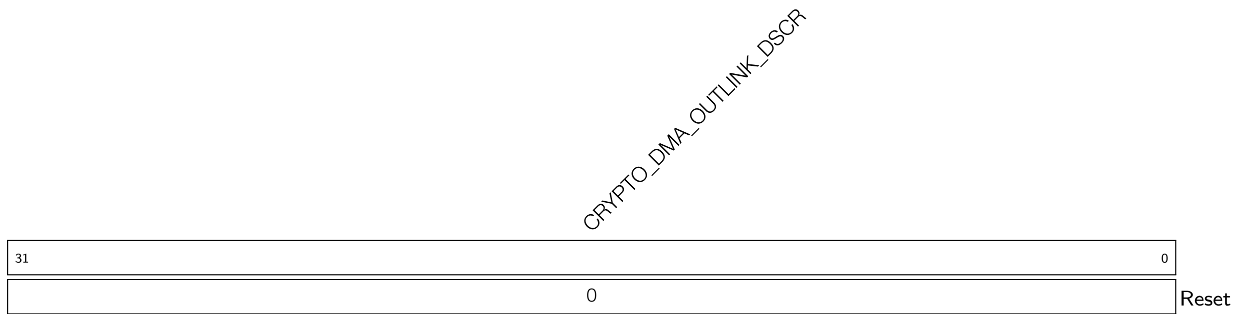
CRYPTO_DMA_INLINK_DSCR The address of the current receive descriptor x. (RO)

Register 2.39: CRYPTO_DMA_IN_DSCR_BF0_REG (0x0050)



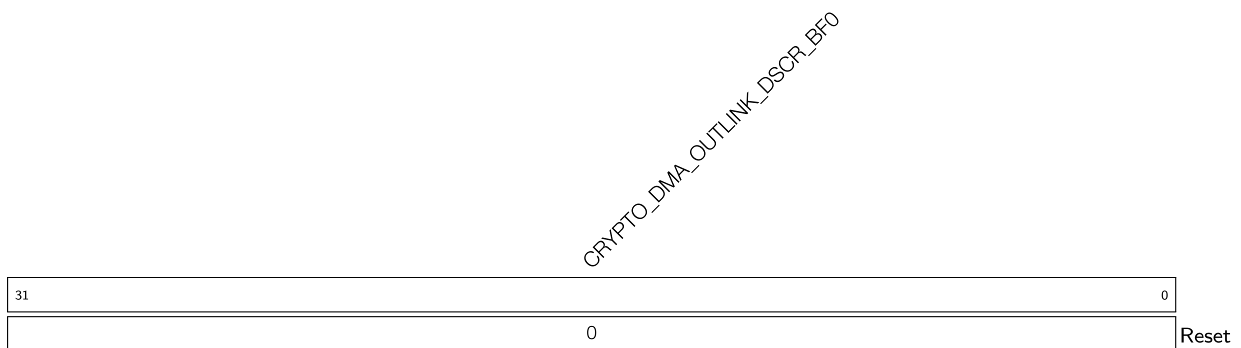
CRYPTO_DMA_INLINK_DSCR_BF0 The address of the last receive descriptor x-1. (RO)

Register 2.40: CRYPTO_DMA_OUT_DSCR_REG (0x0058)



CRYPTO_DMA_OUTLINK_DSCR The address of the current transmit descriptor y. (RO)

Register 2.41: CRYPTO_DMA_OUT_DSCR_BF0_REG (0x005C)



CRYPTO_DMA_OUTLINK_DSCR_BF0 The address of the last transmit descriptor y-1. (RO)

3. System and Memory

3.1 Overview

The ESP32-S2 is a single-core system with one Harvard Architecture Xtensa® LX7 CPU. All internal memory, external memory, and peripherals are located on the CPU buses.

3.2 Features

- **Address Space**
  - 4 GB (32 bits wide) address space in total accessed from the data bus and instruction bus
  - 464 KB internal memory address space accessed from the instruction bus
  - 400 KB internal memory address space accessed from the data bus
  - 1.77 MB peripheral address space
  - 7.5 MB external memory virtual address space accessed from the instruction bus
  - 14.5 MB external memory virtual address space accessed from the data bus
  - 320 KB internal DMA address space
  - 10.5 MB external DMA address space
- **Internal Memory**
  - 128 KB Internal ROM
  - 320 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- **External Memory**
  - Supports up to 1 GB external SPI flash
  - Supports up to 1 GB external SPI RAM
- **DMA**
  - 9 DMA-supported modules / peripherals

Figure 3-1 illustrates the system structure and address mapping.

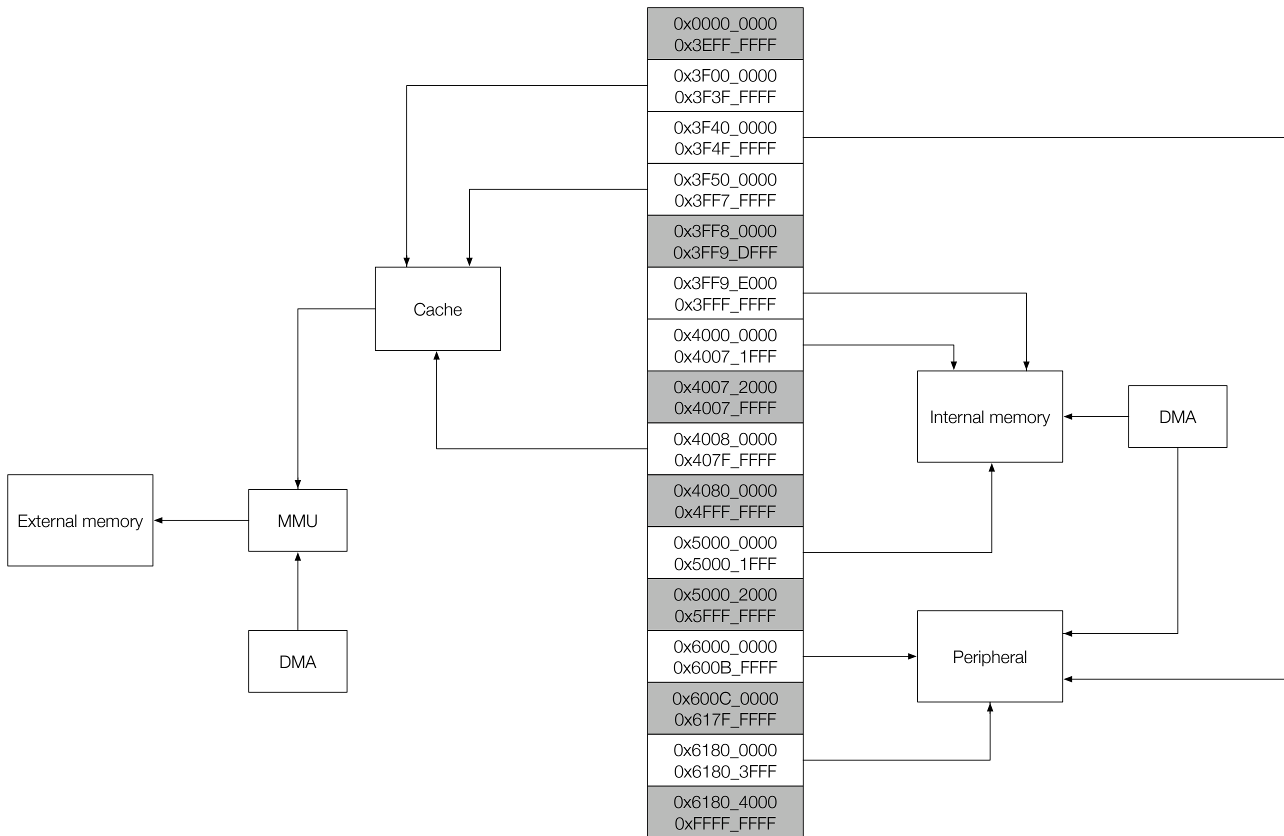


Figure 3-1. System Structure and Address Mapping

Note:

- The memory space with gray background is not available to users.
- The range of addresses available in the address space may be larger or smaller than the actual available memory of a particular type.

3.3 Functional Description

3.3.1 Address Mapping

The Harvard Architecture Xtensa® LX7 CPU can address 4 GB (32 bits wide) memory space.

Addresses below 0x4000_0000 are serviced using the data bus. Addresses in the range 0x4000_0000 ~ 0x4FFF_FFFF are serviced using the instruction bus. Addresses over and including 0x5000_0000 are shared by both data and instruction bus.

Both data bus and instruction bus are little-endian. The CPU can access data via the data bus in a byte-, half-word-, or word-aligned manner. The CPU can also access data via the instruction bus, but only in a word-aligned manner; non-word-aligned access will cause a CPU exception.

The CPU can:

- directly access the internal memory via both data bus and instruction bus;
- access the external memory which is mapped into the address space via cache;
- access modules / peripherals via data bus.

Table 20 lists the address ranges on the data bus and instruction bus and their corresponding target memory.

Some internal and external memory can be accessed via both data bus and instruction bus. In such cases, the same memory is available to the CPU at two address ranges.

Table 20: Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3EFF_FFFF		Reserved
Data bus	0x3F00_0000	0x3F3F_FFFF	4 MB	External memory
	0x3F40_0000	0x3F4F_FFFF	1 MB	Peripherals
	0x3F50_0000	0x3FF7_FFFF	10.5 MB	External memory
	0x3FF8_0000	0x3FF9_DFFF		Reserved
Data bus	0x3FF9_E000	0x3FFF_FFFF	392 KB	Internal memory
Instruction bus	0x4000_0000	0x4007_1FFF	456 KB	Internal memory
	0x4007_2000	0x4007_FFFF		Reserved
Instruction bus	0x4008_0000	0x407F_FFFF	7.5 MB	External memory
	0x4080_0000	0x4FFF_FFFF		Reserved
Data / Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	Internal memory
	0x5000_2000	0x5FFF_FFFF		Reserved
Data / Instruction bus	0x6000_0000	0x600B_FFFF	768 KB	Peripherals
	0x600C_0000	0x617F_FFFF		Reserved
Data / Instruction bus	0x6180_0000	0x6180_3FFF	16 KB	Peripherals
	0x6180_4000	0xFFFF_FFFF		Reserved

3.3.2 Internal Memory

The internal memory consists of four segments: Internal ROM (128 KB), Internal SRAM (320 KB), RTC FAST Memory (8 KB), and RTC SLOW Memory (8 KB).

The Internal ROM is broken down into two parts: Internal ROM 0 (64 KB) and Internal ROM 1 (64 KB).

The Internal SRAM is broken down into two parts: Internal SRAM 0 (32 KB) and Internal SRAM 1 (288 KB).

RTC FAST Memory and RTC SLOW Memory are both implemented as SRAM.

Table 21 lists all types of internal memory and their address ranges on the data bus and instruction bus.

Table 21: Internal Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Data bus	0x3FF9_E000	0x3FF9_FFFF	8 KB	RTC FAST Memory	YES
	0x3FFA_0000	0x3FFA_FFFF	64 KB	Internal ROM 1	NO
	0x3FFB_0000	0x3FFB_7FFF	32 KB	Internal SRAM 0	YES
	0x3FFB_8000	0x3FFF_FFFF	288 KB	Internal SRAM 1	YES

<table>
<tr><th rowspan="2">Bus Type</th><th colspan="2">Boundary Address</th><th rowspan="2">Size</th><th rowspan="2">Target</th><th rowspan="2">Permission Control</th></tr>
<tr><th>Low Address</th><th>High Address</th></tr>
<tr><td rowspan="5">Instruction bus</td><td>0x4000_0000</td><td>0x4000_FFFF</td><td>64 KB</td><td>Internal ROM 0</td><td>NO</td></tr>
<tr><td>0x4001_0000</td><td>0x4001_FFFF</td><td>64 KB</td><td>Internal ROM 1</td><td>NO</td></tr>
<tr><td>0x4002_0000</td><td>0x4002_7FFF</td><td>32 KB</td><td>Internal SRAM 0</td><td>YES</td></tr>
<tr><td>0x4002_8000</td><td>0x4006_FFFF</td><td>288 KB</td><td>Internal SRAM 1</td><td>YES</td></tr>
<tr><td>0x4007_0000</td><td>0x4007_1FFF</td><td>8 KB</td><td>RTC FAST Memory</td><td>YES</td></tr>
<tr><th rowspan="2">Bus Type</th><th colspan="2">Boundary Address</th><th rowspan="2">Size</th><th rowspan="2">Target</th><th rowspan="2">Permission Control</th></tr>
<tr><th>Low Address</th><th>High Address</th></tr>
<tr><td>Data / Instruction bus</td><td>0x5000_0000</td><td>0x5000_1FFF</td><td>8 KB</td><td>RTC SLOW Memory</td><td>YES</td></tr>
</table>

Note:

"YES" in the "Permission Control" column indicates that a permission is required for memory access. Permission Control registers can be used to limit Instruction or Data bus access to individual regions of these memory types.

3.3.2.1 Internal ROM 0

Internal ROM 0 is a 64-KB, read-only memory space, addressed by the CPU on the instruction bus via range(s) described in Table 21.

3.3.2.2 Internal ROM 1

Internal ROM 1 is a 64-KB, read-only memory space, addressed by the CPU on the data or instruction bus via range(s) described in Table 21.

The two address ranges access Internal ROM 1 in the same order, so, for example, addresses 0x3FFA_0000 and 0x4001_0000 access the same word, 0x3FFA_0004 and 0x4001_0004 access the same word, 0x3FFA_0008 and 0x4001_0008 access the same word, etc.

3.3.2.3 Internal SRAM 0

Internal SRAM 0 is a 32-KB, read-and-write memory space, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 21.

Hardware can be configured to use 8 KB, 16 KB, 24 KB, or the entire 32 KB space in this memory to cache external memory. The space used as cache cannot be accessed by the CPU, while the remaining space can still be accessed by the CPU.

3.3.2.4 Internal SRAM 1

Internal SRAM 1 is a 288-KB, read-and-write memory space, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 21.

Internal SRAM 1 comprises eighteen 16-KB (sub)memory blocks. One block can be used as Trace Memory, in which case this block's address range cannot be accessed by the CPU.

3.3.2.5 RTC FAST Memory

RTC FAST Memory is an 8-KB, read-and-write SRAM, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 21.

3.3.2.6 RTC SLOW Memory

RTC SLOW Memory is an 8-KB, read-and-write SRAM, addressed by the CPU via range(s) shared by the data bus and the instruction bus, as described in Table 21.

RTC SLOW Memory can also be used as a peripheral addressable to the CPU via either 0x3F42_1000 ~ 0x3F42_2FFF or 0x6002_1000 ~ 0x6002_2FFF on the data bus.

3.3.3 External Memory

ESP32-S2 supports multiple QSPI/OSPI flash and RAM chips. It also supports hardware encryption/decryption based on XTS-AES to protect user programs and data in the flash and external RAM.

3.3.3.1 External Memory Address Mapping

The CPU accesses the external flash and RAM via the cache. According to the MMU settings, the cache maps the CPU's address to the external physical memory address. Due to this address mapping, the ESP32-S2 can address up to 1 GB external flash and 1 GB external RAM.

Using the cache, ESP32-S2 can support the following address space mappings at the same time.

- Up to 7.5 MB instruction bus address space can be mapped into the external flash or RAM as individual 64 KB blocks, via the instruction cache (ICache). Byte (8-bit), half-word (16-bit) and word (32-bit) reads are supported.
- Up to 4 MB read-only data bus address space can be mapped into the external flash or RAM as individual 64 KB blocks, via ICache. Byte (8-bit), half-word (16-bit) and word (32-bit) reads are supported.
- Up to 10.5 MB data bus address space can be mapped into the external RAM as individual 64 KB blocks, via DCache. Byte (8-bit), half-word (16-bit) or word (32-bit) reads and writes are supported. Blocks from this 10.5 MB space can also be mapped into the external flash or RAM, for read operations only.

Table 22 lists the mapping between the cache and the corresponding address ranges on the data bus and instruction bus.

Table 22: External Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Data bus	0x3F00_0000	0x3F3F_FFFF	4 MB	ICache	YES
Data bus	0x3F50_0000	0x3FF7_FFFF	10.5 MB	DCache	YES
Instruction bus	0x4008_0000	0x407F_FFFF	7.5 MB	ICache	YES

Note:

"YES" in the "Permission Control" column indicates that a permission is required for memory access. Permission Control registers can be used to limit Instruction or Data bus access to individual regions of these memory types.

3.3.3.2 Cache

As shown in Figure 3-2, the caches on ESP32-S2 are separated and allow prompt response upon simultaneous requests from the data bus and instruction bus. Some internal memory space can be used as cache (see Section 3.3.2.3). When a cache miss occurs, the cache controller will initiate a request to the external memory. When ICache and DCache simultaneously initiate a request, the arbiter determines which gets the access to the external memory first. The cache size of ICache and DCache can be configured to 8 KB and 16 KB, respectively, while their block size can be configured to 16 bytes and 32 bytes, respectively.

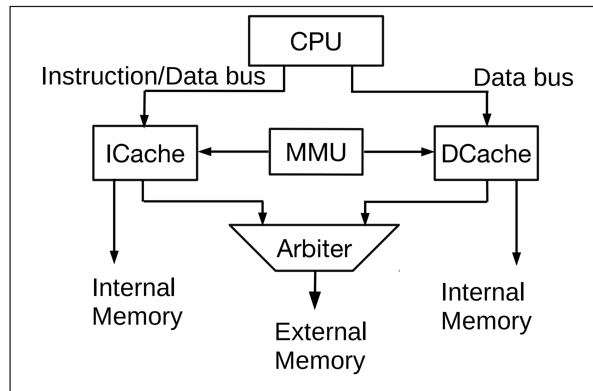


Figure 3-2. Cache Structure

3.3.3.3 Cache Operations

ESP32-S2 caches support the following operations:

1. **Invalidate**: The cache clears the valid bit of a tag. The CPU needs to access the external memory in order to read/write the data. There are two types of invalidation operations: manual invalidation and automatic invalidation. Manual invalidation performs only on data in the specified area in the cache, while automatic invalidation performs on all data in the cache. Both ICache and DCache have this function.

2. **Clean**: The cache clears the dirty bit of the tag and retains the valid bit. The CPU can then read/write the data directly from the cache. Only DCache has this function.

3. **Write-back**: The cache clears the dirty block flag of the tag and retains the valid bit. It also forces the data in the corresponding address to be written back to the external memory. The CPU can then read/write the data directly from the cache. Only DCache has this function.

4. **Preload**: To preload a cache is to load instructions and data into the cache in advance. The minimum unit of a preloading is one block. There are two types of preloading: manual preloading and automatic preloading. Manual preloading means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Automatic preloading means the hardware prefetches a piece of continuous data according to the current hit / miss address (depending on configuration).

5. **Lock / Unlock**: There are two types of lock: prelock and manual lock. When prelock is enabled, the cache locks the data in the specified area when filling the missing data to cache memory. When manual lock is enabled, the cache checks the data that has been filled into the cache memory and locks the data that falls in the specified area. The data in the locked area is always stored in the cache and will not be replaced. But when all the ways within the cache are locked, the cache will replace data, as if the ways were not locked. Unlocking is the reverse of locking. The manual invalidation, clean, and write-back operations are only available after unlocking.

3.3.4 DMA

With DMA, the ESP32-S2 can perform data transfers between:

- modules / peripherals and internal memory;
- different types of internal memory;
- modules / peripherals and external memory;
- internal and external memory.

DMA uses the same addressing as the data bus to read and write Internal SRAM 0 and Internal SRAM 1. Specifically, DMA uses address range 0x3FFB_0000 ~ 0x3FFB_7FFF to access Internal SRAM 0, and 0x3FFB_8000 ~ 0x3FFF_FFFF to access Internal SRAM 1. Note that DMA cannot access the internal memory occupied by the cache.

In addition, DMA addresses the external RAM from 0x3F50_0000 ~ 0x3FF7_FFFF, the same used by the CPU to access DCache. When DCache and DMA access the external memory simultaneously, data consistency is required.

Nine modules / peripherals on the ESP32-S2 support DMA, as shown in Table 23. With DMA, some of them can only access internal memory, some can access both internal and external memory.

For more information on DMA, please refer to Chapter 2: *DMA Controller*.

Table 23: Peripherals with DMA Support

UART0	UART1
SPI2	SPI3
I2S0	
ADC Controller	
Copy DMA	
AES Accelerator	SHA Accelerator

3.3.5 Modules / Peripherals

The CPU can access modules / peripherals via address range 0x3F40_0000 ~ 0x3F4F_FFFF on the data bus, or via 0x6000_0000 ~ 0x600B_FFFF and 0x6180_0000 ~ 0x6180_3FFF shared by the data bus and instruction bus.

3.3.5.1 Naming Conventions for Peripheral Buses

There are two peripheral buses defined as follows:

- **PeriBus1**: Which refers to the address range 0x3F40_0000 ~ 0x3F4F_FFFF on the bus. 0x3F40_0000 is the base address.
- **PeriBus2**: Which refers to the address ranges 0x6000_0000 ~ 0x600B_FFFF and 0x6180_0000 ~ 0x6180_3FFF on the bus. 0x6000_0000 is the base address.

All references to “PeriBus1” and “PeriBus2” in this document indicate the corresponding address range(s).

3.3.5.2 Differences Between PeriBus1 and PeriBus2

The CPU can access modules / peripherals more efficiently through PeriBus1 than through PeriBus2. However, PeriBus1 features speculative reads, which means it cannot guarantee that each read is valid. Therefore, the CPU has to use PeriBus2 to access some special registers, for example, FIFO registers.

In addition, PeriBus1 will upset the order of r/w operations on the bus to improve performance, which may cause programs that have strict requirements on the r/w order to crash. In such cases, please add volatile before the program statement, or use PeriBus2 instead.

3.3.5.3 Module / Peripheral Address Mapping

Table 24 lists all the modules / peripherals and their respective address ranges. Note that addresses in column "Boundary Address" are offsets relative to the base address, instead of absolute addresses. The absolute addresses are the addition of bus base address and the corresponding offsets.

Table 24: Module / Peripheral Address Mapping

Target	Boundary Address		Size	Notes
	Low Address	High Address		
UART0	0x0000_0000	0x0000_0FFF	4 KB	1, 2, 3
Reserved	0x0000_1000	0x0000_1FFF		
SPI1	0x0000_2000	0x0000_2FFF	4 KB	1, 2
SPI0	0x0000_3000	0x0000_3FFF	4 KB	1, 2
GPIO	0x0000_4000	0x0000_4FFF	4 KB	1, 2
Reserved	0x0000_5000	0x0000_6FFF		
TIMER	0x0000_7000	0x0000_7FFF	4 KB	1, 2
RTC	0x0000_8000	0x0000_8FFF	4 KB	1, 2
IO MUX	0x0000_9000	0x0000_9FFF	4 KB	1, 2
Reserved	0x0000_A000	0x0000_EFFF		
I2S0	0x0000_F000	0x0000_FFFF	4 KB	1, 2, 3
UART1	0x0001_0000	0x0001_0FFF	4 KB	1, 2, 3
Reserved	0x0001_1000	0x0001_2FFF		
I2C0	0x0001_3000	0x0001_3FFF	4 KB	1, 2, 3
UHCI0	0x0001_4000	0x0001_4FFF	4 KB	1, 2
Reserved	0x0001_5000	0x0001_5FFF		
RMT	0x0001_6000	0x0001_6FFF	4 KB	1, 2, 3
PCNT	0x0001_7000	0x0001_7FFF	4 KB	1, 2
Reserved	0x0001_8000	0x0001_8FFF		
LED PWM Controller	0x0001_9000	0x0001_9FFF	4 KB	1, 2
eFuse Controller	0x0001_A000	0x0001_AFFF	4 KB	1, 2
Reserved	0x0001_B000	0x0001_EFFF		
Timer Group 0	0x0001_F000	0x0001_FFFF	4 KB	1, 2
Timer Group 1	0x0002_0000	0x0002_0FFF	4 KB	1, 2
RTC SLOW Memory	0x0002_1000	0x0002_2FFF	8 KB	1, 2, 3
System Timer	0x0002_3000	0x0002_3FFF	4 KB	1, 2
SPI2	0x0002_4000	0x0002_4FFF	4 KB	1, 2

Target	Boundary Address		Size	Notes
	Low Address	High Address		
SPI3	0x0002_5000	0x0002_5FFF	4 KB	1, 2
APB Controller	0x0002_6000	0x0002_6FFF	4 KB	1, 2
I2C1	0x0002_7000	0x0002_7FFF	4 KB	1, 2, 3
Reserved	0x0002_8000	0x0002_AFFF		
TWAI Controller	0x0002_B000	0x0002_BFFF	4 KB	1, 2
Reserved	0x0002_C000	0x0003_8FFF		
USB OTG	0x0003_9000	0x0003_9FFF	4 KB	1, 2, 3, 4
AES Accelerator	0x0003_A000	0x0003_AFFF	4 KB	1, 2
SHA Accelerator	0x0003_B000	0x0003_BFFF	4 KB	1, 2
RSA Accelerator	0x0003_C000	0x0003_CFFF	4 KB	1, 2
Digital Signature	0x0003_D000	0x0003_DFFF	4 KB	1, 2
HMAC	0x0003_E000	0x0003_EFFF	4 KB	1, 2
Crypto DMA	0x0003_F000	0x0003_FFFF	4 KB	1, 2
Reserved	0x0004_4000	0x000C_DFFF		
ADC Controller	0x0004_0000	0x0004_0FFF	4 KB	1, 2
Reserved	0x0004_1000	0x0007_FFFF		
USB OTG	0x0008_0000	0x000B_FFFF	256 KB	1, 2, 3, 4
System Registers	0x000C_0000	0x000C_0FFF	4 KB	1
Sensitive Register	0x000C_1000	0x000C_1FFF	4 KB	1
Interrupt Matrix	0x000C_2000	0x000C_2FFF	4 KB	1
Copy DMA	0x000C_3000	0x000C_3FFF	4 KB	1
Reserved	0x000C_4000	0x000C_EFFF		
Dedicated GPIO	0x000C_F000	0x000C_FFFF	4 KB	1
Reserved	0x000D_1000	0x000F_FFFF		
Configure Cache	0x0180_0000	0x0180_3FFF	16 KB	2

Note:

1. This module / peripheral can be accessed from PeriBus1.
2. This module / peripheral can be accessed from PeriBus2.
3. Some special addresses in this module / peripheral are not accessible from PeriBus1 (see Section 3.3.5.4).
4. The address space in this module / peripheral is not continuous.

3.3.5.4 Addresses with Restricted Access from PeriBus1

As mentioned in Section 3.3.5.2, PeriBus1 features speculative reads, which means it is forbidden to read FIFO registers. Table 25 below lists the address (range) with restricted access from PeriBus1.

There are four reserved user-defined registers that can be configured as needed to add more addresses with restricted access. For more information, please refer to Chapter 14: *Permission Control*.

Table 25: Addresses with Restricted Access

Peripherals	Addresses with Restricted Access
UART0	0x3F40_0000
UART1	0x3F41_0000
I2S0	0x3F40_F004
RMT	0x3F41_6000 ~ 0x3F41_600F
I2C0	0x3F41_301C
I2C1	0x3F42_701C
USB OTG	0x3F48_0020, 0x3F48_1000 ~ 0x3F49_0FFF

4. eFuse Controller (eFuse)

4.1 Overview

ESP32-S2 has a 4096-bit eFuse that stores parameters in the SoC. Once an eFuse bit is programmed to 1, it can never be reverted to 0. Software can instruct the eFuse Controller to program individual bits for individual parameters as needed. Some of these parameters can be read by software using the eFuse Controller registers, while some can be directly used by hardware modules.

4.2 Features

- One-time programmable storage
- Configurable write protection
- Configurable software read protection
- Parameters use different hardware encoding schemes to protect against corruption

4.3 Functional Description

4.3.1 Structure

There are 11 eFuse blocks (BLOCK0 ~ BLOCK10).

BLOCK0 holds most core system parameters. Among these parameters, 24 bits are invisible to software and can only be used by hardware and 38 bits are reserved for future use.

Table 26 lists all the parameters in BLOCK0 and their offsets, bit widths, functional description, as well as information on whether they can be used by hardware, and whether they are protected from programming.

The **EFUSE_WR_DIS** parameter is used to restrict the programming of other parameters, while **EFUSE_RD_DIS** is used to restrict software from reading BLOCK4 ~ BLOCK10. More information on these two parameters can be found in sections 4.3.1.1 and 4.3.1.2.

Table 26: Parameters in BLOCK0

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_WR_DIS	0	32	Y	N/A	Disables programming of individual eFuses
EFUSE_RD_DIS	32	7	Y	0	Disables software reading from individual eFuse blocks BLOCK4 ~ 10
EFUSE_DIS_ICACHE	40	1	Y	2	Disables ICache
EFUSE_DIS_DCACHE	41	1	Y	2	Disables DCache
EFUSE_DIS_DOWNLOAD_ICACHE	42	1	Y	2	Disables Icache when SoC is in Download mode
EFUSE_DIS_DOWNLOAD_DCACHE	43	1	Y	2	Disables Dcache when SoC is in Download mode

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_DIS_FORCE_DOWNLOAD	44	1	Y	2	Disables forcing chip into Download mode
EFUSE_DIS_USB	45	1	Y	2	Disables the USB OTG hardware
EFUSE_DIS_CAN	46	1	Y	2	Disables the TWAI Controller hardware
EFUSE_DIS_BOOT_REMAP	47	1	Y	2	Disables capability to Remap RAM to ROM address space
EFUSE_SOFT_DIS_JTAG	49	1	Y	2	Software disables JTAG. When software disabled, JTAG can be activated temporarily by HMAC peripheral.
EFUSE_HARD_DIS_JTAG	50	1	Y	2	Hardware disables JTAG permanently.
EFUSE_DIS_DOWNLOAD_ MANUAL_ENCRYPT	51	1	Y	2	Disables flash encryption when in download boot modes
EFUSE_USB_EXCHG_PINS	56	1	Y	30	Exchanges USB D+ and D- pins
EFUSE_EXT_PHY_ENABLE	57	1	N	30	Enables external USB PHY
EFUSE_USB_FORCE_NOPERSIST	58	1	N	30	Forces to set USB BVALID to 1
EFUSE_VDD_SPI_XPD	68	1	Y	3	If VDD_SPI_FORCE is 1, determines if the VDD_SPI regulator is powered on
EFUSE_VDD_SPI_TIEH	69	1	Y	3	If VDD_SPI_FORCE is 1, determines VDD_SPI voltage. 0: VDD_SPI connects to 1.8 V LDO; 1: VDD_SPI connects to VDD_RTC_IO
EFUSE_VDD_SPI_FORCE	70	1	Y	3	When set, XPD_VDD_PSI_REG and VDD_SPI_TIEH will be used to configure VDD_SPI LDO
EFUSE_WDT_DELAY_SEL	80	2	Y	3	Selects RTC WDT timeout threshold at startup
EFUSE_SPI_BOOT_CRYPT_CNT	82	3	Y	4	Enables encryption and decryption, when an SPI boot mode is set. Feature is enabled when 1 or 3 bits are set in the eFuse, disabled otherwise.
EFUSE_SECURE_BOOT_KEY_ REVOKE0	85	1	N	5	If set, revokes use of secure boot key digest 0
EFUSE_SECURE_BOOT_KEY_ REVOKE1	86	1	N	6	If set, revokes use of secure boot key digest 1
EFUSE_SECURE_BOOT_KEY_ REVOKE2	87	1	N	7	If set, revokes use of secure boot key digest 2

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_KEY_PURPOSE_0	88	4	Y	8	KEY0 purpose, see Table 27
EFUSE_KEY_PURPOSE_1	92	4	Y	9	KEY1 purpose, see Table 27
EFUSE_KEY_PURPOSE_2	96	4	Y	10	KEY2 purpose, see Table 27
EFUSE_KEY_PURPOSE_3	100	4	Y	11	KEY3 purpose, see Table 27
EFUSE_KEY_PURPOSE_4	104	4	Y	12	KEY4 purpose, see Table 27
EFUSE_KEY_PURPOSE_5	108	4	Y	13	KEY5 purpose, see Table 27
EFUSE_SECURE_BOOT_EN	116	1	N	15	Enables secure boot
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	117	1	N	16	Enables aggressive secure boot key revocation mode
EFUSE_FLASH_TPUW	124	4	N	18	Configures flash startup delay after SoC power-up, unit is (ms/2). When the value is 15, delay is 7.5 ms.
EFUSE_DIS_DOWNLOAD_MODE	128	1	N	18	Disables all Download boot modes
EFUSE_DIS_LEGACY_SPI_BOOT	129	1	N	18	Disables Legacy SPI boot mode
EFUSE_UART_PRINT_CHANNEL	130	1	N	18	Selects the default UART for printing boot messages, 0: UART0; 1: UART1
EFUSE_DIS_USB_DOWNLOAD_MODE	132	1	N	18	Disables use of USB in UART download boot mode
EFUSE_ENABLE_SECURITY_DOWNLOAD	133	1	N	18	Enables secure UART download mode (read/write flash only)
EFUSE_UART_PRINT_CONTROL	134	2	N	18	Sets the default UART boot message output mode. 2'b00: Enabled; 2'b01: Enable when GPIO 46 is low at reset; 2'b10: Enable when GPIO 46 is high at rest; 2'b11: Disabled.
EFUSE_PIN_POWER_SELECTION	136	1	N	18	Sets default power supply for GPIO33 ~ GPIO37, set when SPI flash is initialized. 0: VDD3P3_CPU; 1: VDD_SPI
EFUSE_FLASH_TYPE	137	1	N	18	Selects SPI flash type, 0: maximum four data lines; 1: eight data lines
EFUSE_FORCE_SEND_RESUME	138	1	N	18	Forces ROM code to send an SPI flash resume command during SPI boot
EFUSE_SECURE_VERSION	139	16	N	18	Secure version (used by ESP-IDF anti-rollback feature)

Table 27 lists all key purpose and their values. Setting the eFuse parameter EFUSE_KEY_PURPOSE_*n* programs

the purpose for eFuse block KEY*n* (*n*: 0 ~ 5).

Table 27: Key Purpose Values

Key Purpose Values	Purposes
0	User purposes (software-only use)
1	Reserved
2	XTS_AES_256_KEY_1 (flash/PSRAM encryption)
3	XTS_AES_256_KEY_2 (flash/PSRAM encryption)
4	XTS_AES_128_KEY (flash/PSRAM encryption)
5	HMAC Downstream mode
6	JTAG soft enable key (uses HMAC Downstream mode)
7	Digital Signature peripheral key (uses HMAC Downstream mode)
8	HMAC Upstream mode
9	SECURE_BOOT_DIGEST0 (Secure Boot key digest)
10	SECURE_BOOT_DIGEST1 (Secure Boot key digest)
11	SECURE_BOOT_DIGEST2 (Secure Boot key digest)

Table 28 provides the details on the parameters in BLOCK1 ~ BLOCK10.

Table 28: Parameters in BLOCK1-10

BLOCK	Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Software Read Protection by EFUSE_RD_DIS Bit Number	Description
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC address
	EFUSE_SPI_PAD_CONFIGURE	[0:5]	N	20	N/A	CLK
		[6:11]	N	20	N/A	Q (D1)
		[12:17]	N	20	N/A	D (D0)
		[18:23]	N	20	N/A	CS
		[24:29]	N	20	N/A	HD (D3)
		[30:35]	N	20	N/A	WP (D2)
		[36:41]	N	20	N/A	DQS
		[42:47]	N	20	N/A	D4
		[48:53]	N	20	N/A	D5
		[54:59]	N	20	N/A	D6
		[60:65]	N	20	N/A	D7
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	System data
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	System data
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	User data
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	Key0 or user data
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	Key1 or user data
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	Key2 or user data

BLOCK	Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Software Read Protection by EFUSE_RD_DIS Bit Number	Description
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	Key3 or user data
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	Key4 or user data
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	Key5 or user data
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	System data

Among these blocks, BLOCK4 ~ 9 stores KEY0 ~ 5, respectively. Up to six 256-bit keys can be programmed into eFuse. Whenever a key is programmed, its purpose value should also be programmed (see table 27). For example, a key for the JTAG function in HMAC Downstream mode is programmed to KEY3 (i.e., BLOCK7), then, the key purpose value 6 should be programmed to EFUSE_KEY_PURPOSE_3.

BLOCK1 ~ BLOCK10 use the RS coding scheme, so there are some restrictions on writing to these parameters (refer to Section 4.3.1.3: *Data Storage* and Section 4.3.2: *Software Programming of Parameters*).

4.3.1.1 EFUSE_WR_DIS

Parameter EFUSE_WR_DIS determines whether individual eFuse parameters are write-protected. After burning EFUSE_WR_DIS, execute an eFuse read operation so the new values will take effect (refer to *Updating eFuse read registers* in Section 4.3.3).

The columns "Programming-Protected by EFUSE_WR_DIS" in Table 26 and Table 28 list the specific bits of EFUSE_WR_DIS that determine the write protected status of each parameter.

When the corresponding bit is 0, the parameter is not write protected and can be programmed if the parameter has not been programmed.

When the corresponding bit is 1, the parameter is write protected and none of its bits can be modified. The non-programmed bits always remain 0, while programmed bits always remain 1.

4.3.1.2 EFUSE_RD_DIS

Only the eFuse blocks BLCOK4 ~ BLOCK10 can be individually software read protected. The corresponding bit in EFUSE_RD_DIS is shown in Table 28. After burning EFUSE_RD_DIS, execute an eFuse read operation so the new values will take effect (refer to *Updating eFuse read registers* in Section 4.3.3).

If the corresponding EFUSE_RD_DIS bit is 0, then the eFuse block can be read by software. If the corresponding EFUSE_RD_DIS bit is 1, then the parameter controlled by this bit is software read protected.

Other parameters that are not in BLOCK4 ~ BLOCK10 can always be read by software.

Although BLOCK4 ~ BLOCK10 can be set to read-protected, they can still be used by hardware modules, if the EFUSE_KEY_PURPOSE_*n* bit is set accordingly.

4.3.1.3 Data Storage

Internal to the SoC, eFuses use hardware encoding schemes to protect against data corruption.

All BLOCK0 parameters except for EFUSE_WR_DIS are stored with four backups, meaning each bit is stored four times. This backup scheme is automatically applied by the hardware and is not visible to software.

BLOCK1 ~ BLOCK10 use RS (44, 32) coding scheme that supports up to 5 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. The shift register circuit that generates the check code is shown in Figure 4-1, where gf_mul_n (n is an integer) is the result of multiplying a byte of data in the $GF(2^8)$ field by the element α^n.

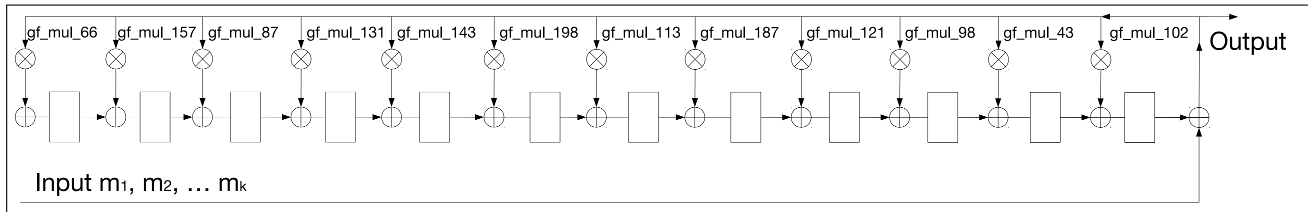


Figure 4-1. Shift Register Circuit

Software must encode the 32-byte parameter using RS (44, 32) to generate a 12-byte check code, and then burn the parameter and the check code into eFuse at the same time. The eFuse Controller automatically decodes the RS encoding and applies error correction when reading back the eFuse block.

Because the RS check codes are generated across the entire 256-bit eFuse block, each block can only be written to one time.

4.3.2 Software Programming of Parameters

The eFuse controller can only write to eFuse parameters in one block at a time. BLOCK0 ~ BLOCK10 share the same registers to store the parameters to be programmed. Configure parameter EFUSE_BLK_NUM to indicate which block is to be programmed.

Programming BLOCK0

When EFUSE_BLK_NUM = 0, BLOCK0 is programmed. Register EFUSE_PGM_DATA0_REG stores EFUSE_WR_DIS. EFUSE_PGM_DATA1_REG ~ EFUSE_PGM_DATA5_REG store the information of new BLOCK0 parameters to be programmed. Note that 24 BLOCK0 bits are reserved and can only be used by hardware. These must always be set to 0 in the programming registers. The specific bits are:

- EFUSE_PGM_DATA1_REG[29:31]
- EFUSE_PGM_DATA1_REG[20:23]
- EFUSE_PGM_DATA2_REG[7:15]
- EFUSE_PGM_DATA2_REG[0:3]
- EFUSE_PGM_DATA3_REG[16:19]

Values written to EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG are ignored when programming BLOCK0.

Programming BLOCK1

When EFUSE_BLK_NUM = 1, EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG store the parameters to be programmed. EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_DATA2_REG store the corresponding

RS check codes. Values written to EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG are ignored when programming BLOCK1, the RS check codes should be calculated as if these bits were all 0.

Programming BLOCK2 ~ 10

When EFUSE_BLK_NUM = 2 ~ 10, EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG store the parameters to be programmed to this block. EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG store the corresponding RS check codes.

Programming process

The process of programming parameters is as follows:

1. Set EFUSE_BLK_NUM parameter as described above.
2. Write the parameters to be programmed into registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
3. Ensure the eFuse clock registers are set correctly as described in Section 4.3.4.1: *eFuse-Programming Timing*.
4. Ensure the eFuse programming voltage VDDQ is set correctly as described in Section 4.3.4.2: *eFuse VDDQ Setting*.
5. Set EFUSE_OP_CODE field in register EFUSE_CONF_REG to 0x5A5A.
6. Set EFUSE_PGM_CMD field in register EFUSE_CMD_REG to 1.
7. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a pgm_done interrupt. Information on how to identify a pgm/read_done interrupt is provided at the end of Section 4.3.3.
8. Clear the parameters written into the register.
9. Trigger an eFuse read operation (see Section 4.3.3: *Software Reading of Parameters*) to update eFuse registers with the new values.

Limitations

For BLOCK0, the programming of different parameters and even the programming of different bits of the same parameter does not need to be done at once. It is, however, recommended that users minimize programming cycles and program all the bits of a parameter in one programming action. In addition, after all parameters controlled by a certain bit of EFUSE_WR_DIS are programmed, that bit should be immediately programmed. The programming of parameters controlled by a certain bit of EFUSE_WR_DIS, and the programming of the bit itself can even be completed at the same time. Repeated programming of already programmed bits is strictly forbidden, otherwise, programming errors will occur.

BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.

BLOCK2 ~ 10 can only be programmed once. Repeated programming is not allowed.

4.3.3 Software Reading of Parameters

Software cannot read eFuse bits directly. The eFuse Controller hardware reads all eFuse bits and stores the results to their corresponding registers in the memory space. Then, software can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in the table below.

Table 29: Registers for Software Reading Parameters

BLOCK	Read Registers	When Programming This Block
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SPI_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_DATA_PART1_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_DATA_PART2_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating eFuse read registers

The eFuse Controller hardware populates the read registers from the internal eFuse storage. This read operation happens on system reset and can also be triggered manually by software as needed, for example if new eFuse values have been programmed.

The process of triggering an eFuse controller read from software is as follows:

1. Configure the eFuse read timing registers as described in Section 4.3.4.3: *eFuse-Read Timing*.
2. Set the EFUSE_OP_CODE field in register EFUSE_CONF_REG to 0x5AA5.
3. Set the EFUSE_READ_CMD field in register EFUSE_CMD_REG to 1.
4. Poll the EFUSE_CMD_REG register until it is 0x0, or wait for a read_done interrupt. Information on how to identify a pgm/read_done interrupt is provided below.
5. Software reads the values of each parameter from memory.

The eFuse read registers will now hold updated values for all eFuse parameters.

Error detection

Error registers allow software to detect an inconsistency in the stored eFuses.

EFUSE_RD_REPEAT_ERR0 ~ 3_REG indicate inconsistencies in the stored backup copies of the parameters in BLOCK0 (except for EFUSE_WR_DIS) . Value 1 indicates an error was detected, and the bit became invalid. Value 0 indicates no error.

Registers EFUSE_RD_RS_ERR0 ~ 1_REG store the number of corrected bytes as well as the result of RS decoding during eFuse reading BLOCK1 ~ BLOCK10.

Software can read the values of above registers only after the eFuse read registers have been updated.

Identifying program/read operation completion

The two methods to identify the completion of program/read operation are described below. Please note that bit 1 corresponds to program operation, and bit 0 corresponds to read operation.

- Method one:
  1. Poll bit 1/0 in register EFUSE_INT_RAW_REG until it is 1, which represents the completion of a program/read operation.
- Method two:

1. Set bit 1/0 in register EFUSE_INT_ENA_REG to 1 to enable eFuse Controller to post a pgm/read_done interrupt.
2. Configure Interrupt Matrix to enable the CPU to respond to eFuse interrupt signal.
3. Wait for the pgm/read_done interrupt.
4. Set the bit 1/0 in register EFUSE_INT_CLR_REG to 1 to clear the pgm/read_done interrupt.

4.3.4 Timing

4.3.4.1 eFuse-Programming Timing

Figure 4-2 shows the timing for programming eFuse. Four registers EFUSE_TSUP_A, EFUSE_TPGM, EFUSE_THP_A, and EFUSE_TPGM_INACTIVE are used to configure the timing. Terms used in the timing diagrams in this section are described as follows:

- CSB: Chip select, active low
- VDDQ: eFuse programming voltage
- PGENB: eFuse programming enable signal, active low

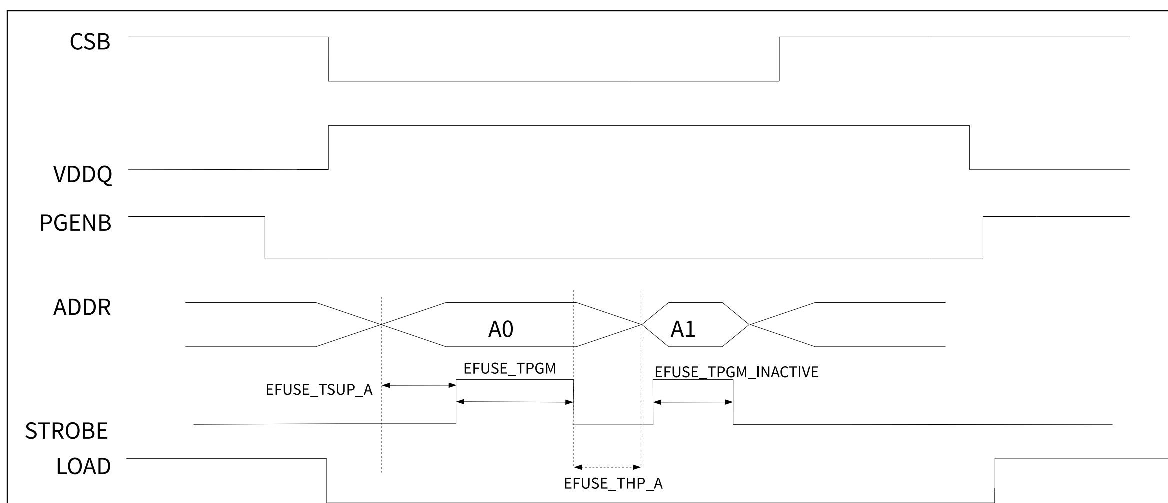


Figure 4-2. eFuse-Programming Timing Diagram

The eFuse block uses the CLK_APB clock, which is configurable. Therefore, the timing parameters should be configured according to the specific clock frequency. After reset, the initial parameters are based on 20 MHz clock frequency.

Table 30: Configuration of eFuse-Programming Timing Parameters

APB Frequency	EFUSE_TSUP_A (> 6.669 ns)	EFUSE_TPGM (9-11 μs, usually 10 μs)	EFUSE_THP_A (> 6.166 ns)	EFUSE_TPGM_INACTIVE (> 35.96 ns)
80 MHz	0x2	0x320	0x2	0x4
40 MHz	0x1	0x190	0x1	0x2
20 MHz	0x1	0xC8	0x1	0x1

In Figure 4-2, Address A0 is programmed, then the corresponding eFuse bit is 1; Address A1 is not programmed, then the corresponding eFuse bit is 0.

4.3.4.2 eFuse VDDQ Timing Setting

VDDQ is the eFuse programming voltage, and its timing parameters should be configured according to the APB clock frequency:

Table 31: Configuration of VDDQ Timing Parameters

APB Frequency	EFUSE_DAC_CLK_DIV (> 1 μs)	EFUSE_PWR_ON_NUM (> EFUSE_DAC_CLK_DIV*255)	EFUSE_PWR_OFF_NUM (> 3 μs)
80 MHz	0xA0	0xA200	0x100
40 MHz	0x50	0x5100	0x80
20 MHz	0x28	0x2880	0x40

4.3.4.3 eFuse-Read Timing

Figure 4-3 shows the timing for reading eFuse. Three registers EFUSE_TSUR_A, EFUSE_TRD, and EFUSE_THR_A are used to configure the timing.

The parameters should be configured according to the specific APB clock frequency. Details can be found in the table below.

Table 32: Configuration of eFuse-Reading Parameters

APB Frequency	EFUSE_TSUR_A (> 6.669 ns)	EFUSE_TRD (> 35.96 ns)	EFUSE_THR_A (> 6.166 ns)
80 MHz	0x2	0x4	0x2
40 MHz	0x1	0x2	0x1
20 MHz	0x1	0x1	0x1

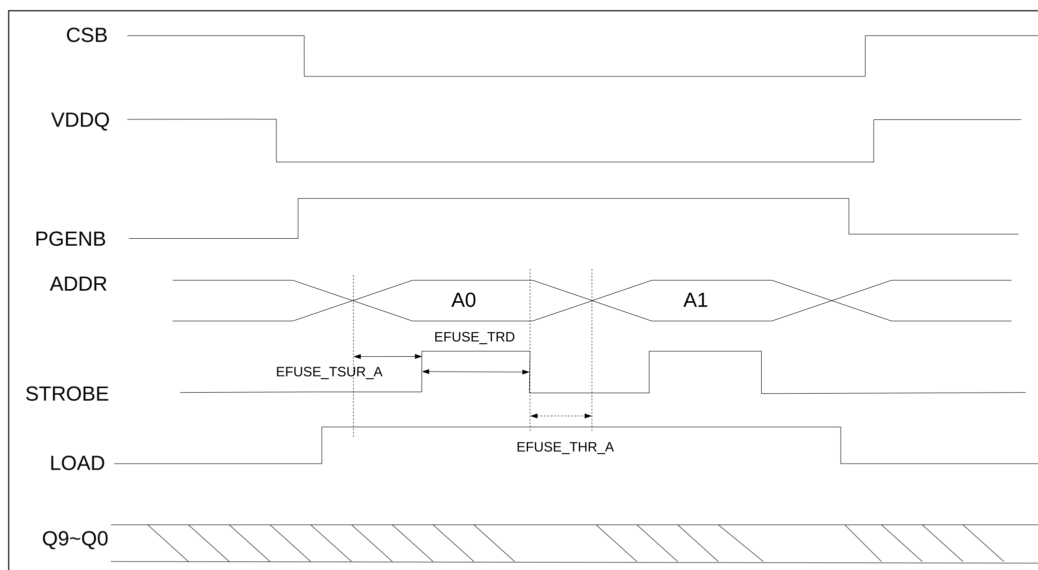


Figure 4-3. Timing Diagram for Reading eFuse

4.3.5 The Use of Parameters by Hardware Modules

Hardware modules are directly hardwired to the ESP32-S2 in order to use the parameters listed in Table 26 and 28, specifically those marked with "Y" in columns "Hardware Use". Software cannot change this behavior.

4.3.6 Interrupts

- pgm_done interrupt: Triggered when eFuse programming has finished. To enable this interrupt, set EFUSE_PGM_DONE_INT_ENA to 1.
- read_done interrupt: Triggered when eFuse reading has finished. To enable this interrupt, set EFUSE_READ_DONE_INT_ENA to 1.

4.4 Base Address

Users can access the eFuse Controller with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 33: eFuse Controller Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x6001A000
PeriBUS2	0x3FC1A000

4.5 Register Summary

The addresses in the following table are relative to the eFuse base addresses provided in Section 4.4.

Name	Description	Address	Access
PGM Data Registers			
EFUSE_PGM_DATA0_REG	Register 0 that stores data to be programmed.	0x0000	R/W
EFUSE_PGM_DATA1_REG	Register 1 that stores data to be programmed.	0x0004	R/W
EFUSE_PGM_DATA2_REG	Register 2 that stores data to be programmed.	0x0008	R/W

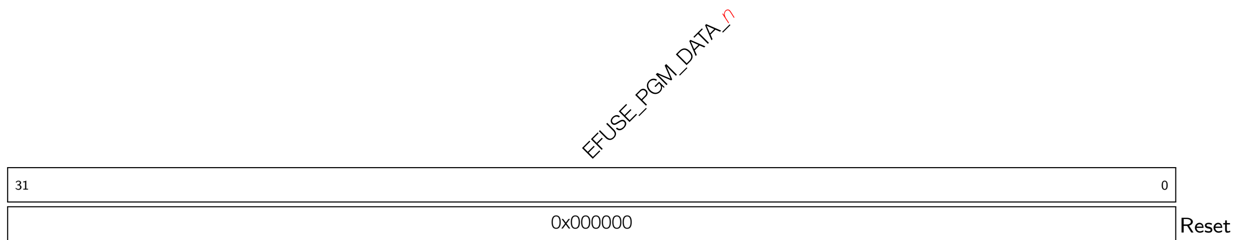
Name	Description	Address	Access
EFUSE_PGM_DATA3_REG	Register 3 that stores data to be programmed.	0x000C	R/W
EFUSE_PGM_DATA4_REG	Register 4 that stores data to be programmed.	0x0010	R/W
EFUSE_PGM_DATA5_REG	Register 5 that stores data to be programmed.	0x0014	R/W
EFUSE_PGM_DATA6_REG	Register 6 that stores data to be programmed.	0x0018	R/W
EFUSE_PGM_DATA7_REG	Register 7 that stores data to be programmed.	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Register 0 that stores the RS code to be programmed.	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Register 1 that stores the RS code to be programmed.	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Register 2 that stores the RS code to be programmed.	0x0028	R/W
Read Data Registers			
EFUSE_RD_WR_DIS_REG	Register 0 of BLOCK0.	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	Register 1 of BLOCK0.	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	Register 2 of BLOCK0.	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	Register 3 of BLOCK0.	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	Register 4 of BLOCK0.	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	Register 5 of BLOCK0.	0x0040	RO
EFUSE_RD_MAC_SPI_SYS_0_REG	Register 0 of BLOCK1.	0x0044	RO
EFUSE_RD_MAC_SPI_SYS_1_REG	Register 1 of BLOCK1.	0x0048	RO
EFUSE_RD_MAC_SPI_SYS_2_REG	Register 2 of BLOCK1.	0x004C	RO
EFUSE_RD_MAC_SPI_SYS_3_REG	Register 3 of BLOCK1.	0x0050	RO
EFUSE_RD_MAC_SPI_SYS_4_REG	Register 4 of BLOCK1.	0x0054	RO
EFUSE_RD_MAC_SPI_SYS_5_REG	Register 5 of BLOCK1.	0x0058	RO
EFUSE_RD_SYS_DATA_PART1_0_REG	Register 0 of BLOCK2 (system).	0x005C	RO
EFUSE_RD_SYS_DATA_PART1_1_REG	Register 1 of BLOCK2 (system).	0x0060	RO
EFUSE_RD_SYS_DATA_PART1_2_REG	Register 2 of BLOCK2 (system).	0x0064	RO
EFUSE_RD_SYS_DATA_PART1_3_REG	Register 3 of BLOCK2 (system).	0x0068	RO
EFUSE_RD_SYS_DATA_PART1_4_REG	Register 4 of BLOCK2 (system).	0x006C	RO
EFUSE_RD_SYS_DATA_PART1_5_REG	Register 5 of BLOCK2 (system).	0x0070	RO
EFUSE_RD_SYS_DATA_PART1_6_REG	Register 6 of BLOCK2 (system).	0x0074	RO
EFUSE_RD_SYS_DATA_PART1_7_REG	Register 7 of BLOCK2 (system).	0x0078	RO
EFUSE_RD_USR_DATA0_REG	Register 0 of BLOCK3 (user).	0x007C	RO
EFUSE_RD_USR_DATA1_REG	Register 1 of BLOCK3 (user).	0x0080	RO
EFUSE_RD_USR_DATA2_REG	Register 2 of BLOCK3 (user).	0x0084	RO
EFUSE_RD_USR_DATA3_REG	Register 3 of BLOCK3 (user).	0x0088	RO
EFUSE_RD_USR_DATA4_REG	Register 4 of BLOCK3 (user).	0x008C	RO
EFUSE_RD_USR_DATA5_REG	Register 5 of BLOCK3 (user).	0x0090	RO
EFUSE_RD_USR_DATA6_REG	Register 6 of BLOCK3 (user).	0x0094	RO
EFUSE_RD_USR_DATA7_REG	Register 7 of BLOCK3 (user).	0x0098	RO
EFUSE_RD_KEY0_DATA0_REG	Register 0 of BLOCK4 (KEY0).	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	Register 1 of BLOCK4 (KEY0).	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	Register 2 of BLOCK4 (KEY0).	0x00A4	RO

Name	Description	Address	Access
EFUSE_RD_KEY0_DATA3_REG	Register 3 of BLOCK4 (KEY0).	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	Register 4 of BLOCK4 (KEY0).	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	Register 5 of BLOCK4 (KEY0).	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	Register 6 of BLOCK4 (KEY0).	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	Register 7 of BLOCK4 (KEY0).	0x00B8	RO
EFUSE_RD_KEY1_DATA0_REG	Register 0 of BLOCK5 (KEY1).	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	Register 1 of BLOCK5 (KEY1).	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	Register 2 of BLOCK5 (KEY1).	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	Register 3 of BLOCK5 (KEY1).	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	Register 4 of BLOCK5 (KEY1).	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	Register 5 of BLOCK5 (KEY1).	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	Register 6 of BLOCK5 (KEY1).	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	Register 7 of BLOCK5 (KEY1).	0x00D8	RO
EFUSE_RD_KEY2_DATA0_REG	Register 0 of BLOCK6 (KEY2).	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	Register 1 of BLOCK6 (KEY2).	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	Register 2 of BLOCK6 (KEY2).	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	Register 3 of BLOCK6 (KEY2).	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	Register 4 of BLOCK6 (KEY2).	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	Register 5 of BLOCK6 (KEY2).	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	Register 6 of BLOCK6 (KEY2).	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	Register 7 of BLOCK6 (KEY2).	0x00F8	RO
EFUSE_RD_KEY3_DATA0_REG	Register 0 of BLOCK7 (KEY3).	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	Register 1 of BLOCK7 (KEY3).	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	Register 2 of BLOCK7 (KEY3).	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	Register 3 of BLOCK7 (KEY3).	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	Register 4 of BLOCK7 (KEY3).	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	Register 5 of BLOCK7 (KEY3).	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	Register 6 of BLOCK7 (KEY3).	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	Register 7 of BLOCK7 (KEY3).	0x0118	RO
EFUSE_RD_KEY4_DATA0_REG	Register 0 of BLOCK8 (KEY4).	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	Register 1 of BLOCK8 (KEY4).	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	Register 2 of BLOCK8 (KEY4).	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	Register 3 of BLOCK8 (KEY4).	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	Register 4 of BLOCK8 (KEY4).	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	Register 5 of BLOCK8 (KEY4).	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Register 6 of BLOCK8 (KEY4).	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Register 7 of BLOCK8 (KEY4).	0x0138	RO
EFUSE_RD_KEY5_DATA0_REG	Register 0 of BLOCK9 (KEY5).	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	Register 1 of BLOCK9 (KEY5).	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	Register 2 of BLOCK9 (KEY5).	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	Register 3 of BLOCK9 (KEY5).	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	Register 4 of BLOCK9 (KEY5).	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	Register 5 of BLOCK9 (KEY5).	0x0150	RO

Name	Description	Address	Access
EFUSE_RD_KEY5_DATA6_REG	Register 6 of BLOCK9 (KEY5).	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	Register 7 of BLOCK9 (KEY5).	0x0158	RO
EFUSE_RD_SYS_DATA_PART2_0_REG	Register 0 of BLOCK10 (system).	0x015C	RO
EFUSE_RD_SYS_DATA_PART2_1_REG	Register 1 of BLOCK10 (system).	0x0160	RO
EFUSE_RD_SYS_DATA_PART2_2_REG	Register 2 of BLOCK10 (system).	0x0164	RO
EFUSE_RD_SYS_DATA_PART2_3_REG	Register 3 of BLOCK10 (system).	0x0168	RO
EFUSE_RD_SYS_DATA_PART2_4_REG	Register 4 of BLOCK10 (system).	0x016C	RO
EFUSE_RD_SYS_DATA_PART2_5_REG	Register 5 of BLOCK10 (system).	0x0170	RO
EFUSE_RD_SYS_DATA_PART2_6_REG	Register 6 of BLOCK10 (system).	0x0174	RO
EFUSE_RD_SYS_DATA_PART2_7_REG	Register 7 of BLOCK10 (system).	0x0178	RO
Error Status Registers			
EFUSE_RD_REPEAT_ERR0_REG	Programming error record register 0 of BLOCK0.	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	Programming error record register 1 of BLOCK0.	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	Programming error record register 2 of BLOCK0.	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	Programming error record register 3 of BLOCK0.	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	Programming error record register 4 of BLOCK0.	0x0190	RO
EFUSE_RD_RS_ERR0_REG	Programming error record register 0 of BLOCK1-10.	0x01C0	RO
EFUSE_RD_RS_ERR1_REG	Programming error record register 1 of BLOCK1-10.	0x01C4	RO
Control/Status Registers			
EFUSE_CLK_REG	eFuse clock configuration register.	0x01C8	R/W
EFUSE_CONF_REG	eFuse operation mode configuration register.	0x01CC	R/W
EFUSE_CMD_REG	eFuse command register.	0x01D4	R/W
EFUSE_DAC_CONF_REG	Controls the eFuse programming voltage.	0x01E8	R/W
EFUSE_STATUS_REG	eFuse status register.	0x01D0	RO
Interrupt Registers			
EFUSE_INT_RAW_REG	eFuse raw interrupt register.	0x01D8	RO
EFUSE_INT_ST_REG	eFuse interrupt status register.	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse interrupt enable register.	0x01E0	R/W
EFUSE_INT_CLR_REG	eFuse interrupt clear register.	0x01E4	WO
Configuration Registers			
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters.	0x01EC	R/W
EFUSE_WR_TIM_CONF0_REG	Configuration register 0 of eFuse programming timing parameters.	0x01F0	R/W
EFUSE_WR_TIM_CONF1_REG	Configuration register 1 of eFuse programming timing parameters.	0x01F4	R/W
EFUSE_WR_TIM_CONF2_REG	Configuration register 2 of eFuse programming timing parameters.	0x01F8	R/W
Version Register			
EFUSE_DATE_REG	Version control register.	0x01FC	R/W

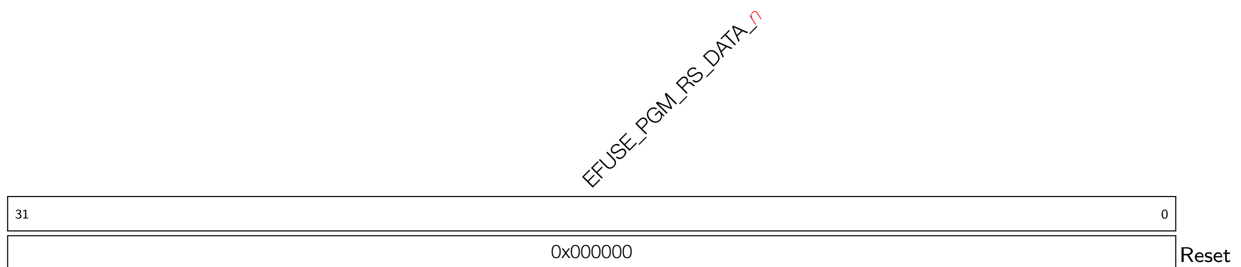
4.6 Registers

Register 4.1: EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+4n*)**



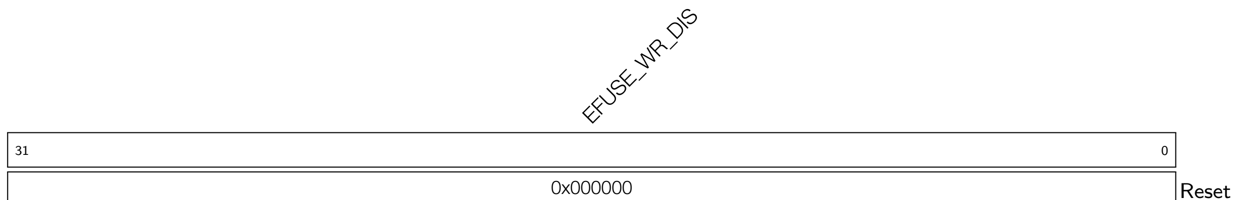
EFUSE_PGM_DATA_*n* The content of the *n*th 32-bit data to be programmed. (R/W)

Register 4.2: EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+4n*)**



EFUSE_PGM_RS_DATA_*n* The content of the *n*th 32-bit RS code to be programmed. (R/W)

Register 4.3: EFUSE_RD_WR_DIS_REG (0x002C)



EFUSE_WR_DIS Disables programming of individual eFuses. (RO)

Register 4.4: EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Bits	Field	Reset
31:29	(reserved)	0 0 0
28:27	EFUSE_RPT4_RESERVED0	0x0
26	EFUSE_USB_FORCE_NOPERSIST	0
25	EFUSE_EXT_PHY_ENABLE	0
24	EFUSE_USB_EXCHG_PINS	0
23:20	(reserved)	0 0 0 0
19	EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	0
18	EFUSE_HARD_DIS_JTAG	0
17	EFUSE_SOFT_DIS_JTAG	0
16	EFUSE_RPT4_RESERVED5	0
15	EFUSE_DIS_BOOT_REMAP	0
14	EFUSE_DIS_CAN	0
13	EFUSE_DIS_USB	0
12	EFUSE_DIS_FORCE_DOWNLOAD	0
11	EFUSE_DIS_DOWNLOAD_DCACHE	0
10	EFUSE_DIS_DOWNLOAD_ICACHE	0
9	EFUSE_DIS_DCACHE	0
8	EFUSE_DIS_ICACHE	0
7	EFUSE_DIS_RTC_RAM_BOOT	0
6:0	EFUSE_RD_DIS	0x0

EFUSE_RD_DIS Disables software reading from individual eFuse blocks (BLOCK4-10). (RO)

EFUSE_DIS_RTC_RAM_BOOT Reserved. (RO)

EFUSE_DIS_ICACHE Set this bit to disable ICache. (RO)

EFUSE_DIS_DCACHE Set this bit to disable DCache. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE Disables Icache when SoC is in Download mode. (RO)

EFUSE_DIS_DOWNLOAD_DCACHE Disables Dcache when SoC is in Download mode. (RO)

EFUSE_DIS_FORCE_DOWNLOAD Set this bit to disable the function that forces chip into download mode. (RO)

EFUSE_DIS_USB Set this bit to disable USB OTG function. (RO)

EFUSE_DIS_CAN Set this bit to disable the TWAI Controller function. (RO)

EFUSE_DIS_BOOT_REMAP Disables capability to Remap RAM to ROM address space. (RO)

EFUSE_RPT4_RESERVED5 Reserved (used for four backups method). (RO)

EFUSE_SOFT_DIS_JTAG Software disables JTAG. When software disabled, JTAG can be activated temporarily by HMAC peripheral. (RO)

EFUSE_HARD_DIS_JTAG Hardware disables JTAG permanently. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Disables flash encryption when in download boot modes. (RO)

EFUSE_USB_EXCHG_PINS Set this bit to exchange USB D+ and D- pins. (RO)

EFUSE_EXT_PHY_ENABLE Set this bit to enable external USB PHY. (RO)

EFUSE_USB_FORCE_NOPERSIST If set, forces USB BVALID to 1. (RO)

EFUSE_RPT4_RESERVED0 Reserved (used for four backups method). (RO)

Register 4.5: EFUSE_RD_REPEAT_DATA1_REG (0x0034)

Bits	31–28	27–24	23	22	21	20–18	17–16	15–7	6	5	4	3–0
Field	EFUSE_KEY_PURPOSE_1	EFUSE_KEY_PURPOSE_0	EFUSE_SECURE_BOOT_KEY_REVOKE2	EFUSE_SECURE_BOOT_KEY_REVOKE1	EFUSE_SECURE_BOOT_KEY_REVOKE0	EFUSE_SPI_BOOT_CRYPT_CNT	EFUSE_WDT_DELAY_SEL	(reserved)	EFUSE_VDD_SPI_FORCE	EFUSE_VDD_SPI_TIEH	EFUSE_VDD_SPI_XPD	(reserved)
Reset	0x0	0x0	0	0	0	0x0	0x0	0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0

EFUSE_VDD_SPI_XPD If VDD_SPI_FORCE is 1, this value determines if the VDD_SPI regulator is powered on. (RO)

EFUSE_VDD_SPI_TIEH If VDD_SPI_FORCE is 1, determines VDD_SPI voltage. 0: VDD_SPI connects to 1.8 V LDO; 1: VDD_SPI connects to VDD_RTC_IO. (RO)

EFUSE_VDD_SPI_FORCE Set this bit to use XPD_VDD_PSI_REG and VDD_SPI_TIEH to configure VDD_SPI LDO. (RO)

EFUSE_WDT_DELAY_SEL Selects RTC watchdog timeout threshold at startup. 0: 40,000 slow clock cycles; 1: 80,000 slow clock cycles; 2: 160,000 slow clock cycles; 3: 320,000 slow clock cycles. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT Enables encryption and decryption, when an SPI boot mode is set. Feature is enabled 1 or 3 bits are set in the eFuse, disabled otherwise. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0 If set, revokes use of secure boot key digest 0. (RO)

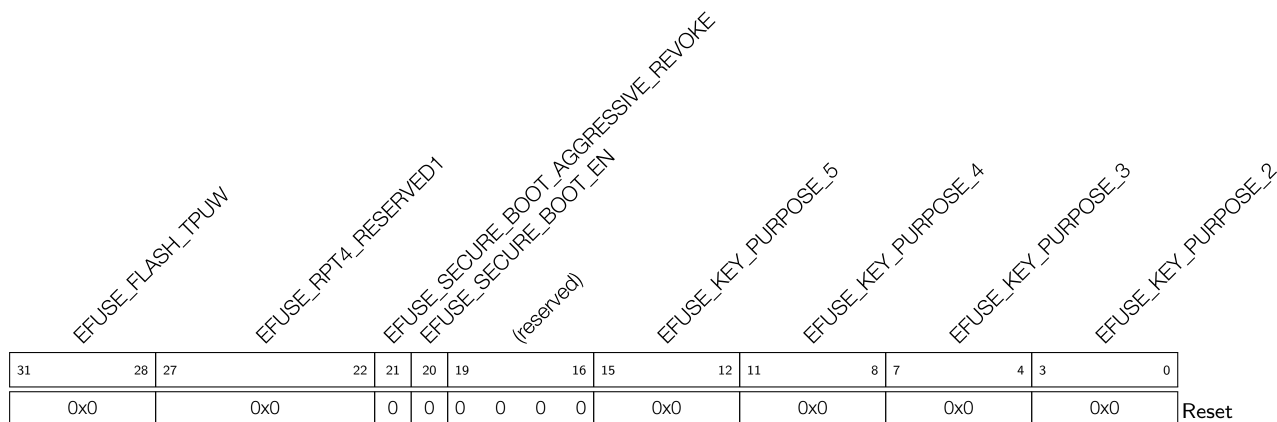
EFUSE_SECURE_BOOT_KEY_REVOKE1 If set, revokes use of secure boot key digest 1. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 If set, revokes use of secure boot key digest 2. (RO)

EFUSE_KEY_PURPOSE_0 Purpose of KEY0. Refer to Table 27 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_1 Purpose of KEY1. Refer to Table 27 *Key Purpose Values*. (RO)

Register 4.6: EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 Purpose of KEY2. Refer to Table 27 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_3 Purpose of KEY3. Refer to Table 27 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_4 Purpose of KEY4. Refer to Table 27 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_5 Purpose of KEY5. Refer to Table 27 *Key Purpose Values*. (RO)

EFUSE_SECURE_BOOT_EN Set this bit to enable secure boot. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE Set this bit to enable aggressive secure boot key revocation mode. (RO)

EFUSE_RPT4_RESERVED1 Reserved (used for four backups method). (RO)

EFUSE_FLASH_TPUW Configures flash startup delay after SoC power-up, in unit of (ms/2). When the value is 15, delay is 7.5 ms. (RO)

Register 4.7: EFUSE_RD_REPEAT_DATA3_REG (0x003C)

31–27	26–11	10	9	8	7–6	5	4	3	2	1	0	
EFUSE_RPT4_RESERVED2	EFUSE_SECURE_VERSION	EFUSE_FORCE_SEND_RESUME	EFUSE_FLASH_TYPE	EFUSE_PIN_POWER_SELECTION	EFUSE_UART_PRINT_CONTROL	EFUSE_ENABLE_SECURITY_DOWNLOAD	EFUSE_DIS_USB_DOWNLOAD_MODE	EFUSE_RPT4_RESERVED3	EFUSE_UART_PRINT_CHANNEL	EFUSE_DIS_LEGACY_SPI_BOOT	EFUSE_DIS_DOWNLOAD_MODE	
0x0	0x00	0	0	0	0x0	0	0	0	0	0	0	Reset

EFUSE_DIS_DOWNLOAD_MODE Set this bit to disable all download boot modes. (RO)

EFUSE_DIS_LEGACY_SPI_BOOT Set this bit to disable Legacy SPI boot mode. (RO)

EFUSE_UART_PRINT_CHANNEL Selects the default UART for printing boot messages. 0: UART0; 1: UART1. (RO)

EFUSE_RPT4_RESERVED3 Reserved (used for four backups method). (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE Set this bit to disable use of USB OTG in UART download boot mode. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Set this bit to enable secure UART download mode (read/write flash only). (RO)

EFUSE_UART_PRINT_CONTROL Set the default UART boot message output mode.
00: Enabled.
01: Enable when GPIO46 is low at reset.
10: Enable when GPIO46 is high at reset.
11: Disabled. (RO)

EFUSE_PIN_POWER_SELECTION Set default power supply for GPIO33-GPIO37, set when SPI flash is initialized. 0: VDD3P3_CPU; 1: VDD_SPI. (RO)

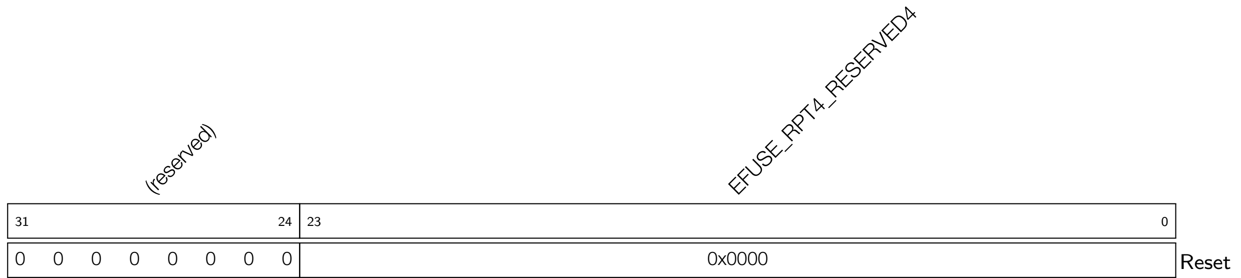
EFUSE_FLASH_TYPE SPI flash type. 0: maximum four data lines, 1: eight data lines. (RO)

EFUSE_FORCE_SEND_RESUME If set, forces ROM code to send an SPI flash resume command during SPI boot. (RO)

EFUSE_SECURE_VERSION Secure version (used by ESP-IDF anti-rollback feature). (RO)

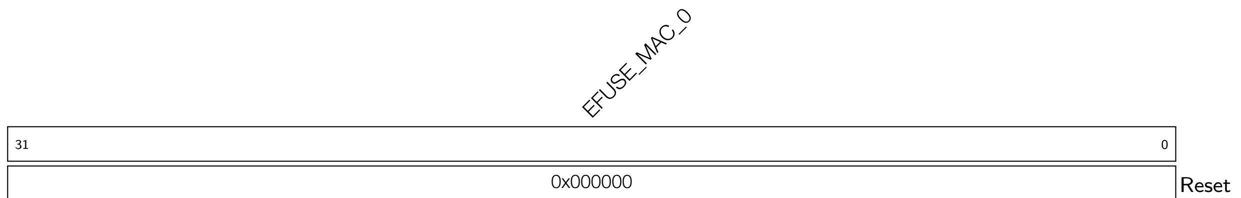
EFUSE_RPT4_RESERVED2 Reserved (used for four backups method). (RO)

Register 4.8: EFUSE_RD_REPEAT_DATA4_REG (0x0040)



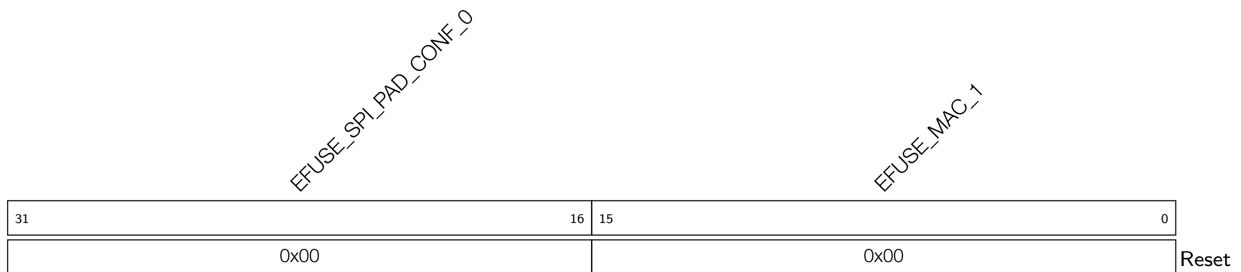
EFUSE_RPT4_RESERVED4 Reserved (used for four backups method). (RO)

Register 4.9: EFUSE_RD_MAC_SPI_SYS_0_REG (0x0044)



EFUSE_MAC_0 Stores the low 32 bits of MAC address. (RO)

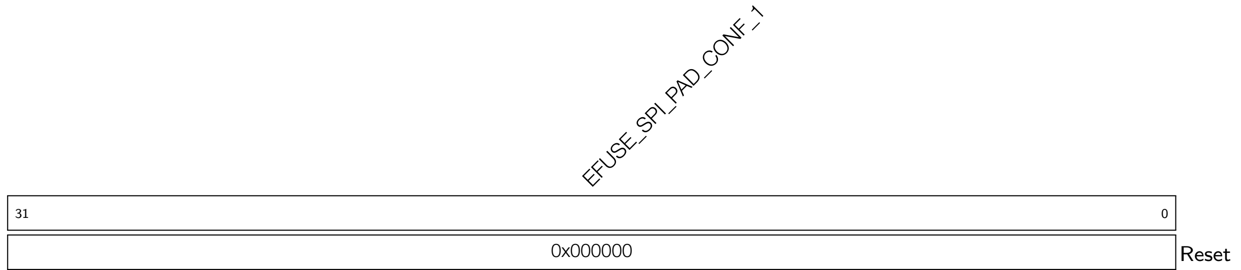
Register 4.10: EFUSE_RD_MAC_SPI_SYS_1_REG (0x0048)



EFUSE_MAC_1 Stores the high 16 bits of MAC address. (RO)

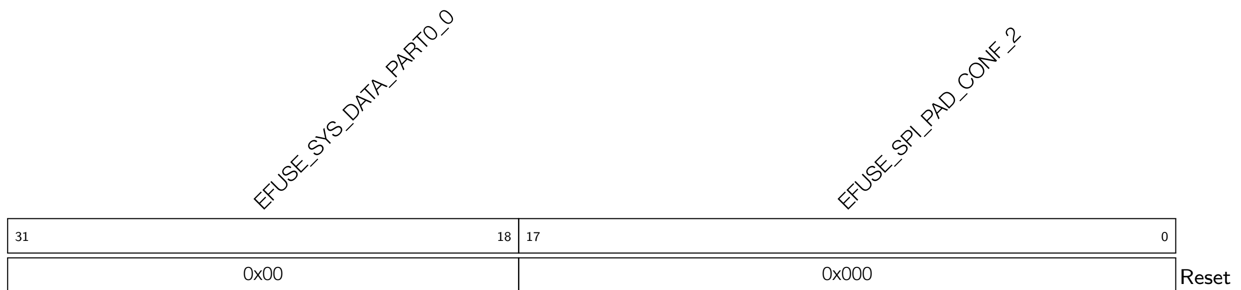
EFUSE_SPI_PAD_CONF_0 Stores the zeroth part of SPI_PAD_CONF. (RO)

Register 4.11: EFUSE_RD_MAC_SPI_SYS_2_REG (0x004C)



EFUSE_SPI_PAD_CONF_1 Stores the first part of SPI_PAD_CONF. (RO)

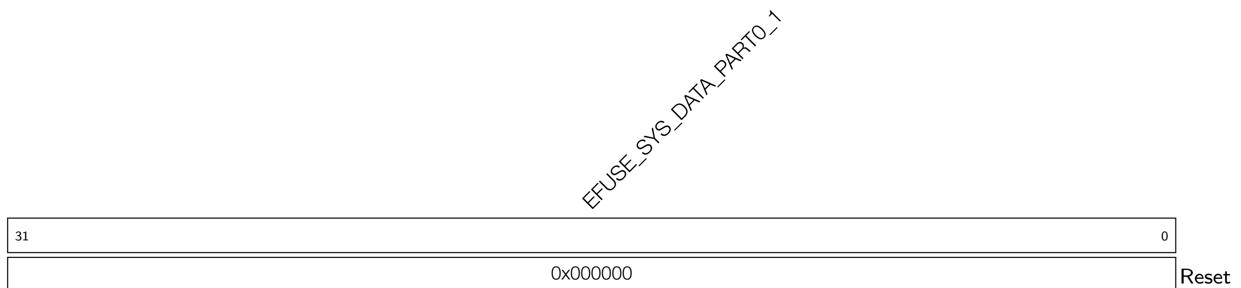
Register 4.12: EFUSE_RD_MAC_SPI_SYS_3_REG (0x0050)



EFUSE_SPI_PAD_CONF_2 Stores the second part of SPI_PAD_CONF. (RO)

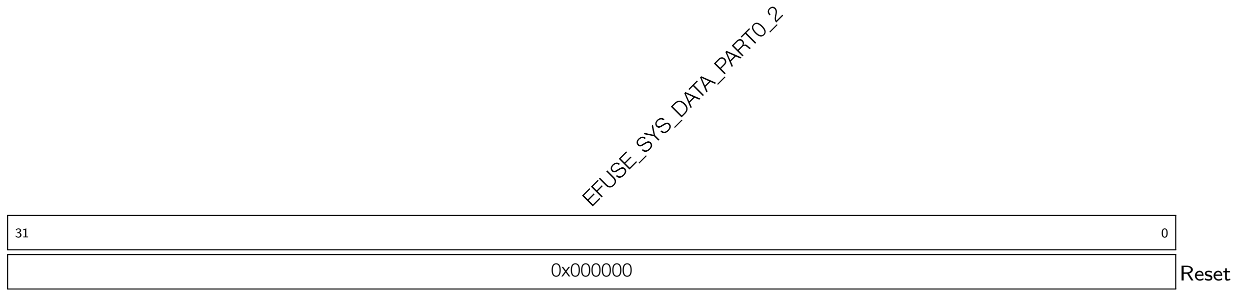
EFUSE_SYS_DATA_PART0_0 Stores the zeroth part of the zeroth part of system data. (RO)

Register 4.13: EFUSE_RD_MAC_SPI_SYS_4_REG (0x0054)



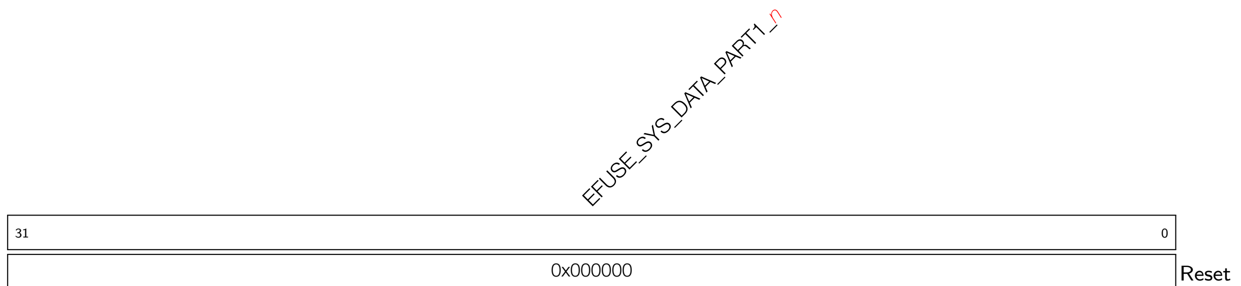
EFUSE_SYS_DATA_PART0_1 Stores the fist part of the zeroth part of system data. (RO)

Register 4.14: EFUSE_RD_MAC_SPI_SYS_5_REG (0x0058)



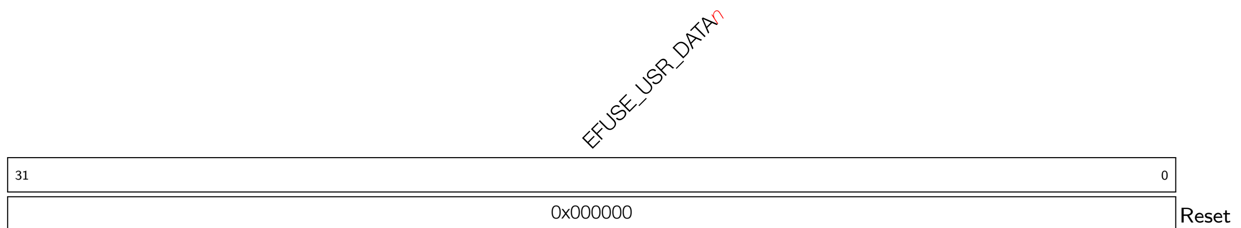
EFUSE_SYS_DATA_PART0_2 Stores the second part of the zeroth part of system data. (RO)

Register 4.15: EFUSE_RD_SYS_DATA_PART1_*n*_REG (*n*: 0-7) (0x005C+4n*)**



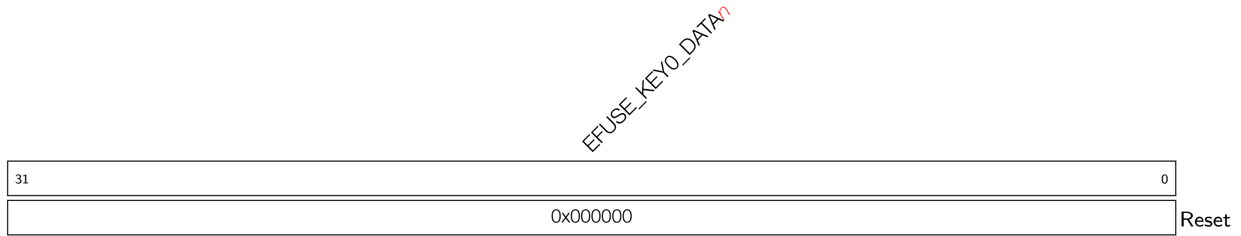
EFUSE_SYS_DATA_PART1_*n* Stores the *n*th 32 bits of the first part of system data. (RO)

Register 4.16: EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+4n*)**



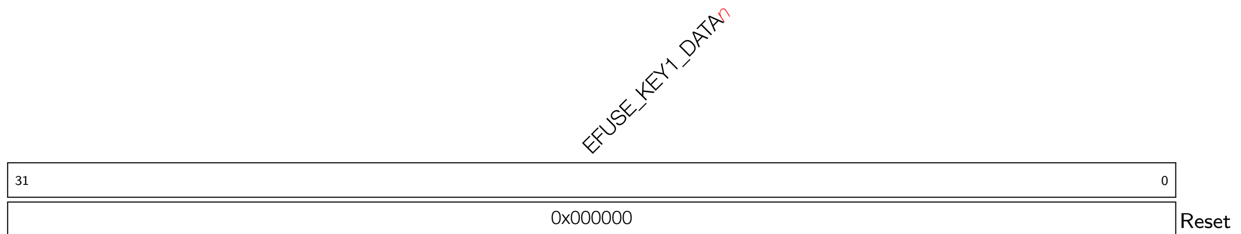
EFUSE_USR_DATA*n* Stores the *n*th 32 bits of BLOCK3 (user). (RO)

Register 4.17: EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+4**n*)



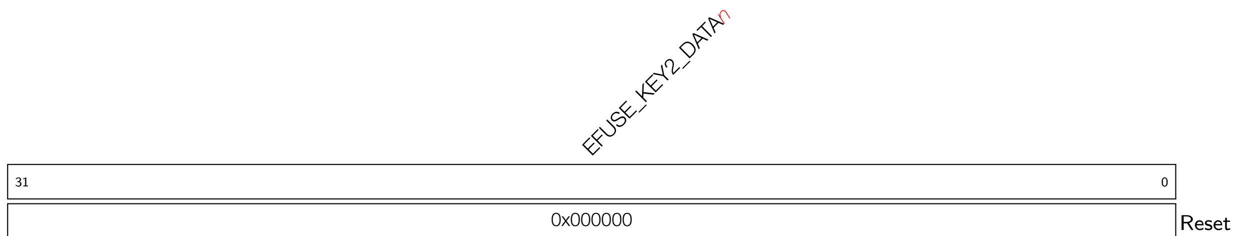
EFUSE_KEY0_DATA*n* Stores the *n*th 32 bits of KEY0. (RO)

Register 4.18: EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+4**n*)



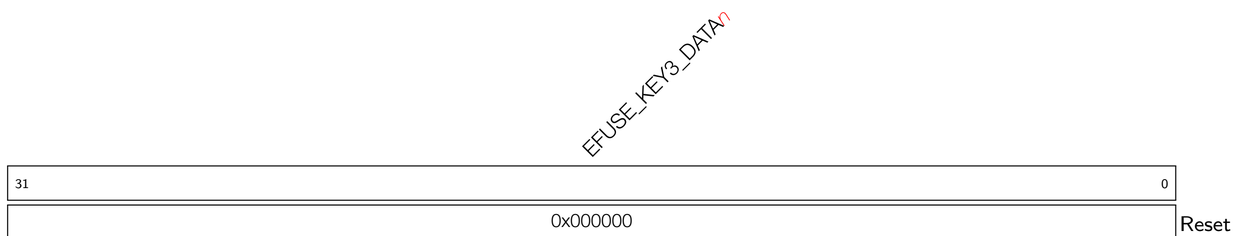
EFUSE_KEY1_DATA*n* Stores the *n*th 32 bits of KEY1. (RO)

Register 4.19: EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+4**n*)



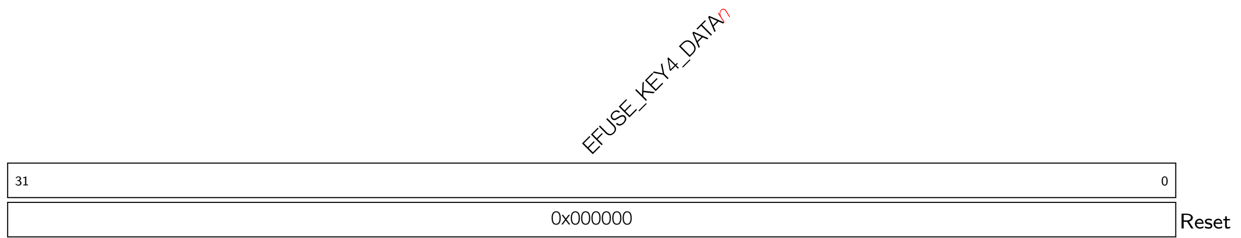
EFUSE_KEY2_DATA*n* Stores the *n*th 32 bits of KEY2. (RO)

Register 4.20: EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+4**n*)



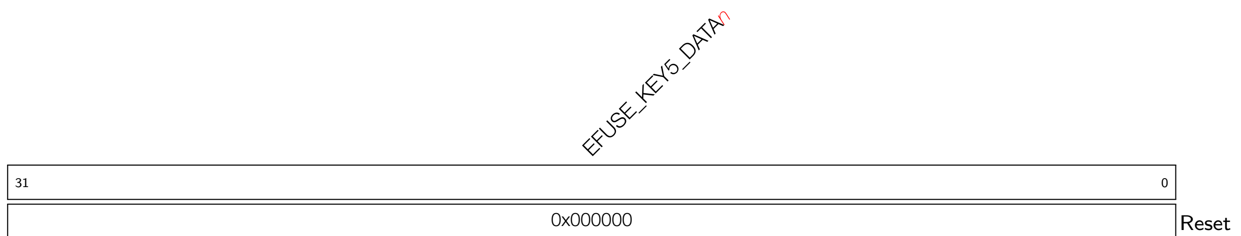
EFUSE_KEY3_DATA*n* Stores the *n*th 32 bits of KEY3. (RO)

Register 4.21: EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+4*n)



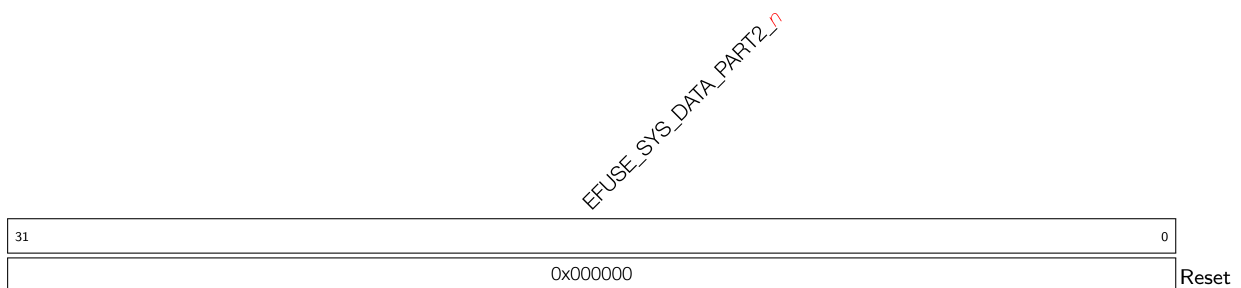
EFUSE_KEY4_DATA*n* Stores the *n*th 32 bits of KEY4. (RO)

Register 4.22: EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+4*n)



EFUSE_KEY5_DATA*n* Stores the *n*th 32 bits of KEY5. (RO)

Register 4.23: EFUSE_RD_SYS_DATA_PART2_*n*_REG (*n*: 0-7) (0x015C+4*n)



EFUSE_SYS_DATA_PART2_*n* Stores the *n*th 32 bits of the 2nd part of system data. (RO)

Register 4.24: EFUSE_RD_REPEAT_ERR0_REG (0x017C)

31..29	28..27	26	25	24	23..20	19	18	17	16	15	14	13	12	11	10	9	8	7	6..0	
(reserved)	EFUSE_RPT4_RESERVED0_ERR	EFUSE_USB_FORCE_NOPERSIST_ERR	EFUSE_EXT_PHY_ENABLE_ERR	EFUSE_USB_EXCHG_PINS_ERR	(reserved)	EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR	EFUSE_HARD_DIS_JTAG_ERR	EFUSE_SOFT_DIS_JTAG_ERR	EFUSE_RPT4_RESERVED5_ERR	EFUSE_DIS_BOOT_REMAP_ERR	EFUSE_DIS_CAN_ERR	EFUSE_DIS_USB_ERR	EFUSE_DIS_FORCE_DOWNLOAD_ERR	EFUSE_DIS_DOWNLOAD_DCACHE_ERR	EFUSE_DIS_DOWNLOAD_ICACHE_ERR	EFUSE_DIS_DCACHE_ERR	EFUSE_DIS_ICACHE_ERR	EFUSE_DIS_RTC_RAM_BOOT_ERR	EFUSE_RD_DIS_ERR	
0 0 0	0x0	0	0	0	0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	Reset

EFUSE_RD_DIS_ERR Any bit equal to 1 denotes a programming error in EFUSE_RD_DIS. (RO)

EFUSE_DIS_RTC_RAM_BOOT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_RTC_RAM_BOOT. (RO)

EFUSE_DIS_ICACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_ICACHE. (RO)

EFUSE_DIS_DCACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DCACHE. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_ICACHE. (RO)

EFUSE_DIS_DOWNLOAD_DCACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_DCACHE. (RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_FORCE_DOWNLOAD. (RO)

EFUSE_DIS_USB_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_USB. (RO)

EFUSE_DIS_CAN_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_CAN. (RO)

EFUSE_DIS_BOOT_REMAP_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_BOOT_REMAP. (RO)

EFUSE_RPT4_RESERVED5_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED5. (RO)

EFUSE_SOFT_DIS_JTAG_ERR Any bit equal to 1 denotes a programming error in EFUSE_SOFT_DIS_JTAG. (RO)

EFUSE_HARD_DIS_JTAG_ERR Any bit equal to 1 denotes a programming error in EFUSE_HARD_DIS_JTAG. (RO)

Continued on the next page...

Register 4.24: EFUSE_RD_REPEAT_ERR0_REG (0x017C)

Continued from the previous page...

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT. (RO)

EFUSE_USB_EXCHG_PINS_ERR Any bit equal to 1 denotes a programming error in EFUSE_USB_EXCHG_PINS. (RO)

EFUSE_EXT_PHY_ENABLE_ERR Any bit equal to 1 denotes a programming error in EFUSE_EXT_PHY_ENABLE. (RO)

EFUSE_USB_FORCE_NOPERSIST_ERR Any bit equal to 1 denotes a programming error in EFUSE_USB_FORCE_NOPERSIST. (RO)

EFUSE_RPT4_RESERVED0_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED0. (RO)

Register 4.25: EFUSE_RD_REPEAT_ERR1_REG (0x0180)

EFUSE_KEY_PURPOSE_1_ERR	EFUSE_KEY_PURPOSE_0_ERR	EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR	EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR	EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR	EFUSE_SPI_BOOT_CRYPT_CNT_ERR	EFUSE_WDT_DELAY_SEL_ERR	(reserved)	EFUSE_VDD_SPI_FORCE_ERR	EFUSE_VDD_SPI_TIEH_ERR	EFUSE_VDD_SPI_XPD_ERR	(reserved)	
31–28	27–24	23	22	21	20–18	17–16	15–7	6	5	4	3–0	
0x0	0x0	0	0	0	0x0	0x0	0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0	Reset

EFUSE_VDD_SPI_XPD_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_XPD. (RO)

EFUSE_VDD_SPI_TIEH_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_TIEH. (RO)

EFUSE_VDD_SPI_FORCE_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_FORCE. (RO)

EFUSE_WDT_DELAY_SEL_ERR Any bit equal to 1 denotes a programming error in EFUSE_WDT_DELAY_SEL. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR Any bit equal to 1 denotes a programming error in EFUSE_SPI_BOOT_CRYPT_CNT. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE0. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE1. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE2. (RO)

EFUSE_KEY_PURPOSE_0_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_0. (RO)

EFUSE_KEY_PURPOSE_1_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_1. (RO)

Register 4.26: EFUSE_RD_REPEAT_ERR2_REG (0x0184)

31–28	27–22	21	20	19–16	15–12	11–8	7–4	3–0	
EFUSE_FLASH_TPUW_ERR	EFUSE_RPT4_RESERVED1_ERR	EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR	EFUSE_SECURE_BOOT_EN_ERR	(reserved)	EFUSE_KEY_PURPOSE_5_ERR	EFUSE_KEY_PURPOSE_4_ERR	EFUSE_KEY_PURPOSE_3_ERR	EFUSE_KEY_PURPOSE_2_ERR	
0x0	0x0	0	0	0 0 0 0	0x0	0x0	0x0	0x0	Reset

EFUSE_KEY_PURPOSE_2_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_2. (RO)

EFUSE_KEY_PURPOSE_3_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_3. (RO)

EFUSE_KEY_PURPOSE_4_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_4. (RO)

EFUSE_KEY_PURPOSE_5_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_5. (RO)

EFUSE_SECURE_BOOT_EN_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_EN. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE. (RO)

EFUSE_RPT4_RESERVED1_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED1. (RO)

EFUSE_FLASH_TPUW_ERR Any bit equal to 1 denotes a programming error in EFUSE_FLASH_TPUW. (RO)

Register 4.27: EFUSE_RD_REPEAT_ERR3_REG (0x0188)

EFUSE_RPT4_RESERVED2_ERR	EFUSE_SECURE_VERSION_ERR	EFUSE_FORCE_SEND_RESUME_ERR	EFUSE_FLASH_TYPE_ERR	EFUSE_PIN_POWER_SELECTION_ERR	EFUSE_UART_PRINT_CONTROL_ERR	EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR	EFUSE_DIS_USB_DOWNLOAD_MODE_ERR	EFUSE_RPT4_RESERVED3_ERR	EFUSE_UART_PRINT_CHANNEL_ERR	EFUSE_DIS_LEGACY_SPI_BOOT_ERR	EFUSE_DIS_DOWNLOAD_MODE_ERR	
31 27	26 11	10	9	8	7 6	5	4	3	2	1	0	
0x0	0x00	0	0	0	0x0	0	0	0	0	0	0	Reset

EFUSE_DIS_DOWNLOAD_MODE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_MODE. (RO)

EFUSE_DIS_LEGACY_SPI_BOOT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_LEGACY_SPI_BOOT. (RO)

EFUSE_UART_PRINT_CHANNEL_ERR Any bit equal to 1 denotes a programming error in EFUSE_UART_PRINT_CHANNEL. (RO)

EFUSE_RPT4_RESERVED3_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED3. (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_USB_DOWNLOAD_MODE. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR Any bit equal to 1 denotes a programming error in EFUSE_ENABLE_SECURITY_DOWNLOAD. (RO)

EFUSE_UART_PRINT_CONTROL_ERR Any bit equal to 1 denotes a programming error in EFUSE_UART_PRINT_CONTROL. (RO)

EFUSE_PIN_POWER_SELECTION_ERR Any bit equal to 1 denotes a programming error in EFUSE_PIN_POWER_SELECTION. (RO)

EFUSE_FLASH_TYPE_ERR Any bit equal to 1 denotes a programming error in EFUSE_FLASH_TYPE. (RO)

EFUSE_FORCE_SEND_RESUME_ERR Any bit equal to 1 denotes a programming error in EFUSE_FORCE_SEND_RESUME. (RO)

EFUSE_SECURE_VERSION_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_VERSION. (RO)

EFUSE_RPT4_RESERVED2_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED2. (RO)

Register 4.28: EFUSE_RD_REPEAT_ERR4_REG (0x0190)

(reserved)	EFUSE_RPT4_RESERVED4_ERR	
31 ... 24	23 ... 0	
0 0 0 0 0 0 0 0	0x0000	Reset

EFUSE_RPT4_RESERVED4_ERR If any bit in RPT4_RESERVED4 is 1, there is a programming error in EFUSE_RPT4_RESERVED4. (RO)

Register 4.29: EFUSE_RD_RS_ERR0_REG (0x01C0)

EFUSE_KEY4_FAIL	EFUSE_KEY4_ERR_NUM	EFUSE_KEY3_FAIL	EFUSE_KEY3_ERR_NUM	EFUSE_KEY2_FAIL	EFUSE_KEY2_ERR_NUM	EFUSE_KEY1_FAIL	EFUSE_KEY1_ERR_NUM	EFUSE_KEY0_FAIL	EFUSE_KEY0_ERR_NUM	EFUSE_USR_DATA_FAIL	EFUSE_USR_DATA_ERR_NUM	EFUSE_SYS_PART1_FAIL	EFUSE_SYS_PART1_NUM	EFUSE_MAC_SPI_8M_FAIL	EFUSE_MAC_SPI_8M_ERR_NUM	
31	30 ... 28	27	26 ... 24	23	22 ... 20	19	18 ... 16	15	14 ... 12	11	10 ... 8	7	6 ... 4	3	2 ... 0	
0	0x0	0	0x0	0	0x0	0	0x0	0	0x0	0	0x0	0	0x0	0	0x0	Reset

EFUSE_MAC_SPI_8M_ERR_NUM The value of this signal means the number of error bytes in BLOCK1. (RO)

EFUSE_MAC_SPI_8M_FAIL 0: Means no failure and that the data of BLOCK1 is reliable; 1: Means that programming BLOCK1 data failed and the number of error bytes is over 5. (RO)

EFUSE_SYS_PART1_NUM The value of this signal means the number of error bytes in BLOCK2. (RO)

EFUSE_SYS_PART1_FAIL 0: Means no failure and that the data of BLOCK2 is reliable; 1: Means that programming BLOCK2 data failed and the number of error bytes is over 5. (RO)

EFUSE_USR_DATA_ERR_NUM The value of this signal means the number of error bytes in BLOCK3. (RO)

EFUSE_USR_DATA_FAIL 0: Means no failure and that the data of BLOCK3 is reliable; 1: Means that programming BLOCK3 data failed and the number of error bytes is over 5. (RO)

EFUSE_KEY*n*_ERR_NUM The value of this signal means the number of error bytes in KEY*n*. (RO)

EFUSE_KEY*n*_FAIL 0: Means no failure and that the data of KEY*n* is reliable; 1: Means that programming KEY*n* failed and the number of error bytes is over 5. (RO)

Register 4.30: EFUSE_RD_RS_ERR1_REG (0x01C4)

31–8	7	6–4	3	2–0	
(reserved)	EFUSE_SYS_PART2_FAIL	EFUSE_SYS_PART2_ERR_NUM	EFUSE_KEY5_FAIL	EFUSE_KEY5_ERR_NUM	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0x0	0	0x0	Reset

EFUSE_KEY5_ERR_NUM The value of this signal means the number of error bytes in KEY5. (RO)

EFUSE_KEY5_FAIL 0: Means no failure and that the data of KEY5 is reliable; 1: Means that programming user data failed and the number of error bytes is over 5. (RO)

EFUSE_SYS_PART2_ERR_NUM The value of this signal means the number of error bytes in BLOCK10. (RO)

EFUSE_SYS_PART2_FAIL 0: Means no failure and that the data of BLOCK10 is reliable; 1: Means that programming BLOCK10 data failed and the number of error bytes is over 5. (RO)

Register 4.31: EFUSE_CLK_REG (0x01C8)

31–17	16	15–3	2	1	0	
(reserved)	EFUSE_CLK_EN	(reserved)	EFUSE_EFUSE_MEM_FORCE_PU	EFUSE_MEM_CLK_FORCE_ON	EFUSE_EFUSE_MEM_FORCE_PD	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	0	Reset

EFUSE_EFUSE_MEM_FORCE_PD If set, forces eFuse SRAM into power-saving mode. (R/W)

EFUSE_MEM_CLK_FORCE_ON If set, forces to activate clock signal of eFuse SRAM. (R/W)

EFUSE_EFUSE_MEM_FORCE_PU If set, forces eFuse SRAM into working mode. (R/W)

EFUSE_CLK_EN If set, forces to enable clock signal of eFuse memory. (R/W)

Register 4.32: EFUSE_CONF_REG (0x01CC)

Bits	31–16	15–0
Field	(reserved)	EFUSE_OP_CODE
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00

EFUSE_OP_CODE 0x5A5A: Operate programming command; 0x5AA5: Operate read command. (R/W)

Register 4.33: EFUSE_CMD_REG (0x01D4)

Bits	31–6	5–2	1	0
Field	(reserved)	EFUSE_BLK_NUM	EFUSE_PGM_CMD	EFUSE_READ_CMD
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0	0

EFUSE_READ_CMD Set this bit to send read command. (R/W)

EFUSE_PGM_CMD Set this bit to send programming command. (R/W)

EFUSE_BLK_NUM The serial number of the block to be programmed. Value 0-10 corresponds to block number 0-10, respectively. (R/W)

Register 4.34: EFUSE_DAC_CONF_REG (0x01E8)

Bits	31–18	17	16–9	8	7–0
Field	(reserved)	EFUSE_OE_CLR	EFUSE_DAC_NUM	EFUSE_DAC_CLK_PAD_SEL	EFUSE_DAC_CLK_DIV
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	255	0	28

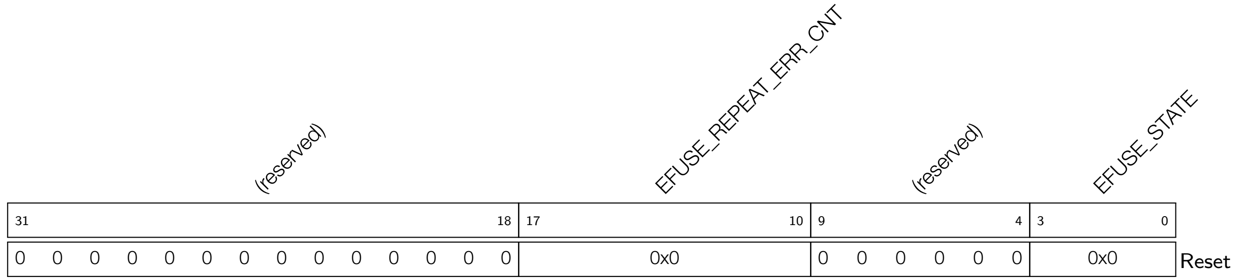
EFUSE_DAC_CLK_DIV Controls the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_CLK_PAD_SEL Don't care. (R/W)

EFUSE_DAC_NUM Controls the rising period of the programming voltage. (R/W)

EFUSE_OE_CLR Reduces the power supply of the programming voltage. (R/W)

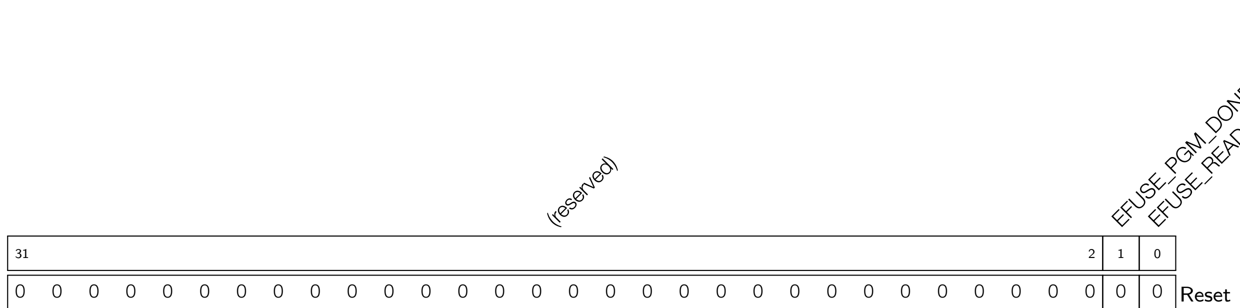
Register 4.35: EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE Indicates the state of the eFuse state machine. (RO)

EFUSE_REPEAT_ERR_CNT Indicates the number of error bits during programming BLOCK0. (RO)

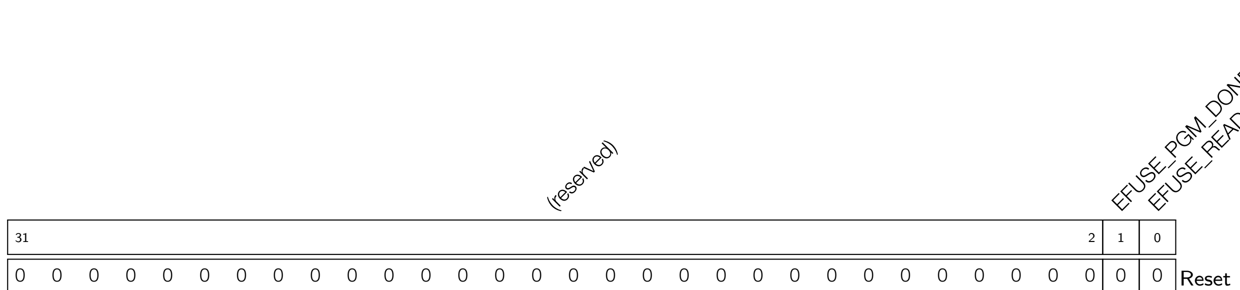
Register 4.36: EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW The raw bit signal for read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_RAW The raw bit signal for pgm_done interrupt. (RO)

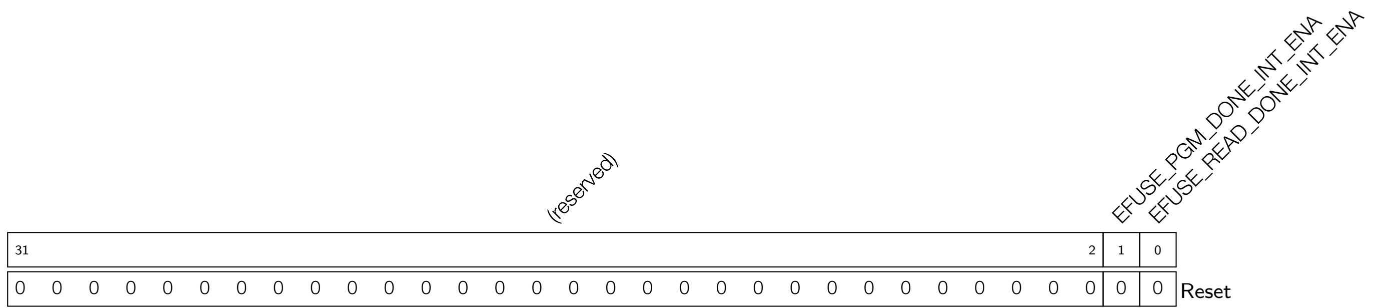
Register 4.37: EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST The status signal for read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_ST The status signal for pgm_done interrupt. (RO)

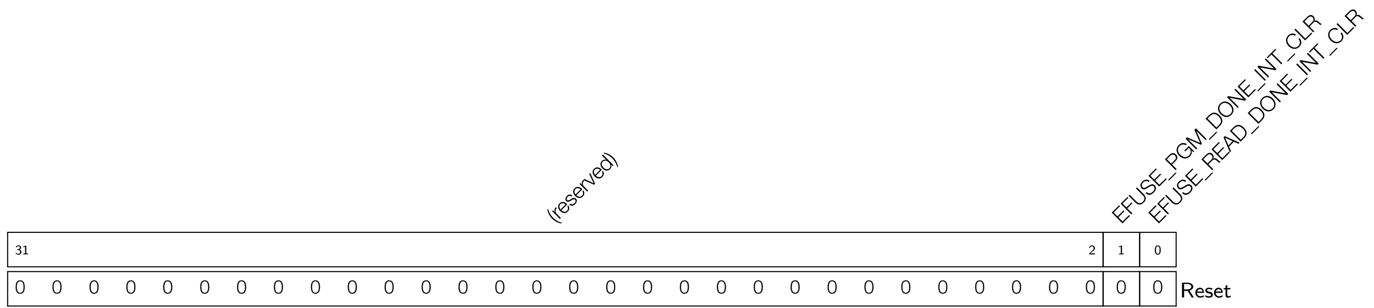
Register 4.38: EFUSE_INT_ENA_REG (0x01E0)



EFUSE_READ_DONE_INT_ENA The enable signal for read_done interrupt. (R/W)

EFUSE_PGM_DONE_INT_ENA The enable signal for pgm_done interrupt. (R/W)

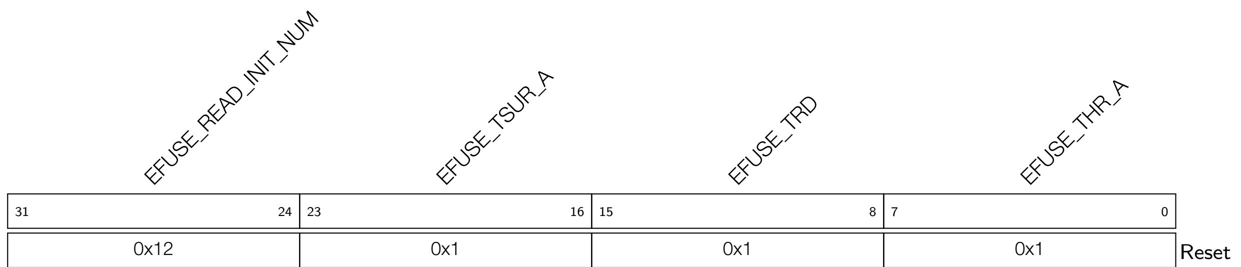
Register 4.39: EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR The clear signal for read_done interrupt. (WO)

EFUSE_PGM_DONE_INT_CLR The clear signal for pgm_done interrupt. (WO)

Register 4.40: EFUSE_RD_TIM_CONF_REG (0x01EC)



EFUSE_THR_A Configures the hold time of read operation. (R/W)

EFUSE_TRD Configures the length of pulse of read operation. (R/W)

EFUSE_TSUR_A Configures the setup time of read operation. (R/W)

EFUSE_READ_INIT_NUM Configures the initial read time of eFuse. (R/W)

Register 4.41: EFUSE_WR_TIM_CONF0_REG (0x01F0)

EFUSE_TPGM (31–16)	EFUSE_TPGM_INACTIVE (15–8)	EFUSE_THP_A (7–0)	
0xc8	0x1	0x1	Reset

EFUSE_THP_A Configures the hold time of programming operation. (R/W)

EFUSE_TPGM_INACTIVE Configures the length of pulse during programming 0 to eFuse. (R/W)

EFUSE_TPGM Configures the length of pulse during programming 1 to eFuse. (R/W)

Register 4.42: EFUSE_WR_TIM_CONF1_REG (0x01F4)

(reserved) (31–24)	EFUSE_PWR_ON_NUM (23–8)	EFUSE_TSUP_A (7–0)	
0 0 0 0 0 0 0 0	0x2880	0x1	Reset

EFUSE_TSUP_A Configures the setup time of programming operation. (R/W)

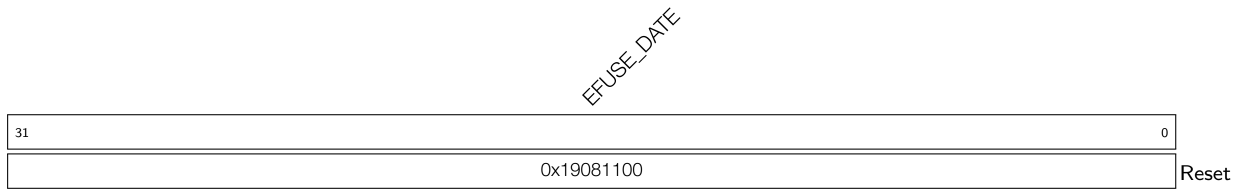
EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. (R/W)

Register 4.43: EFUSE_WR_TIM_CONF2_REG (0x01F8)

(reserved) (31–16)	EFUSE_PWR_OFF_NUM (15–0)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x190	Reset

EFUSE_PWR_OFF_NUM Configures the power outage time for VDDQ. (R/W)

Register 4.44: EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Version control register. (R/W)

5. IO MUX and GPIO Matrix (GPIO, IO_MUX)

5.1 Overview

The ESP32-S2 chip features 43 physical GPIO pads. Each pad can be used as a general-purpose I/O, or be connected to an internal peripheral signal. The IO MUX, RTC IO MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pads. Together these modules provide highly configurable I/O.

Note that the GPIO pads are numbered from 0 ~ 21 and 26 ~ 46, while GPIO46 is input-only.

This chapter describes the selection and connection of the internal signals for the 43 digital pads and control signals: FUN_SEL, IE, OE, WPU, WPD, etc. These internal signals include:

- 116 digital peripheral input signals, control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.
- 182 digital peripheral output signals, control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.
- fast peripheral input and output signals, control signals: IE, OE, etc.
- 22 RTC GPIO signals

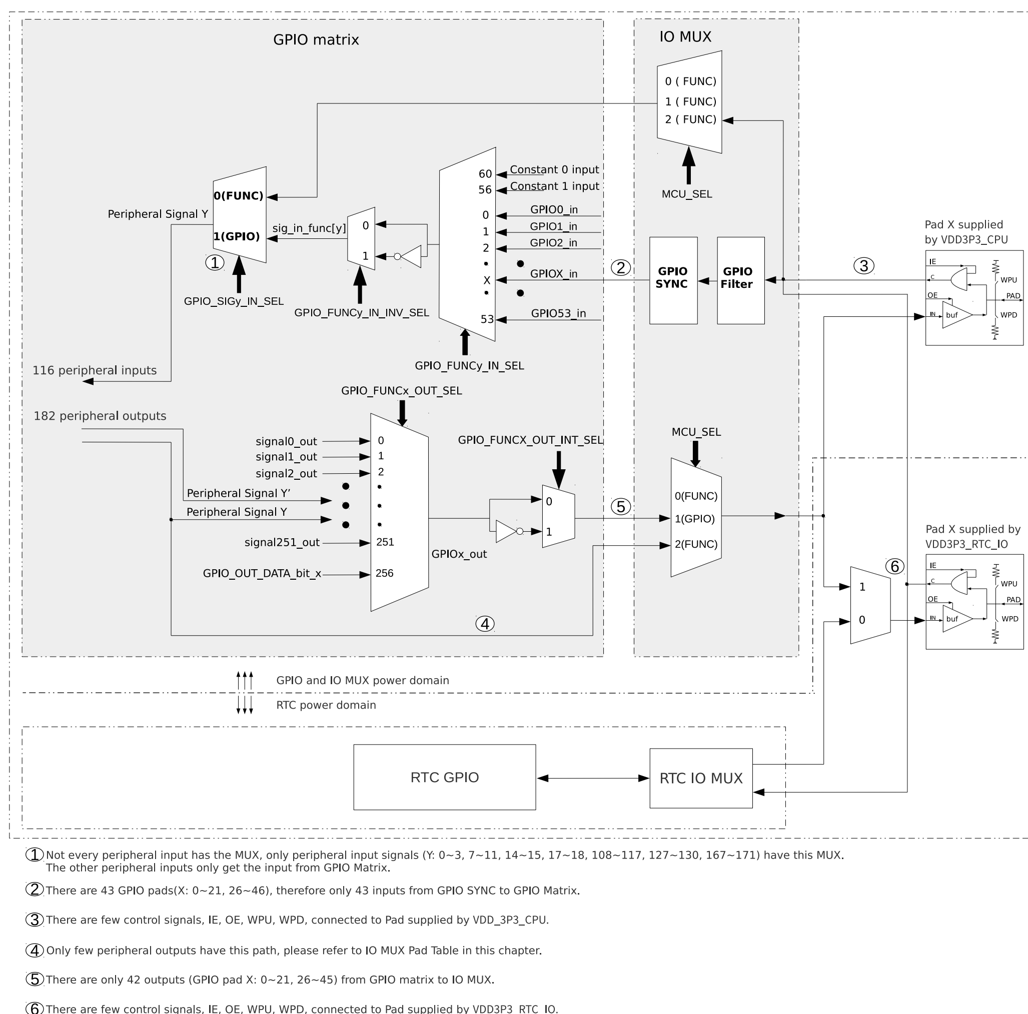


Figure 5-1. IO MUX, RTC IO MUX and GPIO Matrix Overview

Figure 5-1 shows the overview of IO MUX, RTC IO MUX and GPIO matrix.

1. IO MUX provides one configuration register IO_MUX_*n*_REG for each GPIO pad. The pad can be configured to

   - perform GPIO function routed by GPIO matrix;
   - or perform direct connection bypassing GPIO matrix.

   Some high-speed digital signals (SPI, JTAG, UART) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pads directly to the peripheral.

   See Section 5.11 for the IO MUX functions for each I/O pad.

2. GPIO matrix is a full-switching matrix between the peripheral input/output signals and the pads.

   - For input to the chip: each of the 116 internal peripheral inputs can select any GPIO pad as their input source.
   - For output from the chip: each GPIO pad can select any of the 182 peripheral output signals for its output.

   See Section 5.10 for the list of peripheral signals via GPIO matrix.

3. RTC IO MUX is used to connect GPIO pads to their low-power and analog functions. Only a subset of GPIO pads have these optional RTC functions.

   See Section 5.12 for the list of RTC IO MUX functions.

5.2 Peripheral Input via GPIO Matrix

5.2.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 43 GPIOs (0 ~ 21, 26 ~ 46), see Table 36. Meanwhile, register corresponding to the peripheral should be set to receive input signal via GPIO matrix.

5.2.2 Synchronization

When signals are directed using the GPIO matrix, the signal will be synchronized to the APB bus clock by the GPIO SYNC hardware. This synchronization applies to all GPIO matrix signals but does not apply when using the IO MUX, see Figure 5-1.

Figure 5-2 shows the functionality of GPIO SYNC. In the figure, negative sync and positive sync mean GPIO input is synchronized on APB clock falling edge and on APB clock rising edge, respectively.

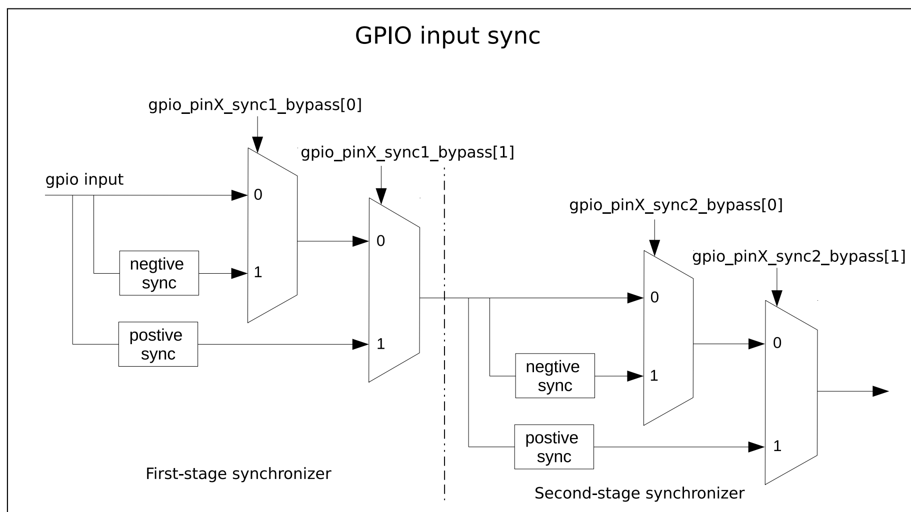


Figure 5-2. GPIO Input Synchronized on Clock Rising Edge or on Falling Edge

5.2.3 Functional Description

To read GPIO pad *X* into peripheral signal *Y*, follow the steps below:

1. Configure register GPIO_FUNC*y*_IN_SEL_CFG_REG corresponding to peripheral signal *Y* in GPIO matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via GPIO matrix.
   - Set GPIO_FUNC*y*_IN_SEL to the value corresponding to GPIO pad *X*.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, there is no MUX module in Figure 5-1 for these signals (see note 1 below Figure 5-1). These peripherals can only receive input signals via GPIO matrix.

2. Enable the filter for pad input signals by setting the register IO_MUX_FILTER_EN. Only the signals with a valid width of more than two clock cycles can be sampled, see Figure 5-3.

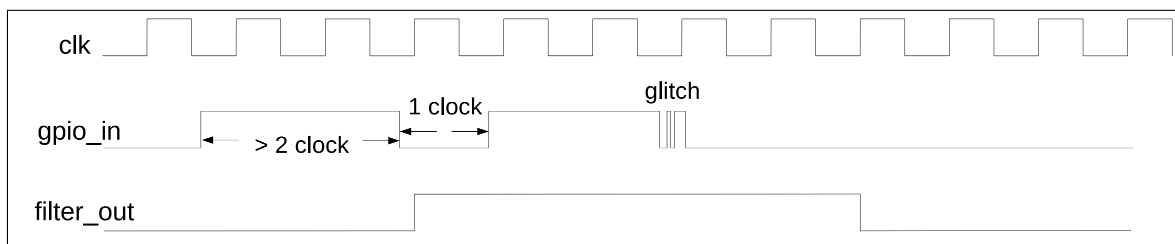


Figure 5-3. Filter Timing Diagram of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pad *X* as follows:
   - Set GPIO_PIN*x*_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first clock, see Figure 5-2.
   - Set GPIO_PIN*x*_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second clock, see Figure 5-2.

4. Configure IO MUX register to enable pad input. For this end, please set IO_MUX_*x*_REG corresponding to GPIO pad *X* as follows:
   - Set IO_MUX_FUN_IE to enable input.

- Set or clear IO_MUX_FUN_WPU and IO_MUX_FUN_WPD, as desired, to enable or disable pull-up and pull-down resistors.

For example, to connect RMT channel 0 input signal (rmt_sig_in0, signal index 83) to GPIO40, please follow the steps below. Note that GPIO40 is also named as MTDO pin.

1. Set GPIO_SIG83_IN_SEL in register GPIO_FUNC83_IN_SEL_CFG_REG to enable peripheral signal input via GPIO matrix.
2. Set GPIO_FUNC83_IN_SEL in register GPIO_FUNC83_IN_SEL_CFG_REG to 40.
3. Set IO_MUX_FUN_IE in register IO_MUX_GPIO40_REG to enable pad input.

Note:

- One input pad can be connected to multiple peripheral input signals.
- The input signal can be inverted by configuring GPIO_FUNC*y*_IN_INV_SEL.
- It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pad. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
  - When GPIO_FUNC*y*_IN_SEL is 0x3C, input signal *X* is always 0.
  - When GPIO_FUNC*y*_IN_SEL is 0x38, input signal *X* is always 1.

5.2.4 Simple GPIO Input

GPIO_IN_REG/GPIO_IN1_REG holds the input values of each GPIO pad. The input value of any GPIO pad can be read at any time without configuring GPIO matrix for a particular peripheral signal. However, it is necessary to enable the input in IO MUX by setting IO_MUX_FUN_IE bit in register IO_MUX_*n*_REG corresponding to pad *X*, as mentioned in Section 5.2.2.

5.3 Peripheral Output via GPIO Matrix

5.3.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (0 ~ 11, 14 ~ 18, and etc.) to one of the 42 GPIOs (0 ~ 21, 26 ~ 45). See Table 36.

The output signal is routed from the peripheral into GPIO matrix and then into IO MUX. IO MUX must be configured to set the chosen pad to GPIO function. This causes the output GPIO signal to be connected to the pad.

> **Note:**
> There is a range of peripheral output signals (223 ~ 227) which are not connected to any peripheral. These can be used to input a signal from one GPIO pad and output directly to another GPIO pad.

5.3.2 Functional Description

Some of the 182 output signals can be set to go through GPIO matrix into IO MUX and then to a pad. Figure 5-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pad *X*, follow these steps:

1. Configure register GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pad *X* in GPIO matrix. Recommended operation: use corresponding W1TS (write 1 to set) and W1TC (write

1 to clear) registers to set or clear GPIO_ENABLE_REG.

- Set the GPIO_FUNC*x*_OUT_SEL field in register GPIO_FUNC*x*_OUT_SEL_CFG_REG to the index of the desired peripheral output signal *Y*.
- If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OEN_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG or in register GPIO_ENABLE1_W1TS_REG, corresponding to GPIO pad *X*. To have the output enable signal decided by internal logic (see the column "Output enable of output signals" in Table 36), clear GPIO_FUNC*x*_OEN_SEL bit instead.
- Clear the corresponding bit in register GPIO_ENABLE_W1TC_REG or in register GPIO_ENABLE1_W1TC_REG to disable the output from the GPIO pad.

2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pad *X*.
3. Configure IO MUX register to enable output via GPIO matrix. Set the IO_MUX_*x*_REG corresponding to GPIO pad *X* as follows:
   - Set the field IO_MUX_MCU_SEL to IO_MUX function corresponding to GPIO pad *X*. This is Function 1, numeric value 1, for all pins.
   - Set the IO_MUX_FUN_DRV field to the desired value for output strength (0 ~ 3). The higher the driver strength, the more current can be sourced/sunk from the pin.
     - 0: ~5 mA
     - 1: ~10 mA
     - 2: ~20 mA (Default value)
     - 3: ~40 mA
   - If using open drain mode, set/clear the IO_MUX_FUN_WPU and IO_MUX_FUN_WPD bits to enable/disable the internal pull-up/down resistors.

Note:

- The output signal from a single peripheral can be sent to multiple pads simultaneously.
- GPIO46 can not be used as an output.
- The output signal can be inverted by setting GPIO_FUNC*n*_OUT_INV_SEL bit.

5.3.3 Simple GPIO Output

GPIO matrix can also be used for simple GPIO output. This can be done as below:

- Set GPIO matrix GPIO_FUNC*n*_OUT_SEL with a special peripheral index 256 (0x100);
- Set the corresponding bit in GPIO_OUT_REG[31:0] or GPIO_OUT1_REG[21:0] register to the desired GPIO output value.

Note:

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31] correspond to GPIO0 ~ GPIO31, and GPIO_OUT_REG[25:22] are invalid.

- GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[13] correspond to GPIO32 ~ GPIO45, and GPIO_OUT1_REG[21:14] are invalid.
- Recommended operation: use corresponding W1TS and W1TC registers, such as GPIO_OUT_W1TS/GPIO_OUT_W1TC to set or clear the registers GPIO_OUT_REG/GPIO_OUT1_REG.

5.3.4 Sigma Delta Modulated Output

5.3.4.1 Functional Description

ESP32-S2 provides a second-order sigma delta modulation module and eight independent modulation channels. The channels are capable to output 1-bit signals (output index: 100 ~ 107) with sigma delta modulation, and by default output is enabled for these channels. This module can also output PDM (pulse density modulation) signal with configurable duty cycle. The transfer function is:

$$H(z) = X(z)z^{-1} + E(z)(1-z^{-1})^2$$

E(z) is quantization error and X(z) is the input.
Sigma Delta modulator supports scaling down of APB_CLK by divider 1 ~ 256:

- Set GPIOSD_FUNCTION_CLK_EN to enable the modulator clock.
- Configure register GPIOSD_SD*n*_PRESCALE (*n* is 0 ~ 7 for eight channels).

After scaling, the clock cycle is equal to one pulse output cycle from the modulator.

GPIOSD_SD*n*_IN is a signed number with a range of [-128, 127] and is used to control the duty cycle [1] of PDM output signal.

- GPIOSD_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIOSD_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIOSD_SD*n*_IN = 127, the duty cycle of the output signal is close to 100%.

The formula for calculating PDM signal duty cycle is shown as below:

$$Duty_Cycle = \frac{GPIOSD_SDn_IN + 128}{256}$$

> **Note:**
> For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period (several pulse cycles, for example 256 pulse cycles).

5.3.4.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pad via GPIO matrix, see Section 5.3.2.
- Enable the modulator clock by setting the register GPIOSD_FUNCTION_CLK_EN.
- Configure the divider value by setting the register GPIOSD_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting the register GPIOSD_SD*n*_IN.

5.4 Dedicated GPIO

5.4.1 Overview

The dedicated GPIO module, consisting of eight input/output channels, is specially designed for CPU interaction with GPIO matrix and IO MUX. Peripheral input/output signals for input/output channels both are indexed from 235 to 242. By default, the output is enabled for output channels.

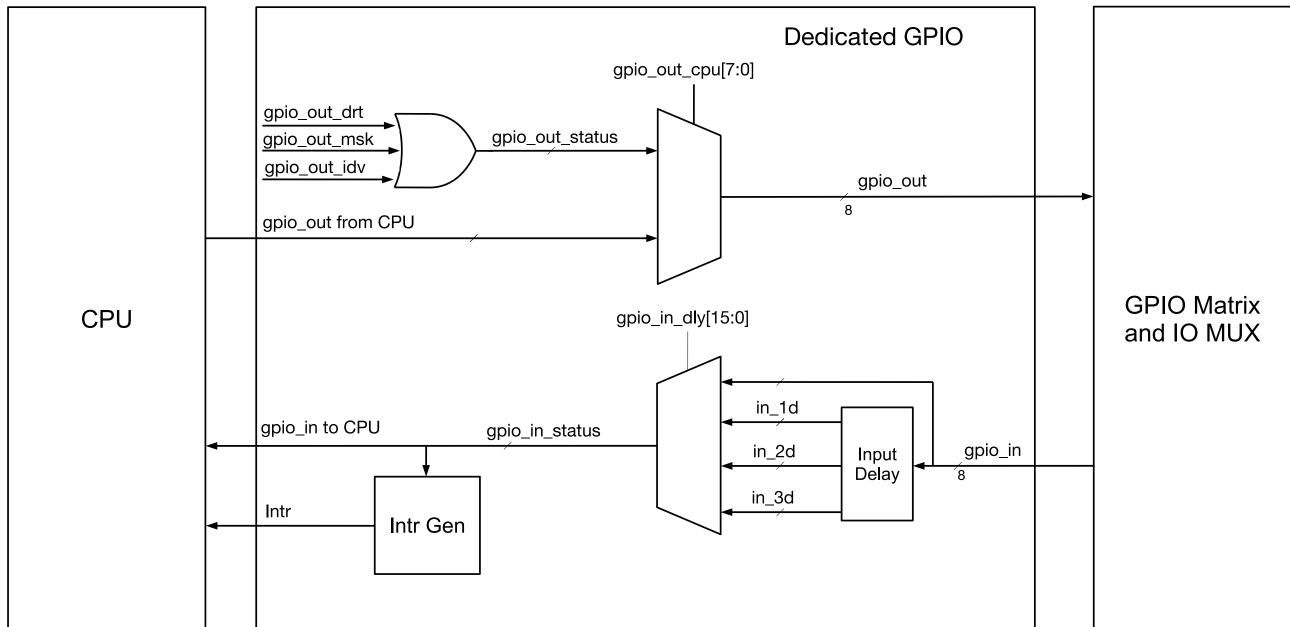


Figure 5-4. Dedicated GPIO Diagram

Figure 5-4 shows the structure of dedicated GPIO module. Users can enable the clock of this module by setting the bit SYSTEM_CLK_EN_DEDICATED_GPIO in register SYSTEM_CPU_PERI_CLK_EN_REG, and reset this module by setting the bit SYSTEM_RST_EN_DEDICATED_GPIO in register SYSTEM_CPU_PERI_RST_EN_REG first, and then clearing this bit. For more information, please refer to Table 87 *Peripheral Clock Gating and Reset Bits* in Chapter 15 *System Registers (SYSTEM)*.

5.4.2 Features

Dedicated GPIO module has the following features:

- Eight output and eight input channels
- Each channel accessible with registers or directly by CPU instructions
- Configurable delay on input channels
- Interrupts on input channels

5.4.3 Functional Description

Dedicated GPIOs may be accessed using registers or directly by calling specific CPU instructions.

When accessing output channels, select between registers and CPU by configuring DEDIC_GPIO_OUT_CPU_REG:

- DEDIC_GPIO_OUT_CPU_SEL*n* = 0, drive GPIO output via registers.
- DEDIC_GPIO_OUT_CPU_SEL*n* = 1, drive GPIO output via CPU instructions.

The dedicated GPIO module also provides two ways to read input channels:

- Query GPIO input value via registers.
- Read GPIO input value via CPU instructions.

5.4.3.1 Accessing GPIO via Registers

Users can control GPIO output via registers in the following ways:

- Write GPIO output value directly by configuring the register DEDIC_GPIO_OUT_DRT_REG.
- Write GPIO output value via masked access by configuring the register DEDIC_GPIO_OUT_MSK_REG.
- Write GPIO output value via individual bits by configuring the register DEDIC_GPIO_OUT_IDV_REG.

User can read the register DEDIC_GPIO_OUT_SCAN_REG to check GPIO status, i.e. gpio_out_status in Figure 5-4, via software.

Users can get a dedicated GPIO input value by reading the register DEDIC_GPIO_IN_SCAN_REG, i.e. the gpio_in_status in Figure 5-4, via software.

The dedicated GPIO module supports for a delay of 1/2/3 clock cycle(s) for input signals, or with no delay, which can be controlled by configuring the register DEDIC_GPIO_IN_DLY_REG for each individual channel. GPIO input status is indicated by interrupts. Users can configure the register DEDIC_GPIO_INTR_RCGN_REG to set trigger modes:

- 0/1: no interrupt
- 2: low level trigger
- 3: high level trigger
- 4: falling edge trigger
- 5: rising edge trigger
- 6/7: edges trigger

5.4.3.2 Accessing GPIO with CPU

CPU can also read/write a dedicated GPIO via instructions.

- Set bits in output channel.
  Assembly syntax: SET_BIT_GPIO_OUT mask
  Addressing: immediate addressing
  Function: write 1 to set the corresponding bits in user register GPIO_OUT, i.e. the gpio_out in Figure 5-4. The "mask" is 8 bits wide. The bits in GPIO_OUT, corresponding to the bits in "mask" with the value of 1, will be set to 1, while the other bits in GPIO_OUT remain unaffected.
- Clear bits in output channel.
  Assembly syntax: CLR_BIT_GPIO_OUT mask
  Addressing: immediate addressing
  Function: write 1 to clear the corresponding bits in user register GPIO_OUT. The "mask" is 8 bits wide. The bits in GPIO_OUT, corresponding to the bits in "mask" with the value of 1, will be cleared, while the other bits in GPIO_OUT remain unaffected.

- Set or clear bits in output channel with masked access
  Assembly syntax: WR_MASK_GPIO_OUT value, mask
  Addressing: register addressing
  Function: write value to user register GPIO_OUT via masked access. The "value" is 8 bits wide, which represents the value to write. The "mask" is 8 bits wide, which represents the bits in GPIO_OUT to be manipulated. For example, mask 0x03 (0000 0011) indicates that the write value is only valid for GPIO_OUT[0] and GPIO_OUT[1]. Only the bits in GPIO_OUT, corresponding to the bits with the value of 1 in the "mask", are updated, that is, updated to the value stored in "value".

- Write "art" register to the output channel.
  Assembly syntax: WUR.GPIO_OUT art
  Addressing: register addressing
  Function: write the value of the address register "art" to the user register GPIO_OUT. Register "art" is 32 bits wide, and only low 8 bits are valid when using this instruction.

- Read the output channel to "arr" register.
  Assembly syntax: RUR.GPIO_OUT arr
  Addressing: register addressing
  Function: read the value of the user register GPIO_OUT to the address register "arr". Register "arr" is 32 bits wide, and only low 8 bits are valid when using this instruction.

- Read the input channel to "l" register.
  Assembly syntax: GET_GPIO_IN l
  Addressing: register addressing
  Function: read the value of the user register GPIO_IN, i.e. the gpio_in in Figure 5-4, to the address register "l". Register "l" is 32 bits wide, the high 24 bits of which are 0, and low 8 bits of which corresponds to the value of the user register GPIO_IN.

5.5 Direct I/O via IO MUX

5.5.1 Overview

Some high-speed signals (SPI and JTAG) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pads directly to the peripheral.

This option is less flexible than routing signals via GPIO matrix, as the IO MUX register for each GPIO pad can only select from a limited number of functions, but high-frequency digital performance can be improved.

5.5.2 Functional Description

Two registers must be configured in order to bypass GPIO matrix for peripheral input signals:

1. IO_MUX_MCU_SEL for the GPIO pad must be set to the required pad function. For the list of pad functions, please refer to Section 5.11.
2. Set GPIO_SIG*n*_IN_SEL to low level to route the input directly to the peripheral.

To bypass GPIO matrix for peripheral output signals, IO_MUX_MCU_SEL for the GPIO pad must be set to the required pad function. For the list of pad functions, please refer to Section 5.11.

Note:

For peripheral I/O signals, not all signals can be connected to peripheral via IO MUX. Some specific input signals and some specific output signals can only be connected to peripheral via GPIO matrix.

5.6 RTC IO MUX for Low Power and Analog I/O

5.6.1 Overview

22 GPIO pads have low power capabilities (RTC domain) and analog functions which are handled by the RTC subsystem of ESP32-S2. IO MUX and GPIO matrix are not used for these functions, rather, RTC IO MUX is used to redirect input/output signals to the RTC subsystem.

When configured as RTC GPIOs, the output pads can still retain the output level value when the chip is in Deep-sleep mode, and the input pads can wake up the chip from Deep-sleep.

Section 5.12 lists the RTC_MUX pins and their functions.

5.6.2 Functional Description

Each pad with analog and RTC functions is controlled by RTCIO_TOUCH_PAD*n*_MUX_SEL bit in register RTCIO_TOUCH_PAD*n*_REG. By default all bits in these registers are set to 0, routing all input/output signals via IO MUX.

If RTCIO_TOUCH_PAD*n*_MUX_SEL is set to 1, then input/output signals to and from that pad is routed to the RTC subsystem. In this mode, RTCIO_TOUCH_PAD*n*_REG is used for digital input/output and the analog features of the pad are also available.

Please refer to Section 5.12 for the list of RTC pin functions and the mapping table of GPIO pads to their analog functions. Note that RTCIO_TOUCH_PAD*n*_REG applies the RTC GPIO pin numbering, not the GPIO pad numbering.

5.7 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-S2 is in Light-sleep mode. If IO_MUX_SLP_SEL in register IO_MUX_*n*_REG for a GPIO pad is set to 1, a different set of bits will be used to control the pad when the chip is in Light-sleep mode.

Table 35: Pin Function Register for IO MUX Light-sleep Mode

IO MUX Function	Normal Execution OR IO_MUX_SLP_SEL = 0	Light-sleep Mode AND IO_MUX_SLP_SEL = 1
Output Drive Strength	IO_MUX_FUN_DRV	IO_MUX_FUN_DRV
Pullup Resistor	IO_MUX_FUN_WPU	IO_MUX_MCU_WPU
Pulldown Resistor	IO_MUX_FUN_WPD	IO_MUX_MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)[1]	IO_MUX_MCU_OE

If IO_MUX_SLP_SEL is set to 0, pin functions remain the same in both normal execution and Light-sleep mode.

Note:
Please refer to Section 5.3.2 for how to enable output in normal execution (when IO_MUX_SLP_SEL = 0).

5.8 Pad Hold Feature

Each IO pad (including the RTC pads) has an individual hold function controlled by a RTC register. When the pad is set to hold, the state is latched at that moment and will not change no matter how the internal signals change

or how the IO MUX/GPIO configuration is modified. Users can use the hold function for the pads to retain the pad state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

Note:

- For digital pads, to maintain pad input/output status in Deep-sleep mode, users can set RTC_CNTL_DG_PAD_FORCE_UNHOLD to 0 before powering down. For RTC pads, the input and output values are controlled by the corresponding bits of register RTC_CNTL_PAD_HOLD_REG, and users can set it to 1 to hold the value or set it to 0 to unhold the value.
- For digital pads, to disable the hold function after the chip is woken up, users can set RTC_CNTL_DG_PAD_FORCE_UNHOLD to 1. To maintain the hold function of the pad, users can change the corresponding bit in register RTC_CNTL_PAD_HOLD_REG to 1.

5.9 I/O Pad Power Supplies

For more information on the power supply for IO pads, please refer to Pin Definition in *ESP32-S2 Datasheet*.

5.9.1 Power Supply Management

Each ESP32-S2 digital pin is connected to one of the four different power domains.

- VDD3P3_RTC_IO: the input power supply for both RTC and CPU
- VDD3P3_CPU: the input power supply for CPU
- VDD3P3_RTC: the input power supply for RTC analog part
- VDD_SPI: configurable power supply

VDD_SPI can be configured to use an internal LDO. The LDO input is VDD3P3_RTC_IO and the output is 1.8 V. If the LDO is not enabled, VDD_SPI is connected directly to the same power supply as VDD3P3_RTC_IO.

The VDD_SPI configuration is determined by the value of strapping pin GPIO45, or can be overriden by eFuse and/or register settings. See *ESP32-S2 Datasheet* sections Power Scheme and Strapping Pins for more details.

5.10 Peripheral Signal List

Table 36 shows the peripheral input/output signals via GPIO matrix.

Table 36: GPIO Matrix

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe
1	SPID_in	0	yes	SPID_out	SPID_oe
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe
4	-	-	-	SPICLK_out_mux	SPICLK_oe
5	-	-	-	SPICS0_out	SPICS0_oe
6	-	-	-	SPICS1_out	SPICS1_oe
7	SPID4_in	0	yes	SPID4_out	SPID4_oe

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
8	SPID5_in	0	yes	SPID5_out	SPID5_oe
9	SPID6_in	0	yes	SPID6_out	SPID6_oe
10	SPID7_in	0	yes	SPID7_out	SPID7_oe
11	SPIDQS_in	0	yes	SPIDQS_out	SPIDQS_oe
14	U0RXD_in	0	yes	U0TXD_out	1'd1
15	U0CTS_in	0	yes	U0RTS_out	1'd1
16	U0DSR_in	0	no	U0DTR_out	1'd1
17	U1RXD_in	0	yes	U1TXD_out	1'd1
18	U1CTS_in	0	yes	U1RTS_out	1'd1
21	U1DSR_in	0	no	U1DTR_out	1'd1
23	I2S0O_BCK_in	0	no	I2S0O_BCK_out	1'd1
25	I2S0O_WS_in	0	no	I2S0O_WS_out	1'd1
27	I2S0I_BCK_in	0	no	I2S0I_BCK_out	1'd1
28	I2S0I_WS_in	0	no	I2S0I_WS_out	1'd1
29	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe
30	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe
39	pcnt_sig_ch0_in0	0	no	gpio_wlan_prio	1'd1
40	pcnt_sig_ch1_in0	0	no	gpio_wlan_active	1'd1
41	pcnt_ctrl_ch0_in0	0	no	-	1'd1
42	pcnt_ctrl_ch1_in0	0	no	-	1'd1
43	pcnt_sig_ch0_in1	0	no	-	1'd1
44	pcnt_sig_ch1_in1	0	no	-	1'd1
45	pcnt_ctrl_ch0_in1	0	no	-	1'd1
46	pcnt_ctrl_ch1_in1	0	no	-	1'd1
47	pcnt_sig_ch0_in2	0	no	-	1'd1
48	pcnt_sig_ch1_in2	0	no	-	1'd1
49	pcnt_ctrl_ch0_in2	0	no	-	1'd1
50	pcnt_ctrl_ch1_in2	0	no	-	1'd1
51	pcnt_sig_ch0_in3	0	no	-	1'd1
52	pcnt_sig_ch1_in3	0	no	-	1'd1
53	pcnt_ctrl_ch0_in3	0	no	-	1'd1
54	pcnt_ctrl_ch1_in3	0	no	-	1'd1
64	usb_otg_iddig_in	0	no	-	1'd1
65	usb_otg_avalid_in	0	no	-	1'd1
66	usb_srp_bvalid_in	0	no	usb_otg_idpullup	1'd1
67	usb_otg_vbusvalid_in	0	no	usb_otg_dppulldown	1'd1
68	usb_srp_sessend_in	0	no	usb_otg_dmpulldown	1'd1
69	-	-	-	usb_otg_drvvbus	1'd1
70	-	-	-	usb_srp_chrgvbus	1'd1
71	-	-	-	usb_srp_dischrgvbus	1'd1
72	SPI3_CLK_in	0	no	SPI3_CLK_out_mux	SPI3_CLK_oe

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
73	SPI3_Q_in	0	no	SPI3_Q_out	SPI3_Q_oe
74	SPI3_D_in	0	no	SPI3_D_out	SPI3_D_oe
75	SPI3_HD_in	0	no	SPI3_HD_out	SPI3_HD_oe
76	SPI3_CS0_in	0	no	SPI3_CS0_out	SPI3_CS0_oe
77	-	-	-	SPI3_CS1_out	SPI3_CS1_oe
78	-	-	-	SPI3_CS2_out	SPI3_CS2_oe
79	-	-	-	ledc_ls_sig_out0	1’d1
80	-	-	-	ledc_ls_sig_out1	1’d1
81	-	-	-	ledc_ls_sig_out2	1’d1
82	-	-	-	ledc_ls_sig_out3	1’d1
83	rmt_sig_in0	0	no	ledc_ls_sig_out4	1’d1
84	rmt_sig_in1	0	no	ledc_ls_sig_out5	1’d1
85	rmt_sig_in2	0	no	ledc_ls_sig_out6	1’d1
86	rmt_sig_in3	0	no	ledc_ls_sig_out7	1’d1
87	-	-	-	rmt_sig_out0	1’d1
88	-	-	-	rmt_sig_out1	1’d1
89	-	-	-	rmt_sig_out2	1’d1
90	-	-	-	rmt_sig_out3	1’d1
95	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe
96	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe
100	-	-	-	gpio_sd0_out	1’d1
101	-	-	-	gpio_sd1_out	1’d1
102	-	-	-	gpio_sd2_out	1’d1
103	-	-	-	gpio_sd3_out	1’d1
104	-	-	-	gpio_sd4_out	1’d1
105	-	-	-	gpio_sd5_out	1’d1
106	-	-	-	gpio_sd6_out	1’d1
107	-	-	-	gpio_sd7_out	1’d1
108	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe
109	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe
110	FSPID_in	0	yes	FSPID_out	FSPID_oe
111	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe
112	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe
113	FSPIIO4_in	0	yes	FSPIIO4_out	FSPIIO4_oe
114	FSPIIO5_in	0	yes	FSPIIO5_out	FSPIIO5_oe
115	FSPIIO6_in	0	yes	FSPIIO6_out	FSPIIO6_oe
116	FSPIIO7_in	0	yes	FSPIIO7_out	FSPIIO7_oe
117	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe
118	-	-	-	FSPICS1_out	FSPICS1_oe
119	-	-	-	FSPICS2_out	FSPICS2_oe
120	-	-	-	FSPICS3_out	FSPICS3_oe

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
121	-	-	-	FSPICS4_out	FSPICS4_oe
122	-	-	-	FSPICS5_out	FSPICS5_oe
123	twai_rx	1	no	twai_tx	1'd1
124	-	-	-	twai_bus_off_on	1'd1
125	-	-	-	twai_clkout	1'd1
126	-	-	-	SUBSPICLK_out_mux	SUBSPICLK_oe
127	SUBSPIQ_in	0	yes	SUBSPIQ_out	SUBSPIQ_oe
128	SUBSPID_in	0	yes	SUBSPID_out	SUBSPID_oe
129	SUBSPIHD_in	0	yes	SUBSPIHD_out	SUBSPIHD_oe
130	SUBSPIWP_in	0	yes	SUBSPIWP_out	SUBSPIWP_oe
131	-	-	-	SUBSPICS0_out	SUBSPICS0_oe
132	-	-	-	SUBSPICS1_out	SUBSPICS1_oe
133	-	-	-	FSPIDQS_out	FSPIDQS_oe
134	-	-	-	FSPI_HSYNC_out	FSPI_HSYNC_oe
135	-	-	-	FSPI_VSYNC_out	FSPI_VSYNC_oe
136	-	-	-	FSPI_DE_out	FSPI_DE_oe
137	-	-	-	FSPICD_out	FSPICD_oe
139	-	-	-	SPI3_CD_out	SPI3_CD_oe
140	-	-	-	SPI3_DQS_out	SPI3_DQS_oe
143	I2S0I_DATA_in0	0	no	I2S0O_DATA_out0	1'd1
144	I2S0I_DATA_in1	0	no	I2S0O_DATA_out1	1'd1
145	I2S0I_DATA_in2	0	no	I2S0O_DATA_out2	1'd1
146	I2S0I_DATA_in3	0	no	I2S0O_DATA_out3	1'd1
147	I2S0I_DATA_in4	0	no	I2S0O_DATA_out4	1'd1
148	I2S0I_DATA_in5	0	no	I2S0O_DATA_out5	1'd1
149	I2S0I_DATA_in6	0	no	I2S0O_DATA_out6	1'd1
150	I2S0I_DATA_in7	0	no	I2S0O_DATA_out7	1'd1
151	I2S0I_DATA_in8	0	no	I2S0O_DATA_out8	1'd1
152	I2S0I_DATA_in9	0	no	I2S0O_DATA_out9	1'd1
153	I2S0I_DATA_in10	0	no	I2S0O_DATA_out10	1'd1
154	I2S0I_DATA_in11	0	no	I2S0O_DATA_out11	1'd1
155	I2S0I_DATA_in12	0	no	I2S0O_DATA_out12	1'd1
156	I2S0I_DATA_in13	0	no	I2S0O_DATA_out13	1'd1
157	I2S0I_DATA_in14	0	no	I2S0O_DATA_out14	1'd1
158	I2S0I_DATA_in15	0	no	I2S0O_DATA_out15	1'd1
159	-	-	-	I2S0O_DATA_out16	1'd1
160	-	-	-	I2S0O_DATA_out17	1'd1
161	-	-	-	I2S0O_DATA_out18	1'd1
162	-	-	-	I2S0O_DATA_out19	1'd1
163	-	-	-	I2S0O_DATA_out20	1'd1
164	-	-	-	I2S0O_DATA_out21	1'd1

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
165	-	-	-	I2S0O_DATA_out22	1'd1
166	-	-	-	I2S0O_DATA_out23	1'd1
167	SUBSPID4_in	0	yes	SUBSPID4_out	SUBSPID4_oe
168	SUBSPID5_in	0	yes	SUBSPID5_out	SUBSPID5_oe
169	SUBSPID6_in	0	yes	SUBSPID6_out	SUBSPID6_oe
170	SUBSPID7_in	0	yes	SUBSPID7_out	SUBSPID7_oe
171	SUBSPIDQS_in	0	yes	SUBSPIDQS_out	SUBSPIDQS_oe
193	I2S0I_H_SYNC	0	no	-	1'd1
194	I2S0I_V_SYNC	0	no	-	1'd1
195	I2S0I_H_ENABLE	0	no	-	1'd1
215	-	-	-	ant_sel0	1'd1
216	-	-	-	ant_sel1	1'd1
217	-	-	-	ant_sel2	1'd1
218	-	-	-	ant_sel3	1'd1
219	-	-	-	ant_sel4	1'd1
220	-	-	-	ant_sel5	1'd1
221	-	-	-	ant_sel6	1'd1
222	-	-	-	ant_sel7	1'd1
223	sig_in_func_223	0	no	sig_in_func223	1'd1
224	sig_in_func_224	0	no	sig_in_func224	1'd1
225	sig_in_func_225	0	no	sig_in_func225	1'd1
226	sig_in_func_226	0	no	sig_in_func226	1'd1
227	sig_in_func_227	0	no	sig_in_func227	1'd1
235	pro_alonegpio_in0	0	no	pro_alonegpio_out0	1'd1
236	pro_alonegpio_in1	0	no	pro_alonegpio_out1	1'd1
237	pro_alonegpio_in2	0	no	pro_alonegpio_out2	1'd1
238	pro_alonegpio_in3	0	no	pro_alonegpio_out3	1'd1
239	pro_alonegpio_in4	0	no	pro_alonegpio_out4	1'd1
240	pro_alonegpio_in5	0	no	pro_alonegpio_out5	1'd1
241	pro_alonegpio_in6	0	no	pro_alonegpio_out6	1'd1
242	pro_alonegpio_in7	0	no	pro_alonegpio_out7	1'd1
251	-	-	-	clk_i2s_mux	1'd1

5.11 IO MUX Pad List

Table 37 shows the IO MUX functions of each I/O pad:

Table 37: IO MUX Pad List

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
0	GPIO0	GPIO0	GPIO0	-	-	-	3	R
1	GPIO1	GPIO1	GPIO1	-	-	-	1	R
2	GPIO2	GPIO2	GPIO2	-	-	-	1	R

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
3	GPIO3	GPIO3	GPIO3	-	-	-	0	R
4	GPIO4	GPIO4	GPIO4	-	-	-	0	R
5	GPIO5	GPIO5	GPIO5	-	-	-	0	R
6	GPIO6	GPIO6	GPIO6	-	-	-	0	R
7	GPIO7	GPIO7	GPIO7	-	-	-	0	R
8	GPIO8	GPIO8	GPIO8	-	SUBSPICS1	-	0	R
9	GPIO9	GPIO9	GPIO9	-	SUBSPIHD	FSPIHD	1	R
10	GPIO10	GPIO10	GPIO10	FSPIIO4	SUBSPICS0	FSPICS0	1	R
11	GPIO11	GPIO11	GPIO11	FSPIIO5	SUBSPID	FSPID	1	R
12	GPIO12	GPIO12	GPIO12	FSPIIO6	SUBSPICLK	FSPICLK	1	R
13	GPIO13	GPIO13	GPIO13	FSPIIO7	SUBSPIQ	FSPIQ	1	R
14	GPIO14	GPIO14	GPIO14	FSPIDQS	SUBSPIWP	FSPIWP	1	R
15	XTAL_32K_P	XTAL_32K_P	GPIO15	U0RTS	-	-	0	R
16	XTAL_32K_N	XTAL_32K_N	GPIO16	U0CTS	-	-	0	R
17	DAC_1	DAC_1	GPIO17	U1TXD	-	-	1	R
18	DAC_2	DAC_2	GPIO18	U1RXD	CLK_OUT3	-	1	R
19	GPIO19	GPIO19	GPIO19	U1RTS	CLK_OUT2	-	0	R
20	GPIO20	GPIO20	GPIO20	U1CTS	CLK_OUT1	-	0	R
21	GPIO21	GPIO21	GPIO21	-	-	-	0	R
26	SPICS1	SPICS1	GPIO26	-	-	-	3	-
27	SPIHD	SPIHD	GPIO27	-	-	-	3	-
28	SPIWP	SPIWP	GPIO28	-	-	-	3	-
29	SPICS0	SPICS0	GPIO29	-	-	-	3	-
30	SPICLK	SPICLK	GPIO30	-	-	-	3	-
31	SPIQ	SPIQ	GPIO31	-	-	-	3	-
32	SPID	SPID	GPIO32	-		-	3	-
33	GPIO33	GPIO33	GPIO33	FSPIHD	SUBSPIHD	SPIIO4	1	-
34	GPIO34	GPIO34	GPIO34	FSPICS0	SUBSPICS0	SPIIO5	1	-
35	GPIO35	GPIO35	GPIO35	FSPID	SUBSPID	SPIIO6	1	-
36	GPIO36	GPIO36	GPIO36	FSPICLK	SUBSPICLK	SPIIO7	1	-
37	GPIO37	GPIO37	GPIO37	FSPIQ	SUBSPIQ	SPIDQS	1	-
38	GPIO38	GPIO38	GPIO38	FSPIWP	SUBSPIWP	-	1	-
39	MTCK	MTCK	GPIO39	CLK_OUT3	SUBSPICS1	-	1	-
40	MTDO	MTDO	GPIO40	CLK_OUT2	-	-	1	-
41	MTDI	MTDI	GPIO41	CLK_OUT1	-	-	1	-
42	MTMS	MTMS	GPIO42	-	-	-	1	-
43	U0TXD	U0TXD	GPIO43	CLK_OUT1	-	-	3	-
44	U0RXD	U0RXD	GPIO44	CLK_OUT2	-	-	3	-
45	GPIO45	GPIO45	GPIO45	-	-	-	2	-
46	GPIO46	GPIO46	GPIO46	-	-	-	2	I

Reset Configurations

"Reset" column shows the default configuration of each pad after reset:

- **0** - IE=0 (input disabled)
- **1** - IE=1 (input enabled)
- **2** - IE=1, WPD=1 (input enabled, pull-down resistor enabled)
- **3** - IE=1, WPU=1 (input enabled, pull-up resistor enabled)

Note:

- **R** - Pad has RTC/analog functions via RTC IO MUX.
- **I** - Pad can only be configured as input GPIO.

Please refer to Appendix A – ESP32-S2 Pin Lists in *ESP32-S2 Datasheet* for more details.

5.12 RTC IO MUX Pin List

Table 38 shows the RTC pins and how they correspond to GPIO pads.

Table 38: RTC IO MUX Pin Summary

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			
			1	2	3	4
0	0	TOUCH_PAD0*	RTC_GPIO0	-	-	sar_i2c_scl_0
1	1	TOUCH_PAD1	RTC_GPIO1	-	-	sar_i2c_sda_0
2	2	TOUCH_PAD2	RTC_GPIO2	-	-	sar_i2c_scl_1
3	3	TOUCH_PAD3	RTC_GPIO3	-	-	sar_i2c_sda_1
4	4	TOUCH_PAD4	RTC_GPIO4	-	-	-
5	5	TOUCH_PAD5	RTC_GPIO5	-	-	-
6	6	TOUCH_PAD6	RTC_GPIO6	-	-	-
7	7	TOUCH_PAD7	RTC_GPIO7	-	-	-
8	8	TOUCH_PAD8	RTC_GPIO8	-	-	-
9	9	TOUCH_PAD9	RTC_GPIO9	-	-	-
10	10	TOUCH_PAD10	RTC_GPIO10	-	-	-
11	11	TOUCH_PAD11	RTC_GPIO11	-	-	-
12	12	TOUCH_PAD12	RTC_GPIO12	-	-	-
13	13	TOUCH_PAD13	RTC_GPIO13	-	-	-
14	14	TOUCH_PAD14	RTC_GPIO14	-	-	-
15	15	X32P	RTC_GPIO15	-	-	-
16	16	X32N	RTC_GPIO16	-	-	-
17	17	PDAC1	RTC_GPIO17	-	-	-
18	18	PDAC2	RTC_GPIO18	-	-	-
19	19	RTC_PAD19	RTC_GPIO19	-	-	-
20	20	RTC_PAD20	RTC_GPIO20	-	-	-
21	21	RTC_PAD21	RTC_GPIO21	-	-	-

Note: TOUCH_PAD0 is an internal channel and its analog functions are not lead to a corresponding external GPIO.

5.13 Base Address

Users can access GPIO, IO MUX, GPIOSD, Dedicated GPIO, and RTCIO registers with one or two base address(es), which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 39: GPIO, IO MUX, GPIOSD, Dedicated GPIO, and RTCIO Base Addresses

Module	Access to Access Peripheral	Base Address
GPIO	PeriBUS1	0x3F404000
	PeriBUS2	0x60004000
IO MUX	PeriBUS1	0x3F409000
	PeriBUS2	0x60009000
GPIOSD	PeriBUS2	0x60004F00
Deicated GPIO	PeriBUS1	0x3F4CF000
RTCIO	PeriBUS1	0x3F408400
	PeriBUS2	0x60008400

5.14 Register Summary

The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the base address, please refer to Section 5.13.

5.14.1 GPIO Matrix Register Summary

Name	Description	Address	Access
GPIO Configuration Registers			
GPIO_BT_SELECT_REG	GPIO bit selection register	0x0000	R/W
GPIO_OUT_REG	GPIO0 ~ 31 output register	0x0004	R/W
GPIO_OUT_W1TS_REG	GPIO0 ~ 31 output bit set register	0x0008	WO
GPIO_OUT_W1TC_REG	GPIO0 ~ 31 output bit clear register	0x000C	WO
GPIO_OUT1_REG	GPIO32 ~ 53 output register	0x0010	R/W
GPIO_OUT1_W1TS_REG	GPIO32 ~ 53 output bit set register	0x0014	WO
GPIO_OUT1_W1TC_REG	GPIO32 ~ 53 output bit clear register	0x0018	WO
GPIO_SDIO_SELECT_REG	GPIO SDIO selection register	0x001C	R/W
GPIO_ENABLE_REG	GPIO0 ~ 31 output enable register	0x0020	R/W
GPIO_ENABLE_W1TS_REG	GPIO0 ~ 31 output enable bit set register	0x0024	WO
GPIO_ENABLE_W1TC_REG	GPIO0 ~ 31 output enable bit clear register	0x0028	WO
GPIO_ENABLE1_REG	GPIO32 ~ 53 output enable register	0x002C	R/W
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ 53 output enable bit set register	0x0030	WO
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ 53 output enable bit clear register	0x0034	WO
GPIO_STRAP_REG	Bootstrap pin value register	0x0038	RO
GPIO_IN_REG	GPIO0 ~ 31 input register	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ 53 input register	0x0040	RO
GPIO_PIN0_REG	Configuration for GPIO pin 0	0x0074	R/W
GPIO_PIN1_REG	Configuration for GPIO pin 1	0x0078	R/W

Name	Description	Address	Access
GPIO_PIN2_REG	Configuration for GPIO pin 2	0x007C	R/W
...	...	...	...
GPIO_PIN51_REG	Configuration for GPIO pin 51	0x0140	R/W
GPIO_PIN52_REG	Configuration for GPIO pin 52	0x0144	R/W
GPIO_PIN53_REG	Configuration for GPIO pin 53	0x0148	R/W
GPIO_FUNC0_IN_SEL_CFG_REG	Peripheral function 0 input selection register	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Peripheral function 1 input selection register	0x0158	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	Peripheral function 2 input selection register	0x015C	R/W
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	Peripheral function 253 input selection register	0x0548	R/W
GPIO_FUNC254_IN_SEL_CFG_REG	Peripheral function 254 input selection register	0x054C	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Peripheral function 255 input selection register	0x0550	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	Peripheral output selection for GPIO0	0x0554	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	Peripheral output selection for GPIO1	0x0558	R/W
GPIO_FUNC2_OUT_SEL_CFG_REG	Peripheral output selection for GPIO2	0x055C	R/W
..	...	...	...
GPIO_FUNC51_OUT_SEL_CFG_REG	Peripheral output selection for GPIO51	0x0620	R/W
GPIO_FUNC52_OUT_SEL_CFG_REG	Peripheral output selection for GPIO52	0x0624	R/W
GPIO_FUNC53_OUT_SEL_CFG_REG	Peripheral output selection for GPIO53	0x0628	R/W
GPIO_CLOCK_GATE_REG	GPIO clock gating register	0x062C	R/W
Interrupt Configuration Registers			
GPIO_STATUS_W1TS_REG	GPIO0 ~ 31 interrupt status bit set register	0x0048	WO
GPIO_STATUS_W1TC_REG	GPIO0 ~ 31 interrupt status bit clear register	0x004C	WO
GPIO_STATUS1_W1TS_REG	GPIO32 ~ 53 interrupt status bit set register	0x0054	WO
GPIO_STATUS1_W1TC_REG	GPIO32 ~ 53 interrupt status bit clear register	0x0058	WO
GPIO Interrupt Source Registers			
GPIO_STATUS_NEXT_REG	GPIO0 ~ 31 interrupt source register	0x014C	RO
GPIO_STATUS_NEXT1_REG	GPIO32 ~ 53 interrupt source register	0x0150	RO
Interrupt Status Registers			
GPIO_STATUS_REG	GPIO0 ~ 31 interrupt status register	0x0044	R/W
GPIO_STATUS1_REG	GPIO32 ~ 53 interrupt status register	0x0050	R/W
GPIO_PCPU_INT_REG	GPIO0 ~ 31 PRO_CPU interrupt status register	0x005C	RO
GPIO_PCPU_NMI_INT_REG	GPIO0 ~ 31 PRO_CPU non-maskable interrupt status register	0x0060	RO
GPIO_PCPU_INT1_REG	GPIO32 ~ 53 PRO_CPU interrupt status register	0x0068	RO
GPIO_PCPU_NMI_INT1_REG	GPIO32 ~ 53 PRO_CPU non-maskable interrupt status register	0x006C	RO

5.14.2 IO MUX Register Summary

Name	Description	Address	Access
IO_MUX_PIN_CTRL	Clock output configuration register	0x0000	R/W
IO_MUX_GPIO0_REG	Configuration register for pad GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	Configuration register for pad GPIO1	0x0008	R/W

Name	Description	Address	Access
IO_MUX_GPIO2_REG	Configuration register for pad GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	Configuration register for pad GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	Configuration register for pad GPIO4	0x0014	R/W
IO_MUX_GPIO5_REG	Configuration register for pad GPIO5	0x0018	R/W
IO_MUX_GPIO6_REG	Configuration register for pad GPIO6	0x001C	R/W
IO_MUX_GPIO7_REG	Configuration register for pad GPIO7	0x0020	R/W
IO_MUX_GPIO8_REG	Configuration register for pad GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	Configuration register for pad GPIO9	0x0028	R/W
IO_MUX_GPIO10_REG	Configuration register for pad GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	Configuration register for pad GPIO11	0x0030	R/W
IO_MUX_GPIO12_REG	Configuration register for pad GPIO12	0x0034	R/W
IO_MUX_GPIO13_REG	Configuration register for pad GPIO13	0x0038	R/W
IO_MUX_GPIO14_REG	Configuration register for pad GPIO14	0x003C	R/W
IO_MUX_GPIO15_REG	Configuration register for pad XTAL_32K_P	0x0040	R/W
IO_MUX_GPIO16_REG	Configuration register for pad XTAL_32K_N	0x0044	R/W
IO_MUX_GPIO17_REG	Configuration register for pad DAC_1	0x0048	R/W
IO_MUX_GPIO18_REG	Configuration register for pad DAC_2	0x004C	R/W
IO_MUX_GPIO19_REG	Configuration register for pad GPIO19	0x0050	R/W
IO_MUX_GPIO20_REG	Configuration register for pad GPIO20	0x0054	R/W
IO_MUX_GPIO21_REG	Configuration register for pad GPIO21	0x0058	R/W
IO_MUX_GPIO26_REG	Configuration register for pad SPICS1	0x006C	R/W
IO_MUX_GPIO27_REG	Configuration register for pad SPIHD	0x0070	R/W
IO_MUX_GPIO28_REG	Configuration register for pad SPIWP	0x0074	R/W
IO_MUX_GPIO29_REG	Configuration register for pad SPICS0	0x0078	R/W
IO_MUX_GPIO30_REG	Configuration register for pad SPICLK	0x007C	R/W
IO_MUX_GPIO31_REG	Configuration register for pad SPIQ	0x0080	R/W
IO_MUX_GPIO32_REG	Configuration register for pad SPID	0x0084	R/W
IO_MUX_GPIO33_REG	Configuration register for pad GPIO33	0x0088	R/W
IO_MUX_GPIO34_REG	Configuration register for pad GPIO34	0x008C	R/W
IO_MUX_GPIO35_REG	Configuration register for pad GPIO35	0x0090	R/W
IO_MUX_GPIO36_REG	Configuration register for pad GPIO36	0x0094	R/W
IO_MUX_GPIO37_REG	Configuration register for pad GPIO37	0x0098	R/W
IO_MUX_GPIO38_REG	Configuration register for pad GPIO38	0x009C	R/W
IO_MUX_GPIO39_REG	Configuration register for pad MTCK	0x00A0	R/W
IO_MUX_GPIO40_REG	Configuration register for pad MTDO	0x00A4	R/W
IO_MUX_GPIO41_REG	Configuration register for pad MTDI	0x00A8	R/W
IO_MUX_GPIO42_REG	Configuration register for pad MTMS	0x00AC	R/W
IO_MUX_GPIO43_REG	Configuration register for pad U0TXD	0x00B0	R/W
IO_MUX_GPIO44_REG	Configuration register for pad U0RXD	0x00B4	R/W
IO_MUX_GPIO45_REG	Configuration register for pad GPIO45	0x00B8	R/W
IO_MUX_GPIO46_REG	Configuration register for pad GPIO46	0x00BC	R/W

5.14.3 Sigma Delta Modulated Output Register Summary

Name	Description	Address	Access
Configuration Registers			
GPIOSD_SIGMADELTA0_REG	Duty Cycle Configure Register of SDM0	0x0000	R/W
GPIOSD_SIGMADELTA1_REG	Duty Cycle Configure Register of SDM1	0x0004	R/W
GPIOSD_SIGMADELTA2_REG	Duty Cycle Configure Register of SDM2	0x0008	R/W
GPIOSD_SIGMADELTA3_REG	Duty Cycle Configure Register of SDM3	0x000C	R/W
GPIOSD_SIGMADELTA4_REG	Duty Cycle Configure Register of SDM4	0x0010	R/W
GPIOSD_SIGMADELTA5_REG	Duty Cycle Configure Register of SDM5	0x0014	R/W
GPIOSD_SIGMADELTA6_REG	Duty Cycle Configure Register of SDM6	0x0018	R/W
GPIOSD_SIGMADELTA7_REG	Duty Cycle Configure Register of SDM7	0x001C	R/W
GPIOSD_SIGMADELTA_CG_REG	Clock Gating Configure Register	0x0020	R/W
GPIOSD_SIGMADELTA_MISC_REG	MISC Register	0x0024	R/W
GPIOSD_SIGMADELTA_VERSION_REG	Version Control Register	0x0028	R/W

5.14.4 Dedicated GPIO Register Summary

Name	Description	Address	Access
Configuration registers			
DEDIC_GPIO_OUT_DRT_REG	Dedicated GPIO direct output register	0x0000	WO
DEDIC_GPIO_OUT_MSK_REG	Dedicated GPIO mask output register	0x0004	WO
DEDIC_GPIO_OUT_IDV_REG	Dedicated GPIO individual output register	0x0008	WO
DEDIC_GPIO_OUT_CPU_REG	Dedicated GPIO output mode selection register	0x0010	R/W
DEDIC_GPIO_IN_DLY_REG	Dedicated GPIO input delay configuration register	0x0014	R/W
DEDIC_GPIO_INTR_RCGN_REG	Dedicated GPIO interrupts generation mode register	0x001C	R/W
Status registers			
DEDIC_GPIO_OUT_SCAN_REG	Dedicated GPIO output status register	0x000C	RO
DEDIC_GPIO_IN_SCAN_REG	Dedicated GPIO input status register	0x0018	RO
Interrupt registers			
DEDIC_GPIO_INTR_RAW_REG	Raw interrupt status	0x0020	RO
DEDIC_GPIO_INTR_RLS_REG	Interrupt enable bits	0x0024	R/W
DEDIC_GPIO_INTR_ST_REG	Masked interrupt status	0x0028	RO
DEDIC_GPIO_INTR_CLR_REG	Interrupt clear bits	0x002C	WO

5.14.5 RTC IO MUX Register Summary

Name	Description	Address	Access
GPIO Configuration and Data Registers			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO output register	0x0000	R/W
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register	0x0004	WO
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register	0x0008	WO
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO output enable register	0x000C	R/W
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit set register	0x0010	WO
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register	0x0014	WO
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register	0x0018	R/W
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register	0x001C	WO

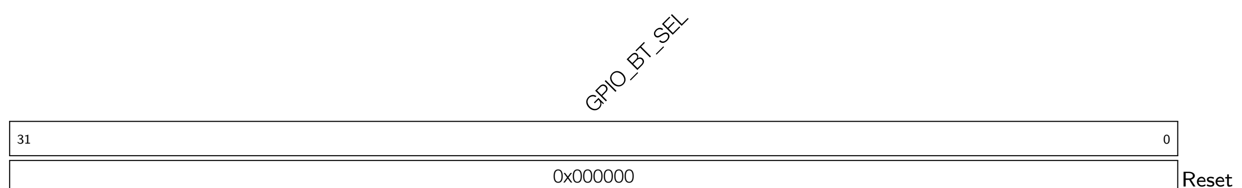
Name	Description	Address	Access
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register	0x0020	WO
RTCIO_RTC_GPIO_IN_REG	RTC GPIO input register	0x0024	RO
RTCIO_RTC_GPIO_PIN0_REG	RTC configuration for pin 0	0x0028	R/W
RTCIO_RTC_GPIO_PIN1_REG	RTC configuration for pin 1	0x002C	R/W
RTCIO_RTC_GPIO_PIN2_REG	RTC configuration for pin 2	0x0030	R/W
RTCIO_RTC_GPIO_PIN3_REG	RTC configuration for pin 3	0x0034	R/W
...	...	...	...
RTCIO_RTC_GPIO_PIN19_REG	RTC configuration for pin 19	0x0074	R/W
RTCIO_RTC_GPIO_PIN20_REG	RTC configuration for pin 20	0x0078	R/W
RTCIO_RTC_GPIO_PIN21_REG	RTC configuration for pin 21	0x007C	R/W
GPIO RTC Function Configuration Registers			
RTCIO_TOUCH_PAD0_REG	Touch pad 0 configuration register	0x0084	R/W
RTCIO_TOUCH_PAD1_REG	Touch pad 1 configuration register	0x0088	R/W
RTCIO_TOUCH_PAD2_REG	Touch pad 2 configuration register	0x008C	R/W
...	...	...	...
RTCIO_TOUCH_PAD13_REG	Touch pad 13 configuration register	0x00B8	R/W
RTCIO_TOUCH_PAD14_REG	Touch pad 14 configuration register	0x00BC	R/W
RTCIO_XTAL_32P_PAD_REG	32KHz crystal P-pad configuration register	0x00C0	R/W
RTCIO_XTAL_32N_PAD_REG	32KHz crystal N-pad configuration register	0x00C4	R/W
RTCIO_PAD_DAC1_REG	DAC1 configuration register	0x00C8	R/W
RTCIO_PAD_DAC2_REG	DAC2 configuration register	0x00CC	R/W
RTCIO_RTC_PAD19_REG	Touch pad 19 configuration register	0x00D0	R/W
RTCIO_RTC_PAD20_REG	Touch pad 20 configuration register	0x00D4	R/W
RTCIO_RTC_PAD21_REG	Touch pad 21 configuration register	0x00D8	R/W
RTCIO_XTL_EXT_CTR_REG	Crystal power down enable GPIO source	0x00E0	R/W
RTCIO_SAR_I2C_IO_REG	RTC I2C pad selection	0x00E4	R/W

5.15 Registers

The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the base address, please refer to Section 5.13.

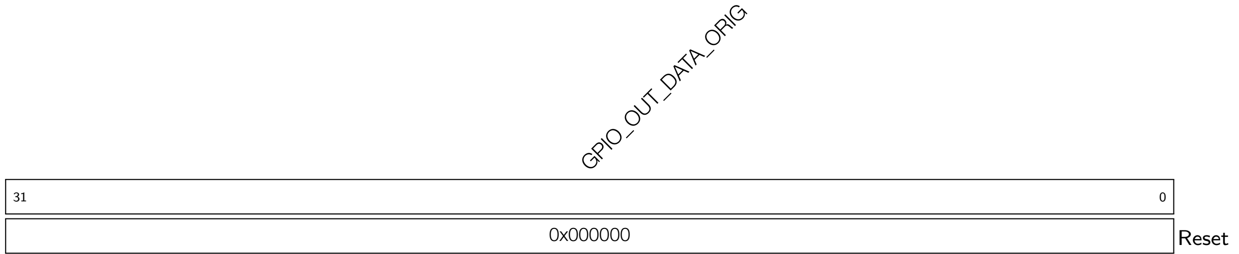
5.15.1 GPIO Matrix Registers

Register 5.1: GPIO_BT_SELECT_REG (0x0000)



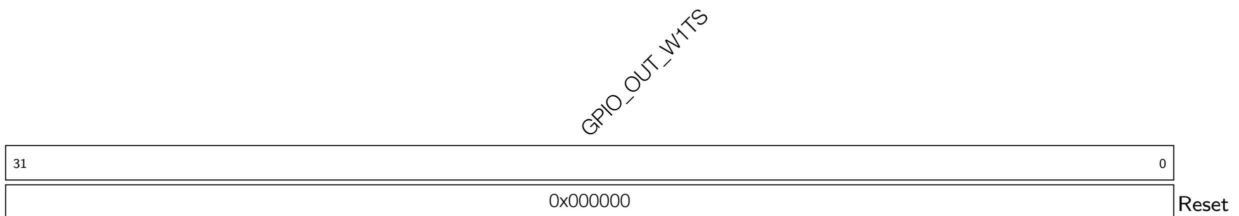
GPIO_BT_SEL Reserved (R/W)

Register 5.2: GPIO_OUT_REG (0x0004)



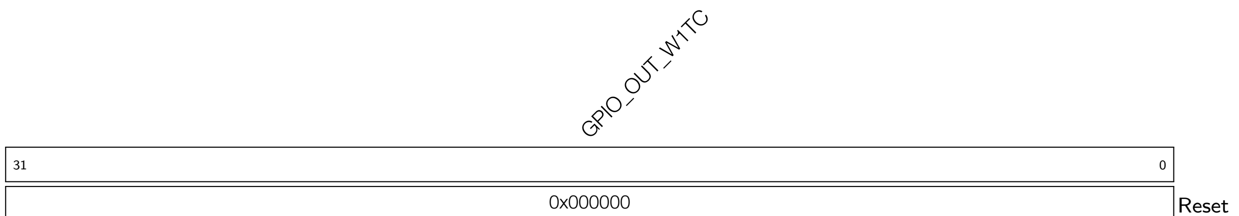
GPIO_OUT_DATA_ORIG GPIO0 ~ 31 output value in simple GPIO output mode. The values of bit0 ~ bit31 correspond to the output value of GPIO0 ~ GPIO31 respectively. Bit22 ~ bit25 are invalid. (R/W)

Register 5.3: GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS GPIO0 ~ 31 output set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT_REG. (WO)

Register 5.4: GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC GPIO0 ~ 31 output clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT_REG. (WO)

Register 5.5: GPIO_OUT1_REG (0x0010)

(reserved)	GPIO_OUT1_DATA_ORIG	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0x0000	Reset

GPIO_OUT1_DATA_ORIG GPIO32 ~ 53 output value in simple GPIO output mode. The values of bit0 ~ bit13 correspond to GPIO32 ~ GPIO45. Bit14 ~ bit21 are invalid. (R/W)

Register 5.6: GPIO_OUT1_W1TS_REG (0x0014)

(reserved)	GPIO_OUT1_W1TS	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0x0000	Reset

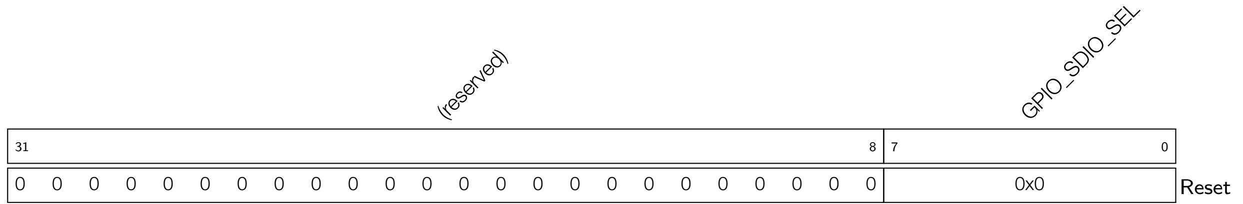
GPIO_OUT1_W1TS GPIO32 ~ 53 output value set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT1_REG. (WO)

Register 5.7: GPIO_OUT1_W1TC_REG (0x0018)

(reserved)	GPIO_OUT1_W1TC	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0x0000	Reset

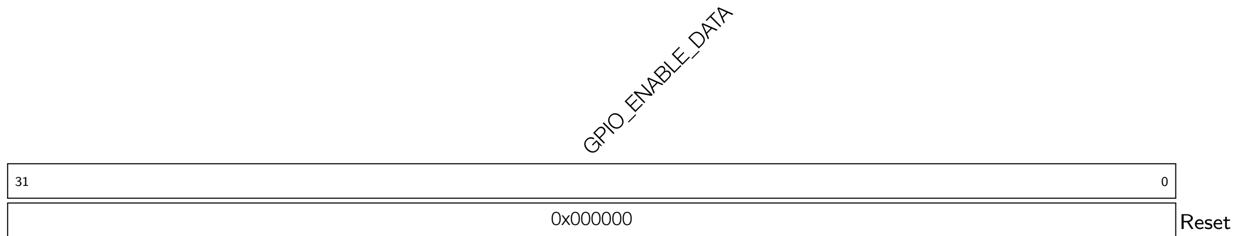
GPIO_OUT1_W1TC GPIO32 ~ 53 output value clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT1_REG. (WO)

Register 5.8: GPIO_SDIO_SELECT_REG (0x001C)



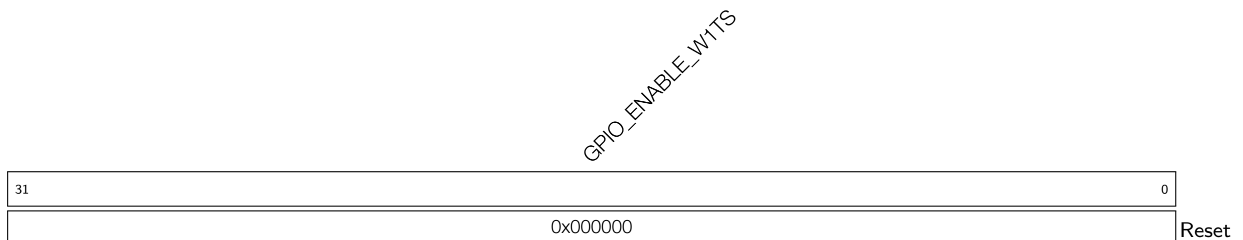
GPIO_SDIO_SEL Reserved (R/W)

Register 5.9: GPIO_ENABLE_REG (0x0020)



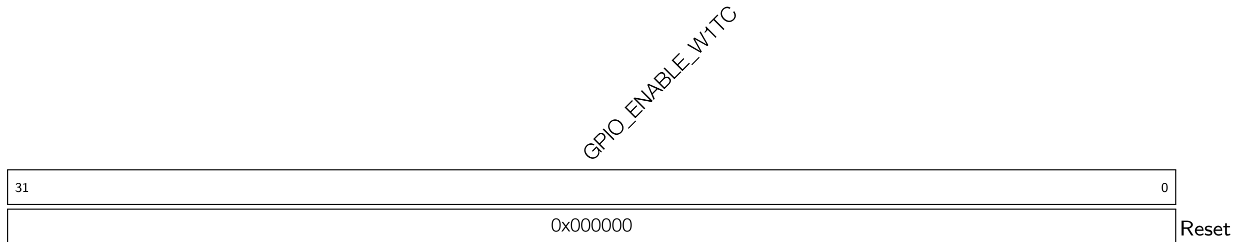
GPIO_ENABLE_DATA GPIO0 ~ 31 output enable register. (R/W)

Register 5.10: GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS GPIO0 ~ 31 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE_REG. (WO)

Register 5.11: GPIO_ENABLE_W1TC_REG (0x0028)

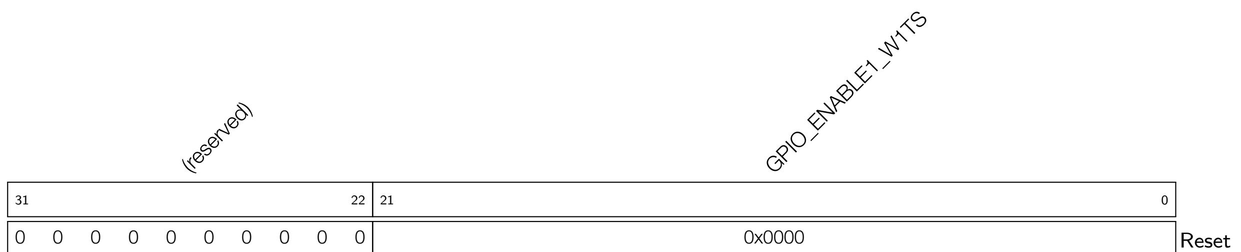


GPIO_ENABLE_W1TC GPIO0 ~ 31 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE_REG. (WO)

Register 5.12: GPIO_ENABLE1_REG (0x002C)

GPIO_ENABLE1_DATA GPIO32 ~ 53 output enable register. (R/W)

Register 5.13: GPIO_ENABLE1_W1TS_REG (0x0030)



GPIO_ENABLE1_W1TS GPIO32 ~ 53 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE1_REG. (WO)

Register 5.14: GPIO_ENABLE1_W1TC_REG (0x0034)

(reserved)	GPIO_ENABLE1_W1TC	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0x0000	Reset

GPIO_ENABLE1_W1TC GPIO32 ~ 53 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE1_REG. (WO)

Register 5.15: GPIO_STRAP_REG (0x0038)

(reserved)	GPIO_STRAPPING	
31 ... 16	15 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

GPIO_STRAPPING GPIO strapping values: bit4 ~ bit2 correspond to stripping pins GPIO45, GPIO0, and GPIO46 respectively. (RO)

Register 5.16: GPIO_IN_REG (0x003C)

GPIO_IN_DATA_NEXT	
31 ... 0	
0	Reset

GPIO_IN_DATA_NEXT GPIO0 ~ 31 input value. Each bit represents a pad input value, 1 for high level and 0 for low level. (RO)

Register 5.17: GPIO_IN1_REG (0x0040)

(reserved)	GPIO_IN_DATA1_NEXT	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0	Reset

GPIO_IN_DATA1_NEXT GPIO32 ~ 53 input value. Each bit represents a pad input value. (RO)

Register 5.18: GPIO_PIN*n*_REG (*n*: 0-53) (0x0074+4n*)**

(reserved)	GPIO_PIN*n*_INT_ENA	GPIO_PIN*n*_CONFIG	GPIO_PIN*n*_WAKEUP_ENABLE	GPIO_PIN*n*_INT_TYPE	(reserved)	GPIO_PIN*n*_SYNC1_BYPASS	GPIO_PIN*n*_PAD_DRIVER	GPIO_PIN*n*_SYNC2_BYPASS	
31 ... 18	17 ... 13	12 ... 11	10	9 ... 7	6 ... 5	4 ... 3	2	1 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0	0x0	0 0	0x0	0	0x0	Reset

GPIO_PIN*n*_SYNC2_BYPASS For the second stage synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

GPIO_PIN*n*_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain output. (R/W)

GPIO_PIN*n*_SYNC1_BYPASS For the first stage synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

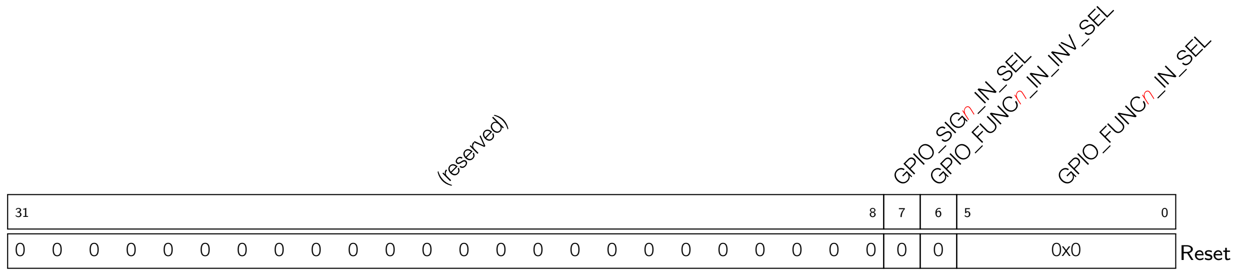
GPIO_PIN*n*_INT_TYPE Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable bit, only wakes up the CPU from Light-sleep (R/W)

GPIO_PIN*n*_CONFIG Reserved (R/W)

GPIO_PIN*n*_INT_ENA Interrupt enable bits. bit13: CPU interrupt enabled; bit14: CPU non-maskable interrupt enabled. (R/W)

Register 5.19: GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-255) (0x0154+4n*)**

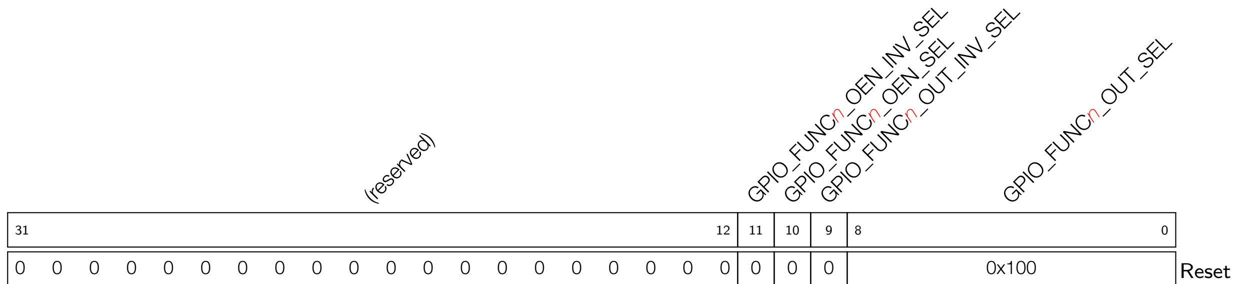


GPIO_FUNC*n*_IN_SEL Selection control for peripheral input signal *m*, selects a pad from the 54 GPIO matrix pads to connect this input signal. Or selects 0x38 for a constantly high input or 0x3C for a constantly low input. (R/W)

GPIO_FUNC*n*_IN_INV_SEL Invert the input value. 1: invert enabled; 0: invert disabled. (R/W)

GPIO_SIG*n*_IN_SEL Bypass GPIO matrix. 1: route signals via GPIO matrix, 0: connect signals directly to peripheral configured in IO_MUX. (R/W)

Register 5.20: GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-53) (0x0554+4n*)**



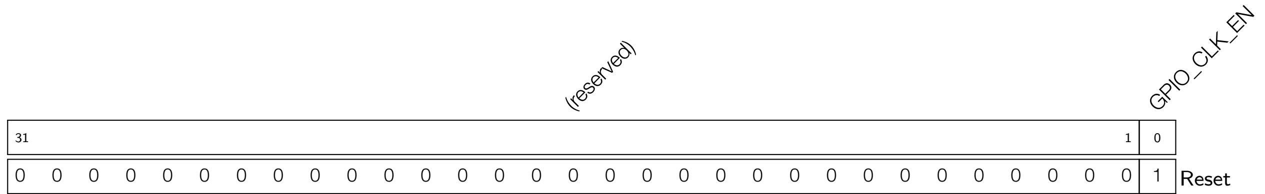
GPIO_FUNC*n*_OUT_SEL Selection control for GPIO output *n*. If a value *s* (0<=*s*<256) is written to this field, the peripheral output signal *s* will be connected to GPIO output *n*. If a value 256 is written to this field, bit *n* of GPIO_OUT_REG/GPIO_OUT1_REG and GPIO_ENABLE_REG/GPIO_ENABLE1_REG will be selected as the output value and output enable. (R/W)

GPIO_FUNC*n*_OUT_INV_SEL 0: Do not invert the output value; 1: Invert the output value. (R/W)

GPIO_FUNC*n*_OEN_SEL 0: Use output enable signal from peripheral; 1: Force the output enable signal to be sourced from bit *n* of GPIO_ENABLE_REG. (R/W)

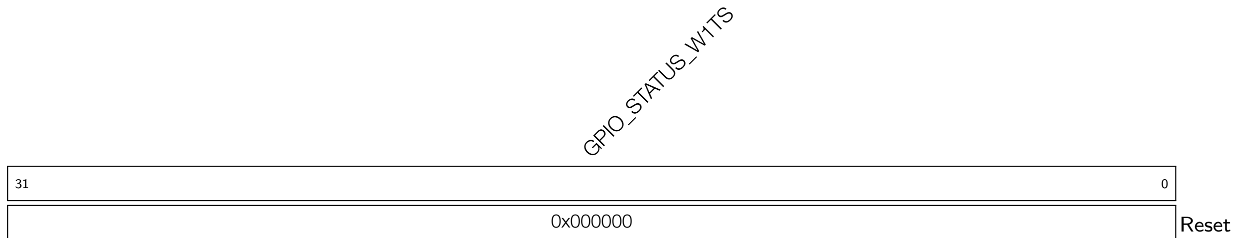
GPIO_FUNC*n*_OEN_INV_SEL 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

Register 5.21: GPIO_CLOCK_GATE_REG (0x062C)



GPIO_CLK_EN Clock gating enable bit. If set to 1, the clock is free running. (R/W)

Register 5.22: GPIO_STATUS_W1TS_REG (0x0048)

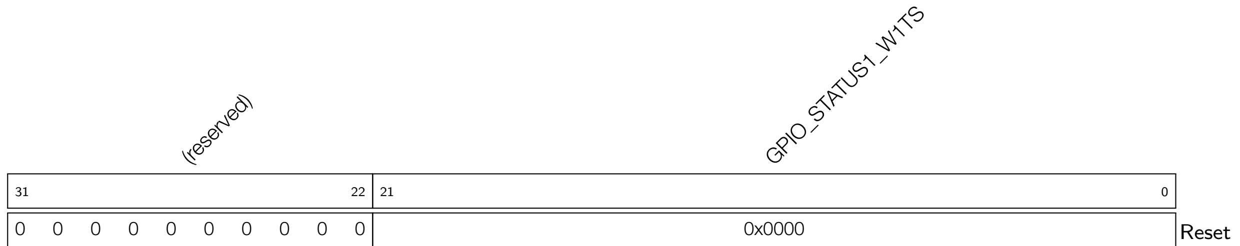


GPIO_STATUS_W1TS GPIO0 ~ 31 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1. Recommended operation: use this register to set GPIO_STATUS_INTERRUPT. (WO)

Register 5.23: GPIO_STATUS_W1TC_REG (0x004C)

GPIO_STATUS_W1TC GPIO0 ~ 31 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT. (WO)

Register 5.24: GPIO_STATUS1_W1TS_REG (0x0054)

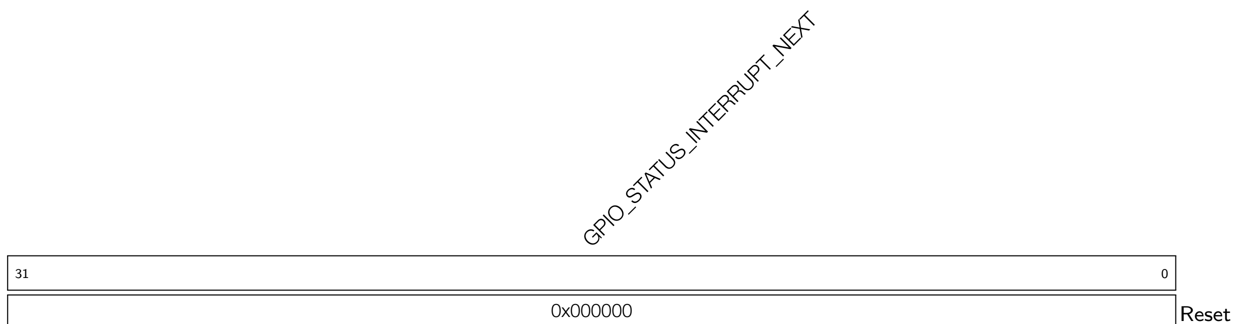


GPIO_STATUS1_W1TS GPIO32 ~ 53 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be set to 1. Recommended operation: use this register to set GPIO_STATUS1_REG. (WO)

Register 5.25: GPIO_STATUS1_W1TC_REG (0x0058)

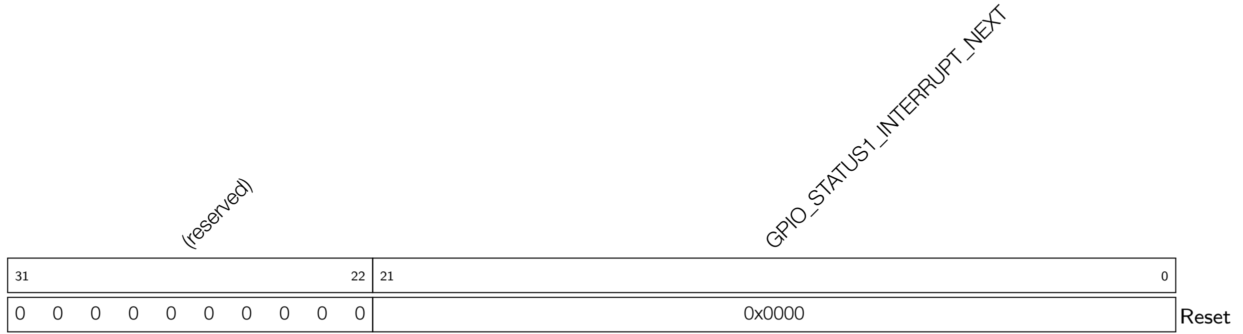
GPIO_STATUS1_W1TC GPIO32 ~ 53 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be cleared. Recommended operation: use this register to clear GPIO_STATUS1_REG. (WO)

Register 5.26: GPIO_STATUS_NEXT_REG (0x014C)



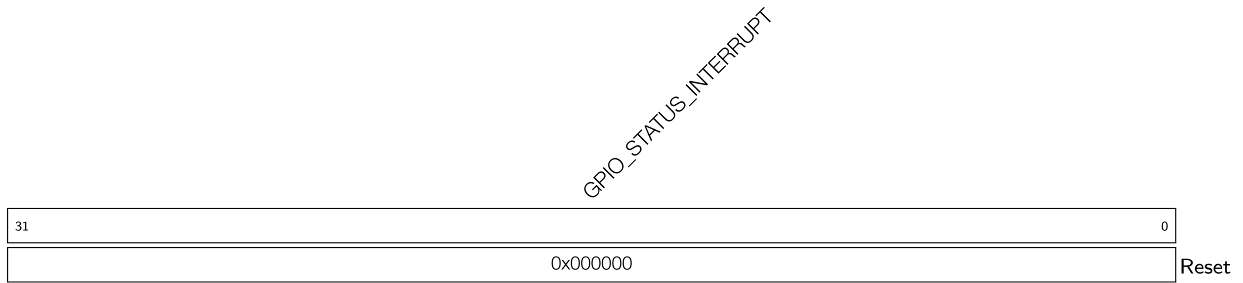
GPIO_STATUS_INTERRUPT_NEXT Interrupt source signal of GPIO0 ~ 31, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. (RO)

Register 5.27: GPIO_STATUS_NEXT1_REG (0x0150)



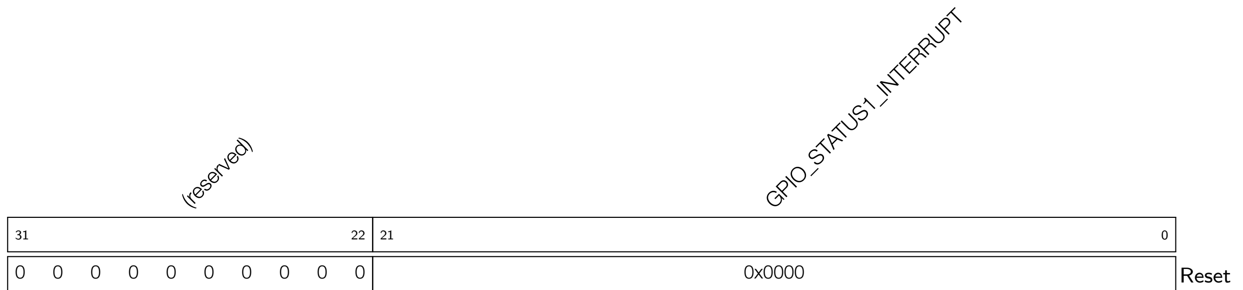
GPIO_STATUS1_INTERRUPT_NEXT Interrupt source signal of GPIO32 ~ 53. (RO)

Register 5.28: GPIO_STATUS_REG (0x0044)



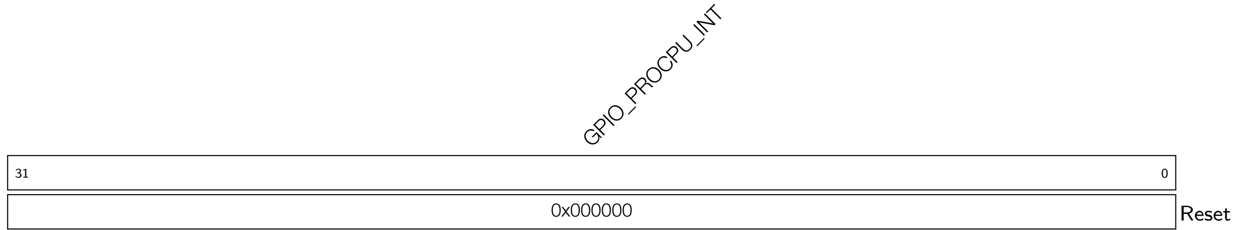
GPIO_STATUS_INTERRUPT GPIO0 ~ 31 interrupt status register. (R/W)

Register 5.29: GPIO_STATUS1_REG (0x0050)



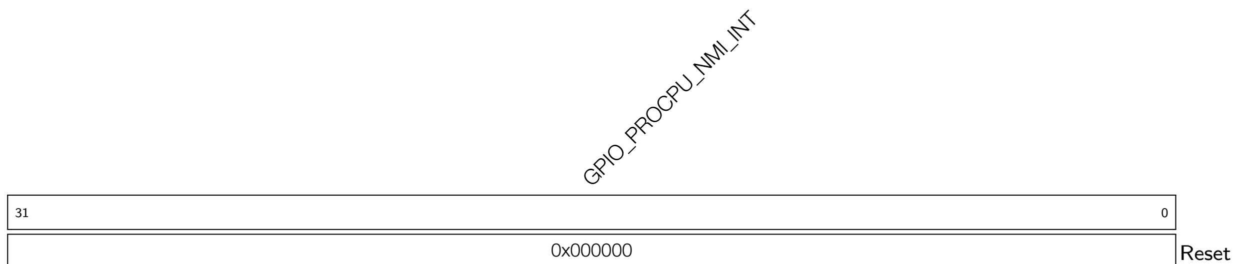
GPIO_STATUS1_INTERRUPT GPIO32 ~ 53 interrupt status register. (R/W)

Register 5.30: GPIO_PCPU_INT_REG (0x005C)



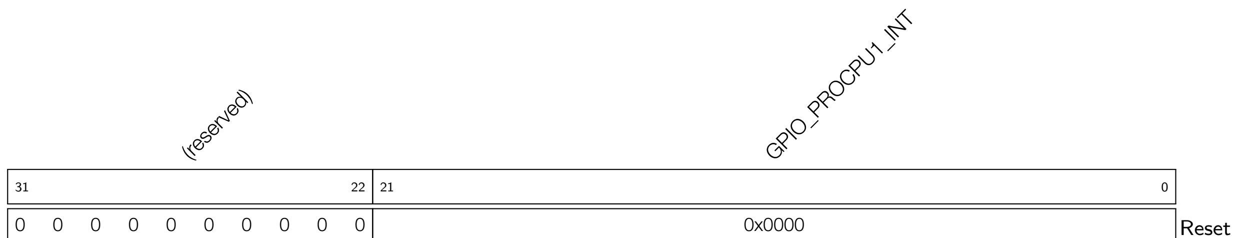
GPIO_PROCPU_INT GPIO0 ~ 31 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 5.31: GPIO_PCPU_NMI_INT_REG (0x0060)



GPIO_PROCPU_NMI_INT GPIO0 ~ 31 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

Register 5.32: GPIO_PCPU_INT1_REG (0x0068)



GPIO_PROCPU1_INT GPIO32 ~ 53 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 13 of GPIO_PIN*n*_REG). (RO)

Register 5.33: GPIO_PCPU_NMI_INT1_REG (0x006C)

(reserved)	GPIO_PROCPU_NMI1_INT	
31 22	21 0	
0 0 0 0 0 0 0 0 0 0	0x0000	Reset

GPIO_PROCPU_NMI1_INT GPIO32 ~ 53 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

5.15.2 IO MUX Registers

Register 5.34: IO_MUX_PIN_CTRL (0x0000)

(reserved)	IO_MUX_PAD_POWER_CTRL	IO_MUX_SWITCH_PRT_NUM	IO_MUX_PIN_CTRL_CLK3	IO_MUX_PIN_CTRL_CLK2	IO_MUX_PIN_CTRL_CLK1	
31 16	15	14 12	11 8	7 4	3 0	
0x0	0x0	0x2	0x0	0x0	0x0	Reset

IO_MUX_PIN_CTRL_CLK*x* If you want to output clock for I2S0 to:

CLK_OUT1, then set IO_MUX_PIN_CTRL_CLK1 = 0x0

CLK_OUT2, then set IO_MUX_PIN_CTRL_CLK2 = 0x0;

CLK_OUT3, then set IO_MUX_PIN_CTRL_CLK3 = 0x0.

Note:

Only the above mentioned combinations of clock source and clock output pins are possible.

The CLK_OUT1 ~ 3 can be found in IO_MUX Pad List.

IO_MUX_SWITCH_PRT_NUM IO pad power switch delay, delay unit is one APB clock.

IO_MUX_PAD_POWER_CTRL Select power voltage for GPIO33 ~ 37. 1: select VDD_SPI 1.8 V; 0: select VDD3P3_CPU 3.3 V.

Register 5.35: IO_MUX_*n*_REG (*n*: GPIO0-GPIO21, GPIO26-GPIO46) (0x0010+4*n)

Bits	31–16	15	14–12	11–10	9	8	7	6–5	4	3	2	1	0
Field	(reserved)	IO_MUX_FILTER_EN	IO_MUX_MCU_SEL	IO_MUX_FUN_DRV	IO_MUX_FUN_IE	IO_MUX_FUN_WPU	IO_MUX_FUN_WPD	(reserved)	IO_MUX_MCU_IE	IO_MUX_MCU_WPU	IO_MUX_MCU_WPD	IO_MUX_SLP_SEL	IO_MUX_MCU_OE
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x2	0	0	0	00	0	0	0	0	0

IO_MUX_MCU_OE Output enable of the pad in sleep mode. 1: Output enabled; 0: Output disabled. (R/W)

IO_MUX_SLP_SEL Sleep mode selection of this pad. Set to 1 to put the pad in sleep mode. (R/W)

IO_MUX_MCU_WPD Pull-down enable of the pad during sleep mode. 1: Internal pull-down enabled; 0: Internal pull-down disabled. (R/W)

IO_MUX_MCU_WPU Pull-up enable of the pad during sleep mode. 1: Internal pull-up enabled; 0: Internal pull-up disabled.

IO_MUX_MCU_IE Input enable of the pad during sleep mode. 1: Input enabled; 0: Input disabled. (R/W)

IO_MUX_FUN_WPD Pull-down enable of the pad. 1: Pull-down enabled; 0: Pull-down disabled. (R/W)

IO_MUX_FUN_WPU Pull-up enable of the pad. 1: Internal pull-up enabled; 0: Internal pull-up disabled. (R/W)

IO_MUX_FUN_IE Input enable of the pad. 1: Input enabled; 0: Input disabled. (R/W)

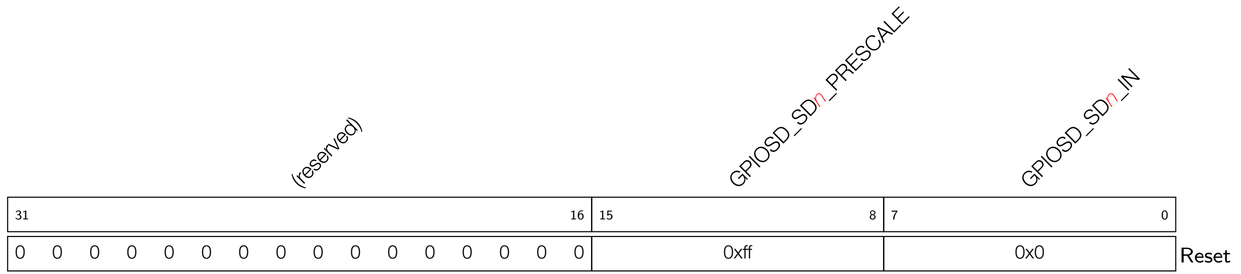
IO_MUX_FUN_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

IO_MUX_MCU_SEL Select IO MUX function for this signal. 0: Select Function 0; 1: Select Function 1, etc. (R/W)

IO_MUX_FILTER_EN Enable filter for pin input signals. 1: Filter enabled; 2: Filter disabled. (R/W)

5.15.3 Sigma Delta Modulated Output Registers

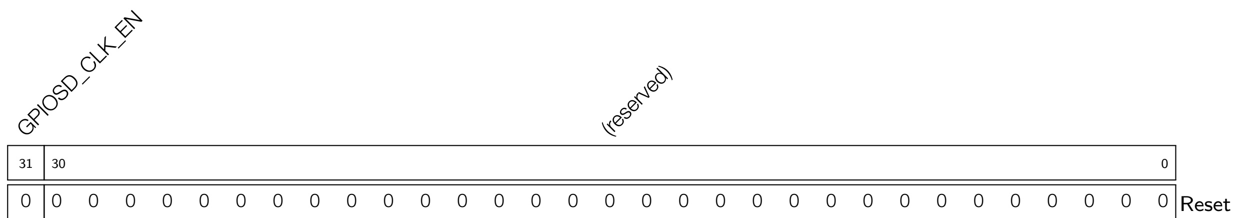
Register 5.36: GPIOSD_SIGMADELTA*n*_REG (*n*: 0-7) (0x0000+4*n)



GPIOSD_SD*n*_IN This field is used to configure the duty cycle of sigma delta modulation output. (R/W)

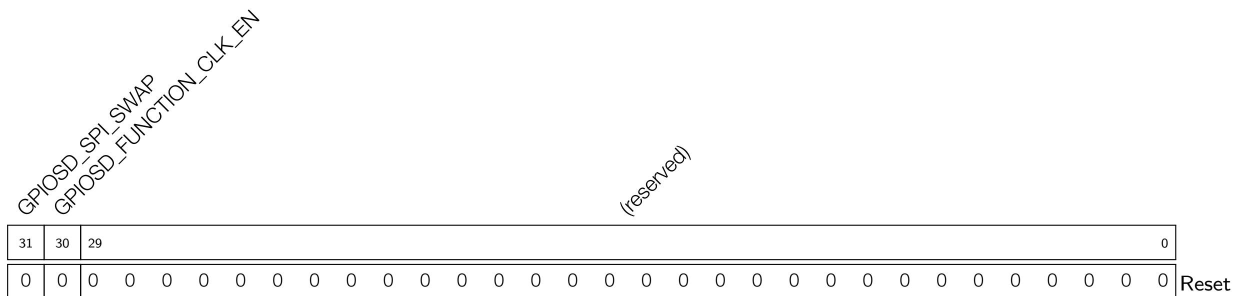
GPIOSD_SD*n*_PRESCALE This field is used to set a divider value to divide APB clock. (R/W)

Register 5.37: GPIOSD_SIGMADELTA_CG_REG (0x0020)



GPIOSD_CLK_EN Clock enable bit of configuration registers for sigma delta modulation. (R/W)

Register 5.38: GPIOSD_SIGMADELTA_MISC_REG (0x0024)



GPIOSD_FUNCTION_CLK_EN Clock enable bit of sigma delta modulation. (R/W)

GPIOSD_SPI_SWAP Reserved. (R/W)

Register 5.39: GPIOSD_SIGMADELTA_VERSION_REG (0x0028)

(reserved)	GPIOSD_GPIO_SD_DATE	
31 28	27 0	
0 0 0 0	0x1802260	Reset

GPIOSD_GPIO_SD_DATE Version control register. (R/W)

5.15.4 Dedicated GPIO Registers

Register 5.40: DEDIC_GPIO_OUT_DRT_REG (0x0000)

(reserved)	DEDIC_GPIO_OUT_DRT_VLAUE	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

DEDIC_GPIO_OUT_DRT_VLAUE This register is used to configure direct output value of 8-channel dedicated GPIO. (WO)

Register 5.41: DEDIC_GPIO_OUT_MSK_REG (0x0004)

(reserved)	DEDIC_GPIO_OUT_MSK	DEDIC_GPIO_OUT_VALUE	
31 16	15 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	Reset

DEDIC_GPIO_OUT_VALUE This register is used to configure updated output value of 8-channel dedicated GPIO. (WO)

DEDIC_GPIO_OUT_MSK This register is used to configure channels which would be updated. 1: corresponding channel's output would be updated. (WO)

Register 5.42: DEDIC_GPIO_OUT_IDV_REG (0x0008)

(reserved)	DEDIC_GPIO_OUT_IDV_CH7	DEDIC_GPIO_OUT_IDV_CH6	DEDIC_GPIO_OUT_IDV_CH5	DEDIC_GPIO_OUT_IDV_CH4	DEDIC_GPIO_OUT_IDV_CH3	DEDIC_GPIO_OUT_IDV_CH2	DEDIC_GPIO_OUT_IDV_CH1	DEDIC_GPIO_OUT_IDV_CH0	
31 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

DEDIC_GPIO_OUT_IDV_CH0 Configure channel 0 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH1 Configure channel 1 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH2 Configure channel 2 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH3 Configure channel 3 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH4 Configure channel 4 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH5 Configure channel 5 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH6 Configure channel 6 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH7 Configure channel 7 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

Register 5.43: DEDIC_GPIO_OUT_CPU_REG (0x0010)

(reserved)	DEDIC_GPIO_OUT_CPU_SEL7	DEDIC_GPIO_OUT_CPU_SEL6	DEDIC_GPIO_OUT_CPU_SEL5	DEDIC_GPIO_OUT_CPU_SEL4	DEDIC_GPIO_OUT_CPU_SEL3	DEDIC_GPIO_OUT_CPU_SEL2	DEDIC_GPIO_OUT_CPU_SEL1	DEDIC_GPIO_OUT_CPU_SEL0	
31 ... 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

DEDIC_GPIO_OUT_CPU_SEL0 Select GPIO out value configured by registers or CPU instructions for channel 0. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL1 Select GPIO out value configured by registers or CPU instructions for channel 1. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL2 Select GPIO out value configured by registers or CPU instructions for channel 2. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL3 Select GPIO out value configured by registers or CPU instructions for channel 3. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL4 Select GPIO out value configured by registers or CPU instructions for channel 4. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL5 Select GPIO out value configured by registers or CPU instructions for channel 5. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL6 Select GPIO out value configured by registers or CPU instructions for channel 6. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL7 Select GPIO out value configured by registers or CPU instructions for channel 7. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

Register 5.44: DEDIC_GPIO_IN_DLY_REG (0x0014)

(reserved)	DEDIC_GPIO_IN_DLY_CH7	DEDIC_GPIO_IN_DLY_CH6	DEDIC_GPIO_IN_DLY_CH5	DEDIC_GPIO_IN_DLY_CH4	DEDIC_GPIO_IN_DLY_CH3	DEDIC_GPIO_IN_DLY_CH2	DEDIC_GPIO_IN_DLY_CH1	DEDIC_GPIO_IN_DLY_CH0	
31 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

DEDIC_GPIO_IN_DLY_CH0 Configure GPIO0 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH1 Configure GPIO1 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH2 Configure GPIO2 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH3 Configure GPIO3 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH4 Configure GPIO4 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH5 Configure GPIO5 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH6 Configure GPIO6 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH7 Configure GPIO7 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

Register 5.45: DEDIC_GPIO_INTR_RCGN_REG (0x001C)

(reserved)	DEDIC_GPIO_INTR_MODE_CH7	DEDIC_GPIO_INTR_MODE_CH6	DEDIC_GPIO_INTR_MODE_CH5	DEDIC_GPIO_INTR_MODE_CH4	DEDIC_GPIO_INTR_MODE_CH3	DEDIC_GPIO_INTR_MODE_CH2	DEDIC_GPIO_INTR_MODE_CH1	DEDIC_GPIO_INTR_MODE_CH0	
31 24	23 21	20 18	17 15	14 12	11 9	8 6	5 3	2 0	
0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

DEDIC_GPIO_INTR_MODE_CH0 Configure channel 0 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH1 Configure channel 1 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH2 Configure channel 2 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH3 Configure channel 3 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

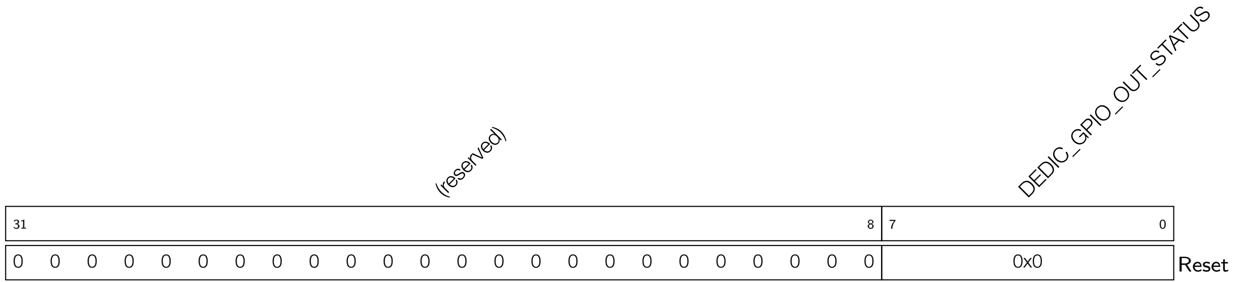
DEDIC_GPIO_INTR_MODE_CH4 Configure channel 4 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH5 Configure channel 5 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH6 Configure channel 6 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

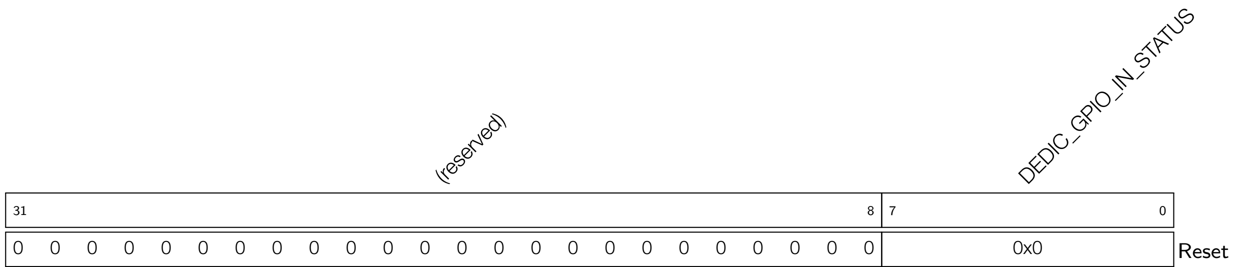
DEDIC_GPIO_INTR_MODE_CH7 Configure channel 7 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

Register 5.46: DEDIC_GPIO_OUT_SCAN_REG (0x000C)



DEDIC_GPIO_OUT_STATUS GPIO output value configured by DEDIC_GPIO_OUT_DRT_REG, DEDIC_GPIO_OUT_MSK_REG, and DEDIC_GPIO_OUT_IDV_REG. (RO)

Register 5.47: DEDIC_GPIO_IN_SCAN_REG (0x0018)



DEDIC_GPIO_IN_STATUS GPIO input value after configured by DEDIC_GPIO_IN_DLY_REG. (RO)

Register 5.48: DEDIC_GPIO_INTR_RAW_REG (0x0020)

(reserved)	DEDIC_GPIO7_INT_RAW	DEDIC_GPIO6_INT_RAW	DEDIC_GPIO5_INT_RAW	DEDIC_GPIO4_INT_RAW	DEDIC_GPIO3_INT_RAW	DEDIC_GPIO2_INT_RAW	DEDIC_GPIO1_INT_RAW	DEDIC_GPIO0_INT_RAW	
31 … 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

DEDIC_GPIO0_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO0 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO1_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO1 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO2_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO2 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO3_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO3 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

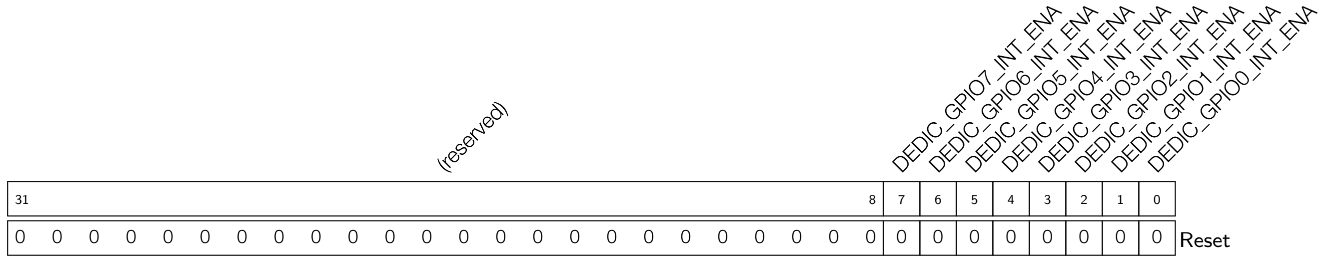
DEDIC_GPIO4_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO4 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO5_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO5 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO6_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO6 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO7_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO7 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

Register 5.49: DEDIC_GPIO_INTR_RLS_REG (0x0024)



DEDIC_GPIO0_INT_ENA The enable bit for DEDIC_GPIO0_INT_ST register. (R/W)

DEDIC_GPIO1_INT_ENA The enable bit for DEDIC_GPIO1_INT_ST register. (R/W)

DEDIC_GPIO2_INT_ENA The enable bit for DEDIC_GPIO2_INT_ST register. (R/W)

DEDIC_GPIO3_INT_ENA The enable bit for DEDIC_GPIO3_INT_ST register. (R/W)

DEDIC_GPIO4_INT_ENA The enable bit for DEDIC_GPIO4_INT_ST register. (R/W)

DEDIC_GPIO5_INT_ENA The enable bit for DEDIC_GPIO5_INT_ST register. (R/W)

DEDIC_GPIO6_INT_ENA The enable bit for DEDIC_GPIO6_INT_ST register. (R/W)

DEDIC_GPIO7_INT_ENA The enable bit for DEDIC_GPIO7_INT_ST register. (R/W)

Register 5.50: DEDIC_GPIO_INTR_ST_REG (0x0028)

(reserved)	DEDIC_GPIO7_INT_ST	DEDIC_GPIO6_INT_ST	DEDIC_GPIO5_INT_ST	DEDIC_GPIO4_INT_ST	DEDIC_GPIO3_INT_ST	DEDIC_GPIO2_INT_ST	DEDIC_GPIO1_INT_ST	DEDIC_GPIO0_INT_ST	
31 ... 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

DEDIC_GPIO0_INT_ST This is the status bit for DEDIC_GPIO0_INT_RAW when DEDIC_GPIO0_INT_ENA is set to 1. (RO)

DEDIC_GPIO1_INT_ST This is the status bit for DEDIC_GPIO1_INT_RAW when DEDIC_GPIO1_INT_ENA is set to 1. (RO)

DEDIC_GPIO2_INT_ST This is the status bit for DEDIC_GPIO2_INT_RAW when DEDIC_GPIO2_INT_ENA is set to 1. (RO)

DEDIC_GPIO3_INT_ST This is the status bit for DEDIC_GPIO3_INT_RAW when DEDIC_GPIO3_INT_ENA is set to 1. (RO)

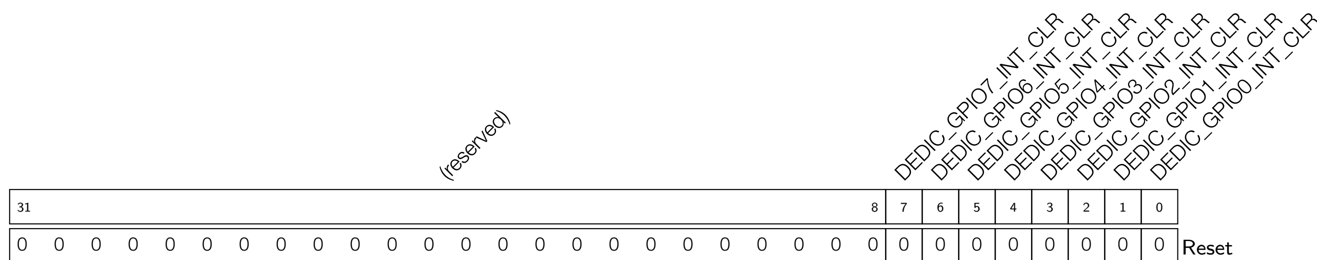
DEDIC_GPIO4_INT_ST This is the status bit for DEDIC_GPIO4_INT_RAW when DEDIC_GPIO4_INT_ENA is set to 1. (RO)

DEDIC_GPIO5_INT_ST This is the status bit for DEDIC_GPIO5_INT_RAW when DEDIC_GPIO5_INT_ENA is set to 1. (RO)

DEDIC_GPIO6_INT_ST This is the status bit for DEDIC_GPIO6_INT_RAW when DEDIC_GPIO6_INT_ENA is set to 1. (RO)

DEDIC_GPIO7_INT_ST This is the status bit for DEDIC_GPIO7_INT_RAW when DEDIC_GPIO7_INT_ENA is set to 1. (RO)

Register 5.51: DEDIC_GPIO_INTR_CLR_REG (0x002C)



DEDIC_GPIO0_INT_CLR Set this bit to clear the DEDIC_GPIO0_INT_RAW interrupt. (WO)

DEDIC_GPIO1_INT_CLR Set this bit to clear the DEDIC_GPIO1_INT_RAW interrupt. (WO)

DEDIC_GPIO2_INT_CLR Set this bit to clear the DEDIC_GPIO2_INT_RAW interrupt. (WO)

DEDIC_GPIO3_INT_CLR Set this bit to clear the DEDIC_GPIO3_INT_RAW interrupt. (WO)

DEDIC_GPIO4_INT_CLR Set this bit to clear the DEDIC_GPIO4_INT_RAW interrupt. (WO)

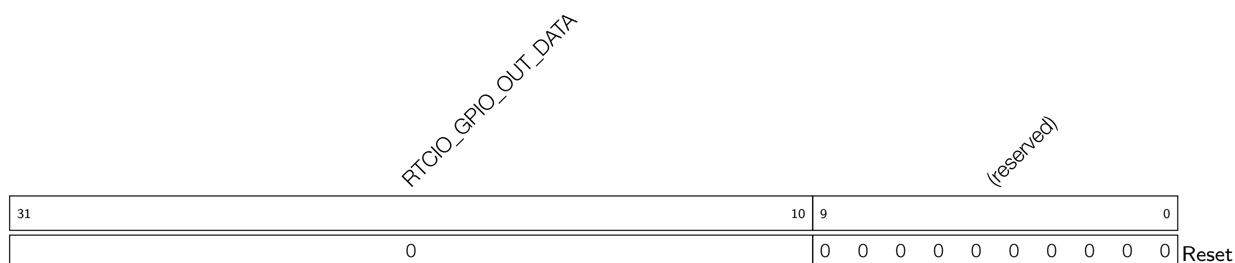
DEDIC_GPIO5_INT_CLR Set this bit to clear the DEDIC_GPIO5_INT_RAW interrupt. (WO)

DEDIC_GPIO6_INT_CLR Set this bit to clear the DEDIC_GPIO6_INT_RAW interrupt. (WO)

DEDIC_GPIO7_INT_CLR Set this bit to clear the DEDIC_GPIO7_INT_RAW interrupt. (WO)

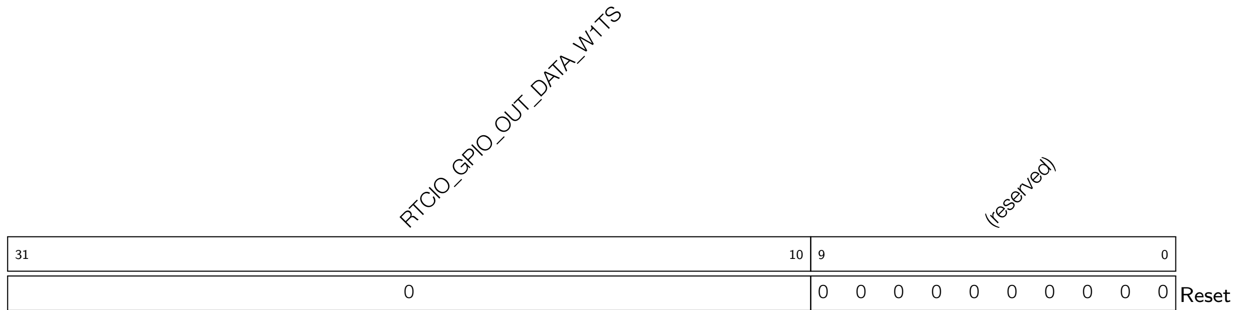
5.15.5 RTC IO MUX Registers

Register 5.52: RTCIO_RTC_GPIO_OUT_REG (0x0000)



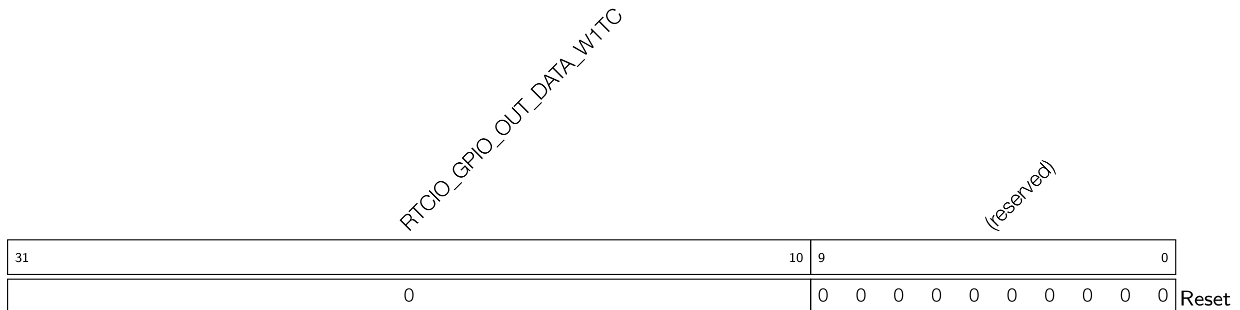
RTCIO_GPIO_OUT_DATA GPIO0 ~ 21 output register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. (R/W)

Register 5.53: RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0004)



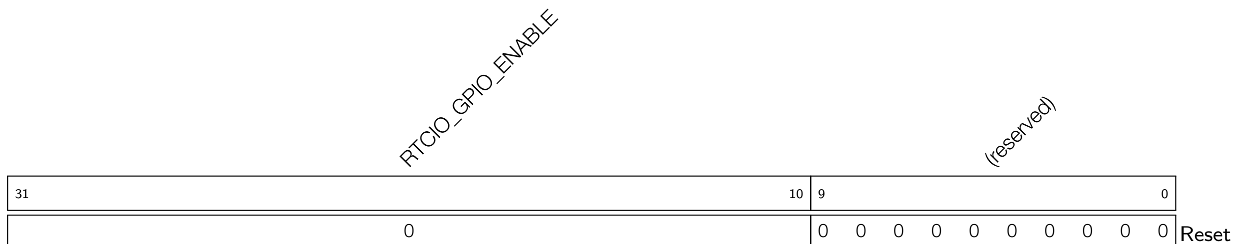
RTCIO_GPIO_OUT_DATA_W1TS GPIO0 ~ 21 output set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set RTCIO_RTC_GPIO_OUT_REG. (WO)

Register 5.54: RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0008)



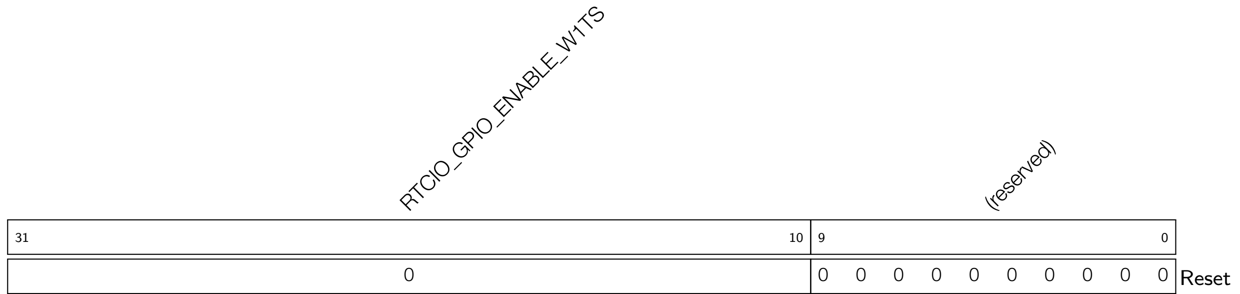
RTCIO_GPIO_OUT_DATA_W1TC GPIO0 ~ 21 output clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear RTCIO_RTC_GPIO_OUT_REG. (WO)

Register 5.55: RTCIO_RTC_GPIO_ENABLE_REG (0x000C)



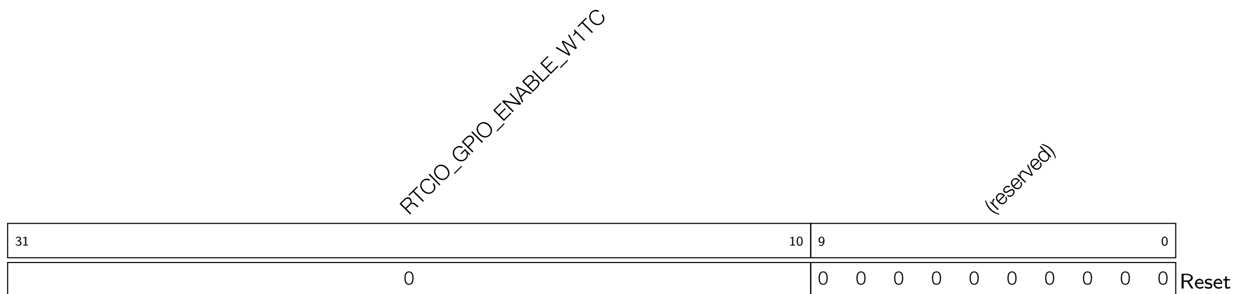
RTCIO_GPIO_ENABLE GPIO0 ~ 21 output enable. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. If the bit is set to 1, it means this GPIO pad is output. (R/W)

Register 5.56: RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0010)



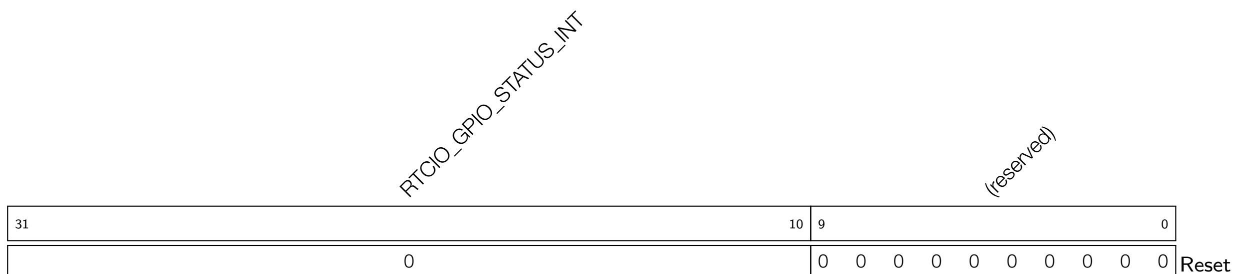
RTCIO_GPIO_ENABLE_W1TS GPIO0 ~ 21 output enable set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set RTCIO_RTC_GPIO_ENABLE_REG. (WO)

Register 5.57: RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0014)



RTCIO_GPIO_ENABLE_W1TC GPIO0 ~ 21 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear RTCIO_RTC_GPIO_ENABLE_REG. (WO)

Register 5.58: RTCIO_RTC_GPIO_STATUS_REG (0x0018)

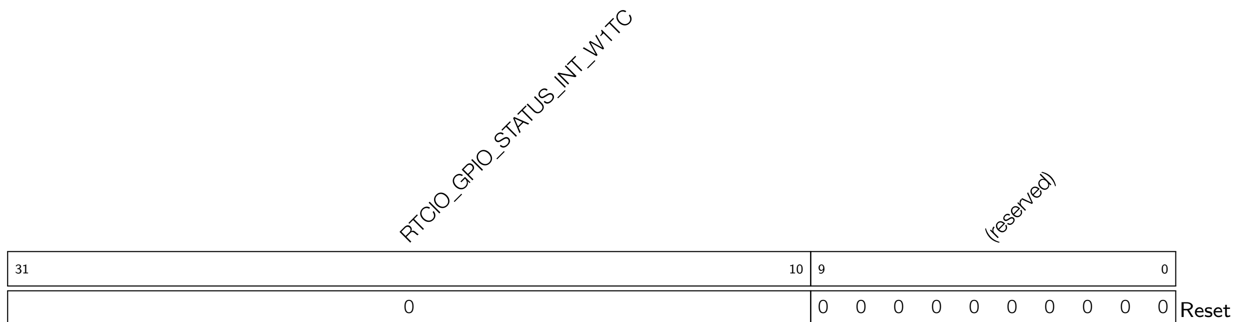


RTCIO_GPIO_STATUS_INT GPIO0 ~ 21 interrupt status register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. This register should be used together with RTCIO_RTC_GPIO_PIN*n*_INT_TYPE in RTCIO_RTC_GPIO_PIN*n*_REG. 0: no interrupt; 1: corresponding interrupt. (R/W)

Register 5.59: RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x001C)

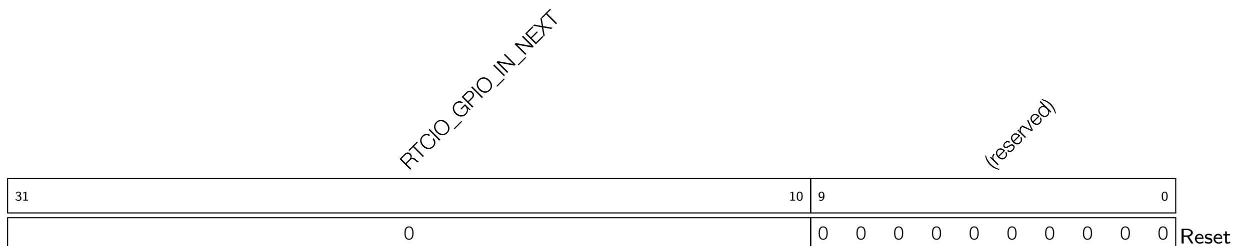
RTCIO_GPIO_STATUS_INT_W1TS GPIO0 ~ 21 interrupt set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_GPIO_STATUS_INT will be set to 1. Recommended operation: use this register to set RTCIO_GPIO_STATUS_INT. (WO)

Register 5.60: RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0020)



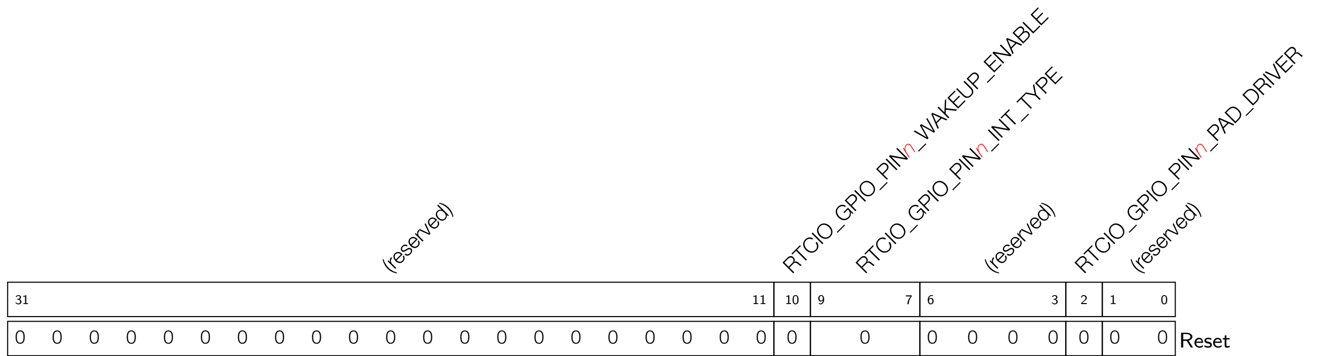
RTCIO_GPIO_STATUS_INT_W1TC GPIO0 ~ 21 interrupt clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_GPIO_STATUS_INT will be cleared. Recommended operation: use this register to clear RTCIO_GPIO_STATUS_INT. (WO)

Register 5.61: RTCIO_RTC_GPIO_IN_REG (0x0024)



RTCIO_GPIO_IN_NEXT GPIO0 ~ 21 input value. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. Each bit represents a pad input value, 1 for high level, and 0 for low level. (RO)

Register 5.62: RTCIO_RTC_GPIO_PINn_REG (n: 0-21) (0x0028+4*n)



RTCIO_GPIO_PINn_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain. (R/W)

RTCIO_GPIO_PINn_INT_TYPE GPIO interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

RTCIO_GPIO_PINn_WAKEUP_ENABLE GPIO wake-up enable. This will only wake up ESP32-S2 from Light-sleep. (R/W)

Register 5.63: RTCIO_TOUCH_PADn_REG (n: 0-14) (0x0084+4*n)

Bits	31	30–29	28	27	26	25–23	22	21	20	19	18–17	16	15	14	13	12–0
Field	(reserved)	RTCIO_TOUCH_PADn_DRV	RTCIO_TOUCH_PADn_RDE	RTCIO_TOUCH_PADn_RUE	(reserved)	RTCIO_TOUCH_PADn_DAC	RTCIO_TOUCH_PADn_START	RTCIO_TOUCH_PADn_TIE_OPT	RTCIO_TOUCH_PADn_XPD	RTCIO_TOUCH_PADn_MUX_SEL	RTCIO_TOUCH_PADn_FUN_SEL	RTCIO_TOUCH_PADn_SLP_SEL	RTCIO_TOUCH_PADn_SLP_IE	RTCIO_TOUCH_PADn_SLP_OE	RTCIO_TOUCH_PADn_FUN_IE	(reserved)
Reset	0	2	1	0	0	0x4	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0

RTCIO_TOUCH_PADn_FUN_IE Input enable in normal execution. (R/W)

RTCIO_TOUCH_PADn_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_TOUCH_PADn_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_TOUCH_PADn_SLP_SEL 0: no sleep mode; 1: enable sleep mode. (R/W)

RTCIO_TOUCH_PADn_FUN_SEL Function selection. (R/W)

RTCIO_TOUCH_PADn_MUX_SEL Connect the RTC pad input to digital pad input. 0 is available. (R/W)

RTCIO_TOUCH_PADn_XPD Touch sensor power on. (R/W)

RTCIO_TOUCH_PADn_TIE_OPT The tie option of touch sensor. 0: tie low; 1: tie high. (R/W)

RTCIO_TOUCH_PADn_START Start touch sensor. (R/W)

RTCIO_TOUCH_PADn_DAC Touch sensor slope control. 3-bit for each touch pad, defaults to 0x4. (R/W)

RTCIO_TOUCH_PADn_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_TOUCH_PADn_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_TOUCH_PADn_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.64: RTCIO_XTAL_32P_PAD_REG (0x00C0)

Bits	31	30–29	28	27	26–20	19	18–17	16	15	14	13	12–0
Field	(reserved)	RTCIO_X32P_DRV	RTCIO_X32P_RDE	RTCIO_X32P_RUE	(reserved)	RTCIO_X32P_MUX_SEL	RTCIO_X32P_FUN_SEL	RTCIO_X32P_SLP_SEL	RTCIO_X32P_SLP_IE	RTCIO_X32P_SLP_OE	RTCIO_X32P_FUN_IE	(reserved)
Reset	0	2	0	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0

RTCIO_X32P_FUN_IE Input enable in normal execution. (R/W)

RTCIO_X32P_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_X32P_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_X32P_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_X32P_FUN_SEL Function selection (R/W)

RTCIO_X32P_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_X32P_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_X32P_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_X32P_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.65: RTCIO_XTAL_32N_PAD_REG (0x00C4)

Bits	31	30–29	28	27	26–20	19	18–17	16	15	14	13	12–0
Field	(reserved)	RTCIO_X32N_DRV	RTCIO_X32N_RDE	RTCIO_X32N_RUE	(reserved)	RTCIO_X32N_MUX_SEL	RTCIO_X32N_FUN_SEL	RTCIO_X32N_SLP_SEL	RTCIO_X32N_SLP_IE	RTCIO_X32N_SLP_OE	RTCIO_X32N_FUN_IE	(reserved)
Reset	0	2	0	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0

RTCIO_X32N_FUN_IE Input enable in normal execution. (R/W)

RTCIO_X32N_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_X32N_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_X32N_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_X32N_FUN_SEL Function selection (R/W)

RTCIO_X32N_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_X32N_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_X32N_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_X32N_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.66: RTCIO_PAD_DAC1_REG (0x00C8)

(reserved)	RTCIO_PDAC1_DRV	RTCIO_PDAC1_RDE	RTCIO_PDAC1_RUE	(reserved)	RTCIO_PDAC1_MUX_SEL	RTCIO_PDAC1_FUN_SEL	RTCIO_PDAC1_SLP_SEL	RTCIO_PDAC1_SLP_IE	RTCIO_PDAC1_SLP_OE	RTCIO_PDAC1_FUN_IE	RTCIO_PDAC1_DAC_XPD_FORCE	RTCIO_PDAC1_XPD_DAC	RTCIO_PDAC1_DAC	(reserved)	
31	30 29	28	27	26 20	19	18 17	16	15	14	13	12	11	10 3	2 0	
0	2	0	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0 0 0	Reset

RTCIO_PDAC1_DAC Configure DAC_1 output when RTCIO_PDAC1_DAC_XPD_FORCE is set to 1. (R/W)

RTCIO_PDAC1_XPD_DAC When RTCIO_PDAC1_DAC_XPD_FORCE is set to 1, 1: enable DAC_1 output; 0: disable DAC_1 output. (R/W)

RTCIO_PDAC1_DAC_XPD_FORCE 1: use RTCIO_PDAC1_XPD_DAC to control DAC_1 output; 0: use SAR ADC FSM to control DAC_1 output. (R/W)

RTCIO_PDAC1_FUN_IE Input enable in normal execution. (R/W)

RTCIO_PDAC1_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_PDAC1_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_PDAC1_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_PDAC1_FUN_SEL DAC_1 function selection. (R/W)

RTCIO_PDAC1_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_PDAC1_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_PDAC1_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_PDAC1_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.67: RTCIO_PAD_DAC2_REG (0x00CC)

(reserved)	RTCIO_PDAC2_DRV	RTCIO_PDAC2_RDE	RTCIO_PDAC2_RUE	(reserved)	RTCIO_PDAC2_MUX_SEL	RTCIO_PDAC2_FUN_SEL	RTCIO_PDAC2_SLP_SEL	RTCIO_PDAC2_SLP_IE	RTCIO_PDAC2_SLP_OE	RTCIO_PDAC2_FUN_IE	RTCIO_PDAC2_DAC_XPD_FORCE	RTCIO_PDAC2_XPD_DAC	RTCIO_PDAC2_DAC	(reserved)	
31	30 29	28	27	26 20	19	18 17	16	15	14	13	12	11	10 3	2 0	
0	2	0	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0 0 0	Reset

RTCIO_PDAC2_DAC Configure DAC_2 output when RTCIO_PDAC2_DAC_XPD_FORCE is set to 1. (R/W)

RTCIO_PDAC2_XPD_DAC When RTCIO_PDAC2_DAC_XPD_FORCE is set to 1, 1: enable DAC_2 output; 0: disable DAC_2 output. (R/W)

RTCIO_PDAC2_DAC_XPD_FORCE 1: use RTCIO_PDAC2_XPD_DAC to control DAC_2 output; 0: use SAR ADC FSM to control DAC_2 output. (R/W)

RTCIO_PDAC2_FUN_IE Input enable in normal execution. (R/W)

RTCIO_PDAC2_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_PDAC2_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_PDAC2_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_PDAC2_FUN_SEL DAC_2 function selection. (R/W)

RTCIO_PDAC2_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO. (R/W)

RTCIO_PDAC2_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_PDAC2_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_PDAC2_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.68: RTCIO_RTC_PAD19_REG (0x00D0)

31	30–29	28	27	26–20	19	18–17	16	15	14	13	12–0	
(reserved)	RTCIO_RTC_PAD19_DRV	RTCIO_RTC_PAD19_RDE	RTCIO_RTC_PAD19_RUE	(reserved)	RTCIO_RTC_PAD19_MUX_SEL	RTCIO_RTC_PAD19_FUN_SEL	RTCIO_RTC_PAD19_SLP_SEL	RTCIO_RTC_PAD19_SLP_IE	RTCIO_RTC_PAD19_SLP_OE	RTCIO_RTC_PAD19_FUN_IE	(reserved)	
0	2	1	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_PAD19_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD19_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD19_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD19_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD19_FUN_SEL Function selection (R/W)

RTCIO_RTC_PAD19_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_RTC_PAD19_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD19_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_RTC_PAD19_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.69: RTCIO_RTC_PAD20_REG (0x00D4)

Bits	31	30–29	28	27	26–20	19	18–17	16	15	14	13	12–0
Field	(reserved)	RTCIO_RTC_PAD20_DRV	RTCIO_RTC_PAD20_RDE	RTCIO_RTC_PAD20_RUE	(reserved)	RTCIO_RTC_PAD20_MUX_SEL	RTCIO_RTC_PAD20_FUN_SEL	RTCIO_RTC_PAD20_SLP_SEL	RTCIO_RTC_PAD20_SLP_IE	RTCIO_RTC_PAD20_SLP_OE	RTCIO_RTC_PAD20_FUN_IE	(reserved)
Reset	0	2	1	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0

RTCIO_RTC_PAD20_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD20_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD20_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD20_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD20_FUN_SEL Function selection. (R/W)

RTCIO_RTC_PAD20_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO. (R/W)

RTCIO_RTC_PAD20_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD20_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_RTC_PAD20_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.70: RTCIO_RTC_PAD21_REG (0x00D8)

(reserved)	RTCIO_RTC_PAD21_DRV	RTCIO_RTC_PAD21_RDE	RTCIO_RTC_PAD21_RUE	(reserved)	RTCIO_RTC_PAD21_MUX_SEL	RTCIO_RTC_PAD21_FUN_SEL	RTCIO_RTC_PAD21_SLP_SEL	RTCIO_RTC_PAD21_SLP_IE	RTCIO_RTC_PAD21_SLP_OE	RTCIO_RTC_PAD21_FUN_IE	(reserved)	
31	30 29	28	27	26 20	19	18 17	16	15	14	13	12 0	
0	2	1	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_PAD21_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD21_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD21_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD21_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD21_FUN_SEL Function selection. (R/W)

RTCIO_RTC_PAD21_MUX_SEL 1: use RTC GPIO,0: use digital GPIO. (R/W)

RTCIO_RTC_PAD21_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD21_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

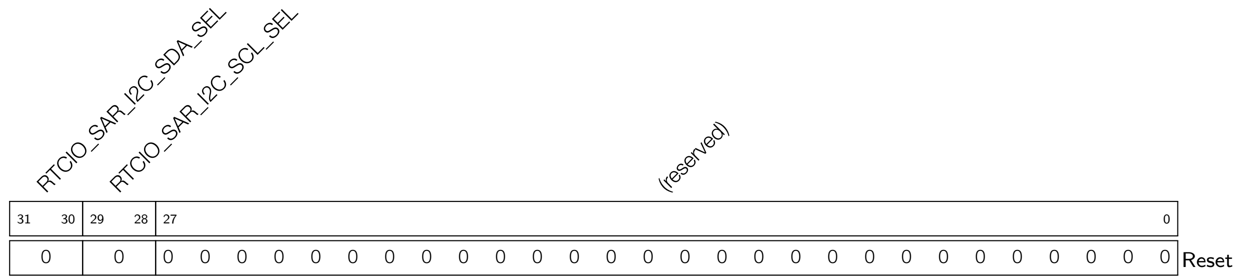
RTCIO_RTC_PAD21_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.71: RTCIO_XTL_EXT_CTR_REG (0x00E0)

RTCIO_XTL_EXT_CTR_SEL	(reserved)	
31 27	26 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_XTL_EXT_CTR_SEL Select the external crystal power down enable source to get into sleep mode. 0: select GPIO0; 1: select GPIO1, etc. The input value on this pin XOR RTC_CNTL_EXT_XTL_CONF_REG[30] is the crystal power down enable signal. (R/W)

Register 5.72: RTCIO_SAR_I2C_IO_REG (0x00E4)



RTCIO_SAR_I2C_SCL_SEL Selects a pad the RTC I2C SCL signal connects to. 0: use TOUCH PAD0; 1: use TOUCH PAD2. (R/W)

RTCIO_SAR_I2C_SDA_SEL Selects a pad the RTC I2C SDA signal connects to. 0: use TOUCH PAD1; 1: use TOUCH PAD3. (R/W)

6. Reset and Clock

6.1 Reset

6.1.1 Overview

ESP32-S2 provides four types of reset that occur at different levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset.

All reset types mentioned above (except Chip Reset) maintain the data stored in internal memory. Figure 6-1 shows the scopes of affected subsystems when different types of reset occur.

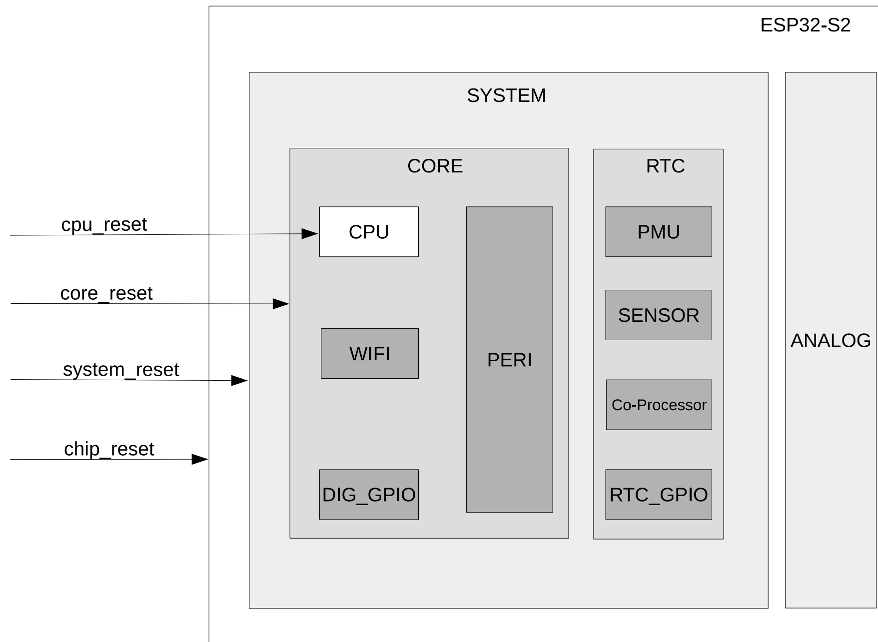


Figure 6-1. System Reset

- CPU Reset: Only resets CPU core. Once such reset is released, programs will be executed from CPU reset vector.
- Core Reset: Resets the whole digital system except RTC, including CPU, peripherals, Wi-Fi, and digital GPIOs.
- System Reset: Resets the whole digital system, including RTC.
- Chip Reset: Resets the whole chip.

6.1.2 Reset Source

CPU will be reset immediately when any of the reset above occurs. Users can get reset source codes by reading register RTC_CNTL_RESET_CAUSE_PROCPU after the reset is released.

Table 45 lists different reset sources and the types of reset they trigger.

Table 45: Reset Source

Code	Source	Reset Type	Comments
0x01	Chip reset	Chip Reset	See the note below
0x0F	Brown-out system reset	System Reset	Triggered by brown-out detector
0x10	RWDT system reset	System Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x13	GLITCH reset	System Reset	-
0x03	Software system reset	Core Reset	Triggered by configuring RTC_CNTL_SW_SYS_RST
0x05	Deep-sleep reset	Core Reset	See Chapter 9 *Low-Power Management (RTC_CNTL)*
0x07	MWDT0 global reset	Core Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x08	MWDT1 global reset	Core Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x09	RWDT core reset	Core Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x0C	Software CPU reset	CPU Reset	Triggered by configuring RTC_CNTL_SW_PROCPU_RST
0x0D	RWDT CPU reset	CPU Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x11	MWDT1 CPU reset	CPU Reset	See Chapter 12 *Watchdog Timers (WDT)*

Note:

- Chip Reset can be triggered by the following three sources:
  - Triggered by chip power-on;
  - Triggered by brown-out detector;
  - Triggered by Super Watchdog (SWD).
- Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on register configuration. For more information, please see Chapter 9 *Low-Power Management (RTC_CNTL)*.

6.2 Clock

6.2.1 Overview

ESP32-S2 provides multiple clock sources, which allow CPU, peripherals and RTC to work at different frequencies, thus providing more flexibility in meeting the requirements of various application scenarios. Figure 6-2 shows the system clock structure.

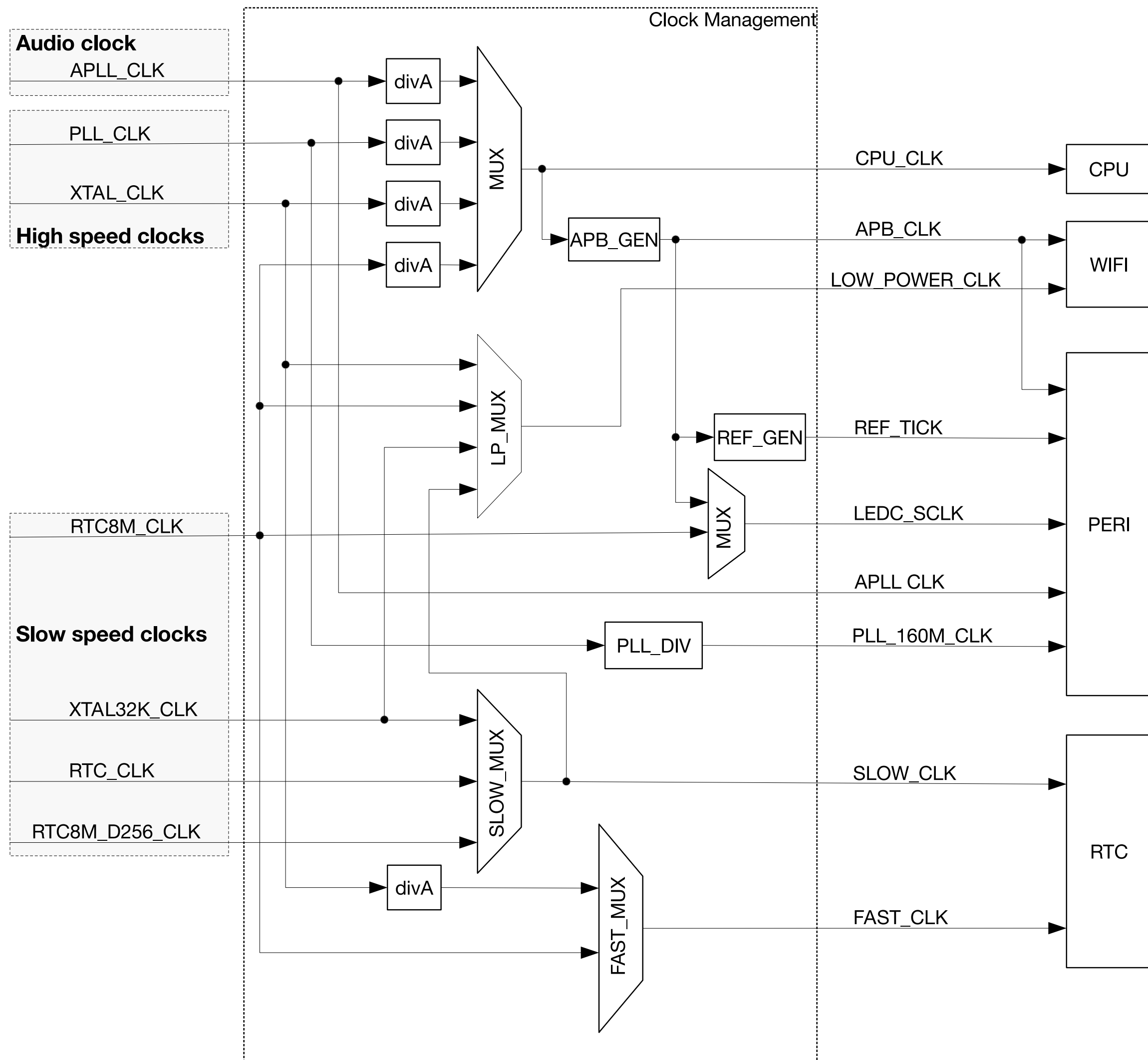


Figure 6-2. System Clock

6.2.2 Clock Source

ESP32-S2 uses external crystal, internal PLL, or internal oscillator working as clock sources to generate different kinds of clocks, which can be classified in three types depending on their clock speed.

- High speed clock for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_CLK (320 MHz or 480 MHz): internal PLL clock
  - XTAL_CLK (40 MHz): external crystal clock
- Slow speed clock for low-power devices, such as power management unit and low-power peripherals
  - XTAL32K_CLK (32 kHz): external crystal clock
  - RTC8M_CLK (8 MHz by default): internal divide-by-N oscillator of 8 MHz, with adjustable frequency
  - RTC8M_D256_CLK (31.250 kHz by default): internal clock derived from RTC8M_CLK divided by 256
  - RTC_CLK (90 kHz by default): internal oscillator with adjustable frequency
- Audio clock for audio-related devices
  - APLL_CLK (16 MHz ~ 128 MHz): internal Audio PLL clock

6.2.3 CPU Clock

As Figure 6-2 shows, CPU_CLK is the master clock for CPU and it can be as high as 240 MHz when CPU works in high performance mode. Alternatively, CPU can run at lower frequencies, such as at 2 MHz, to lower power consumption.

Users can set PLL_CLK, APLL_CLK, RTC8M_CLK or XTAL_CLK as CPU_CLK clock source by configuring register SYSTEM_SOC_CLK_SEL, see Table 46 and Table 47.

Table 46: CPU_CLK Source

SYSTEM_SOC_CLK_SEL Value	Clock Source
0	XTAL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

Table 47: CPU_CLK Selection

Clock Source	SEL_0*	SEL_1*	SEL_2*	CPU Clock Frequency
XTAL_CLK	0	-	-	CPU_CLK = XTAL_CLK / (SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
PLL_CLK (480 MHz)	1	1	0	CPU_CLK = PLL_CLK / 6 CPU_CLK frequency is 80 MHz
PLL_CLK (480 MHz)	1	1	1	CPU_CLK = PLL_CLK / 3 CPU_CLK frequency is 160 MHz
PLL_CLK (480 MHz)	1	1	2	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 240 MHz
PLL_CLK (320 MHz)	1	0	0	CPU_CLK = PLL_CLK / 4 CPU_CLK frequency is 80 MHz
PLL_CLK (320 MHz)	1	0	1	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 160 MHz
RTC8M_CLK	2	-	-	CPU_CLK = RTC8M_CLK / (SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
APLL_CLK	3	0	0	CPU_CLK = APLL_CLK / 4
APLL_CLK	3	0	1	CPU_CLK = APLL_CLK / 2

*SEL_0: The value of register SYSTEM_SOC_CLK_SEL.

*SEL_1: The value of register SYSTEM_PLL_FREQ_SEL.

*SEL_2: The value of register SYSTEM_CPUPERIOD_SEL.

Note:

- When users select XTAL_CLK as CPU clock source and adjust the divider value by configuring register SYSTEM_PRE_DIV_CNT, the rules below should be followed.
  - If current divider value is 2 (SYSTEM_PRE_DIV_CNT = 1) and the target is x (x ≠ 1), users should set the divider first to 1 (SYSTEM_PRE_DIV_CNT = 0) and then to x (SYSTEM_PRE_DIV_CNT = x - 1).
  - If current divider value is x (SYSTEM_PRE_DIV_CNT = x - 1) and the target is 2, users should set the divider first to 1 (SYSTEM_PRE_DIV_CNT = 0) and then to 2 (SYSTEM_PRE_DIV_CNT = 1).

– For other target divider value x, users can adjust the register directly (SYSTEM_PRE_DIV_CNT = x - 1).

6.2.4 Peripheral Clock

Peripheral clocks include APB_CLK, REF_TICK, LEDC_PWM_CLK, APLL_CLK and PLL_160M_CLK. Table 48 shows which clock can be used by which peripheral.

Table 48: Peripheral Clock Usage

Peripheral	APB_CLK	REF_TICK	LEDC_PWM_CLK	APLL_CLK	PLL_160M_CLK
TIMG	Y	Y			
I²S	Y			Y	Y
UHCI	Y				
UART	Y	Y			
RMT	Y	Y			
LED_PWM	Y	Y	Y		
I²C	Y	Y			
SPI	Y				
PCNT	Y				
eFuse Controller	Y				
SARADC/DAC	Y			Y	
USB	Y				
CRYPTO					Y
TWAI Controller	Y				
System Timer	Y				

6.2.4.1 APB_CLK Source

APB_CLK is determined by the clock source of CPU_CLK as shown in Table 49.

Table 49: APB_CLK Source

CPU_CLK Source	APB_CLK
PLL_CLK	80 MHz
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

6.2.4.2 REF_TICK Source

REF_TICK is derived from XTAL_CLK or RTC8M_CLK via a divider. When PLL_CLK, APLL_CLK or XTAL_CLK is set as CPU clock source, REF_TICK will be divided from XTAL_CLK. When RTC8M_CLK is set as CPU clock source, REF_TICK will be divided from RTC8M_CLK. In such way, REF_TICK frequency remains unchanged when APB_CLK changes its clock source. Table 50 shows the configuration of these clock divider registers.

Table 50: REF_TICK Source

CPU_CLK Source	Clock Divider Register
PLL_CLK I XTAL_CLK I APLL_CLK	APB_CTRL_XTAL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

Normally, one REF_TICK cycle lasts for 1 μs, so APB_CTRL_XTAL_TICK_NUM should be configured to 39 (default), and APB_CTRL_CK8M_TICK_NUM to 7 (default).

6.2.4.3 LEDC_PWM_CLK Source

LEDC_PWM_CLK clock source is selected by configuring register LEDC_APB_CLK_SEL, as shown in Table 51.

Table 51: LEDC_PWM_CLK Source

LEDC_APB_CLK_SEL Value	LEDC_PWM_CLK Source
0 (Default)	-
1	APB_CLK
2	RTC8M_CLK
3	XTAL_CLK

6.2.4.4 APLL_SCLK Source

APLL_CLK is sourced from PLL_CLK, and its output frequency is configured using APLL configuration registers. See Section 6.2.7 for more information.

6.2.4.5 PLL_160M_CLK Source

PLL_160M_CLK is divided from PLL_CLK according to current PLL frequency.

6.2.4.6 Clock Source Considerations

Peripherals that need to work with other clocks, such as RMT and I²C, generally operate using PLL_CLK frequency as a reference. When this frequency changes, peripherals should update their clock configuration to operate at the same frequency after the change. Peripherals accessing REF_TICK can continue operating normally without changing their clock configuration when switching clock sources. Please see Table 48.

LED module uses RTC8M_CLK as clock source when APB_CLK is disabled. In other words, when the system is in low-power mode, most peripherals will be halted (APB_CLK is turned off), but LED can work normally via RTC8M_CLK.

6.2.5 Wi-Fi Clock

Wi-Fi can work only when APB_CLK uses PLL_CLK as its clock source. Suspending PLL_CLK requires that Wi-Fi has entered low-power mode first.

LOW_POWER_CLK uses XTAL32K_CLK, XTAL_CLK, RTC8M_CLK or SLOW_CLK (the low clock selected by RTC) as its clock source for Wi-Fi in low-power mode.

6.2.6 RTC Clock

The clock sources for SLOW_CLK and FAST_CLK are low-frequency clocks. RTC module can operate when most other clocks are stopped.

SLOW_CLK derived from RTC_CLK, XTAL32K_CLK or RTC8M_D256_CLK is used to clock Power Management module.

FAST_CLK is used to clock On-chip Sensor module. It can be sourced from a divided XTAL_CLK or from RTC8M_CLK.

6.2.7 Audio PLL Clock

The operation of audio and other time-critical data-transfer applications requires highly-configurable, low-jitter and accurate clock sources. The clock sources derived from system clocks that serve digital peripherals may carry jitter and, therefore, are not suitable for a high-precision clock frequency setting.

Providing an integrated precision clock source can minimize system cost. To this end, ESP32-S2 integrates an audio PLL to clock I²S module.

Audio PLL formula is as follows:

$$f_{\text{out}} = \frac{f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4)}{2(odiv + 2)}$$

Parameters are defined below:

- f_{xtal}: the frequency of crystal oscillator, usually 40 MHz;
- sdm0: the value is 0 ~ 255;
- sdm1: the value is 0 ~ 255;
- sdm2: the value is 0 ~ 63;
- odiv: the value is 0 ~ 31;
- The operating frequency range of the numerator is 350 MHz ~ 500 MHz.

$$350MHz < f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4) < 500MHz$$

Audio PLL can be manually enabled or disabled via registers RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD, respectively. Disabling it takes priority over enabling it. When RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD are 0, PLL follows the state of the system. When the system enters sleep mode, PLL will be disabled automatically; when the system wakes up, PLL will be enabled automatically.

7. Chip Boot Control (BOOTCTRL)

7.1 Overview

ESP32-S2 has three strapping pins:

- GPIO0
- GPIO45
- GPIO46

Software can read the values of the three strapping pins from register GPIO_STRAPPING. During chip reset triggered by power-on, brown-out or by analog super watchdog (see Chapter 6 *Reset and Clock*), hardware samples and stores the voltage level of strapping pins as strapping bit of “0” or “1” in latches, and hold these bits until the chip is powered down or shut down.

By default, GPIO0, GPIO45 and GPIO46 are connected to internal pull-up/pull-down during chip reset. Consequently, if the three GPIOs are unconnected or their connected external circuits are high-impedance, the internal weak pull-up/pull-down will determine the default input level of the strapping pins (see Table 52).

Table 52: Default Configuration of Strapping Pins

Pin	GPIO0	GPIO45	GPIO46
Default	Pull-up	Pull-down	Pull-down

To change the default configuration of strapping pins, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-S2. After the reset is released, the strapping pins work as normal-function pins.

7.2 Boot Mode

GPIO0 and GPIO46 control the boot mode after reset.

Table 53: Boot Mode

Pin	SPI Boot	Download Boot
GPIO0	1	0
GPIO46	x	0

Table 53 shows the strapping pin values and the associated boot modes. ”x” means that this value is ignored. Currently only the two boot modes shown are supported. The strapping combination of GPIO0 = 0 and GPIO46 = 1 is not supported and will trigger unexpected behavior.

In SPI boot mode, the CPU boots the system by reading the program stored in SPI flash.

In download boot mode, users can download code to SRAM or Flash using UART0, UART1, QPI or USB interface. It is also possible to load a program into SRAM and execute it in this mode.

The following eFuses control boot mode behavior:

- EFUSE_DIS_FORCE_DOWNLOAD. If this eFuse is 0 (default), software can switch the chip from SPI boot mode to download boot mode by setting register RTC_CNTL_FORCE_DOWNLOAD_BOOT and triggering

a CPU reset. If this eFuse is 1, this register is disabled.

- EFUSE_DIS_DOWNLOAD_MODE. If this eFuse is 1, download boot mode is disabled.
- EFUSE_ENABLE_SECURITY_DOWNLOAD. If this eFuse is 1, download boot mode only allows reading, writing and erasing plaintext flash and does not support any SRAM or register operations. This eFuse is ignored if download boot mode is disabled.

7.3 ROM Code Printing to UART

GPIO46 controls ROM code printing of information during the early boot process. This GPIO is used together with the UART_PRINT_CONTROL eFuse.

Table 54: ROM Code Printing Control

UART_PRINT_CONTROL	GPIO46	ROM Code Printing
0	-	ROM code will always print information to UART during boot. GPIO46 is not used.
1	0	Print is enabled during boot
	1	Print is disabled
2	0	Print is disabled
	1	Print is enabled during boot
3	-	Print is always disabled during boot. GPIO46 is not used.

ROM code will print to pin U0TXD (default) or to pin DAC_1, depending on the eFuse bit UART_PRINT_CHANNEL (0: UART0; 1: DAC_1).

7.4 VDD_SPI Voltage

GPIO45 is used to select the VDD_SPI power supply voltage at reset:

- GPIO45 = 0, VDD_SPI pin is powered directly from VDD3P3_RTC_IO via resistor R_{SPI}. Typically this voltage is 3.3 V.
- GPIO45 = 1, VDD_SPI pin is powered from internal 1.8 V LDO.

This functionality can be overridden by setting eFuse bit VDD_SPI_FORCE to 1, in which case the VDD_SPI_TIEH eFuse value determines the VDD_SPI voltage:

- VDD_SPI_FORCE = 1 and VDD_SPI_TIEH = 0, VDD_SPI connects to 1.8 V LDO.
- VDD_SPI_FORCE = 1 and VDD_SPI_TIEH = 1, VDD_SPI connects to VDD3P3_RTC_IO.

8. Interrupt Matrix (INTERRUPT)

8.1 Overview

The interrupt matrix embedded in ESP32-S2 independently allocates peripheral interrupt sources to the CPU peripheral interrupts, so as to timely inform the CPU to process the interrupts once the interrupt signals are generated. This flexible function is applicable to a variety of application scenarios.

8.2 Features

- Accept 95 peripheral interrupt sources as input
- Generate 26 peripheral interrupts to the CPU as output
- Disable CPU non-maskable interrupt (NMI) sources
- Query current interrupt status of peripheral interrupt sources

The structure of the interrupt matrix is shown in Figure 8-1.

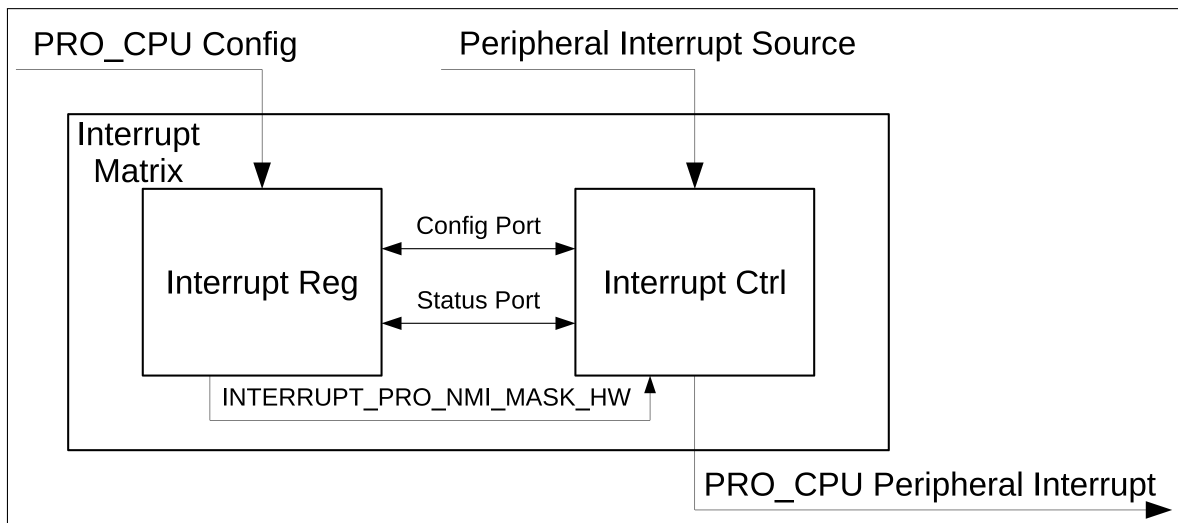


Figure 8-1. Interrupt Matrix Structure

8.3 Functional Description

8.3.1 Peripheral Interrupt Sources

ESP32-S2 has 95 peripheral interrupt sources in total, all of which can be allocated to the CPU. For the peripheral interrupt sources and their configuration/status registers, please refer to Table 55.

Table 55: CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources

No.	Source	Configuration Register	Status Register	
			Bit	Name
0	reserved	reserved	0	INTERRUPT_PRO_INTR_STATUS_REG_0_REG
1	reserved	reserved	1	
2	PWR_INTR	INTERRUPT_PRO_PWR_INTR_MAP_REG	2	
3	reserved	reserved	3	
4	reserved	reserved	4	
5	reserved	reserved	5	
6	reserved	reserved	6	
7	reserved	reserved	7	
8	reserved	reserved	8	
9	reserved	reserved	9	
10	reserved	reserved	10	
11	reserved	reserved	11	
12	reserved	reserved	12	
13	UHCI0_INTR	INTERRUPT_PRO_UHCI0_INTR_MAP_REG	13	
14	reserved	reserved	14	
15	TG_T0_LEVEL_INT	INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG	15	
16	TG_T1_LEVEL_INT	INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG	16	
17	TG_WDT_LEVEL_INT	INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG	17	
18	TG_LACT_LEVEL_INT	INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG	18	
19	TG1_T0_LEVEL_INT	INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG	19	
20	TG1_T1_LEVEL_INT	INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG	20	
21	TG1_WDT_LEVEL_INT	INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	21	
22	TG1_LACT_LEVEL_INT	INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	22	
23	GPIO_INTERRUPT_PRO	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG	23	
24	GPIO_INTERRUPT_PRO_NMI	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	24	
25	reserved	reserved	25	
26	reserved	reserved	26	
27	DEDICATED_GPIO_IN_INTR	INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG	27	
28	CPU_INTR_FROM_CPU_0	INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	28	
29	CPU_INTR_FROM_CPU_1	INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	29	
30	CPU_INTR_FROM_CPU_2	INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	30	
31	CPU_INTR_FROM_CPU_3	INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	31	
32	SPI_INTR_1	INTERRUPT_PRO_SPI_INTR_1_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
33	SPI_INTR_2	INTERRUPT_PRO_SPI_INTR_2_MAP_REG	1	
34	SPI_INTR_3	INTERRUPT_PRO_SPI_INTR_3_MAP_REG	2	

No.	Source	Configuration Register	Status Register	
			Bit	Name
35	I2S0_INT	INTERRUPT_PRO_I2S0_INT_MAP_REG	3	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
36	reserved	reserved	4	
37	UART_INTR	INTERRUPT_PRO_UART_INTR_MAP_REG	5	
38	UART1_INTR	INTERRUPT_PRO_UART1_INTR_MAP_REG	6	
39	reserved	reserved	7	
40	reserved	reserved	8	
41	reserved	reserved	9	
42	reserved	reserved	10	
43	reserved	reserved	11	
44	reserved	reserved	12	
45	LEDC_INT	INTERRUPT_PRO_LEDC_INT_MAP_REG	13	
46	EFUSE_INT	INTERRUPT_PRO_EFUSE_INT_MAP_REG	14	
47	CAN_INT	INTERRUPT_PRO_CAN_INT_MAP_REG	15	
48	USB_INTR	INTERRUPT_PRO_USB_INTR_MAP_REG	16	
49	RTC_CORE_INTR	INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG	17	
50	RMT_INTR	INTERRUPT_PRO_RMT_INTR_MAP_REG	18	
51	PCNT_INTR	INTERRUPT_PRO_PCNT_INTR_MAP_REG	19	
52	I2C_EXT0_INTR	INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG	20	
53	I2C_EXT1_INTR	INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG	21	
54	RSA_INTR	INTERRUPT_PRO_RSA_INTR_MAP_REG	22	
55	SHA_INTR	INTERRUPT_PRO_SHA_INTR_MAP_REG	23	
56	AES_INTR	INTERRUPT_PRO_AES_INTR_MAP_REG	24	
57	SPI2_DMA_INT	INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG	25	
58	SPI3_DMA_INT	INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG	26	
59	reserved	reserved	27	
60	TIMER_INT	INTERRUPT_PRO_TIMER_INT1_MAP_REG	28	
61	TIMER_INT2	INTERRUPT_PRO_TIMER_INT2_MAP_REG	29	
62	TG_T0_EDGE_INT	INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	30	
63	TG_T1_EDGE_INT	INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	31	
64	TG_WDT_EDGE_INT	INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_2_REG
65	TG_LACT_EDGE_INT	INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	1	
66	TG1_T0_EDGE_INT	INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	2	
67	TG1_T1_EDGE_INT	INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	3	
68	TG1_WDT_EDGE_INT	INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	4	
69	TG1_LACT_EDGE_INT	INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	5	
70	CACHE_IA_INT	INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	6	
71	SYSTIMER_TARGET0_INT	INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	7	

No.	Source	Configuration Register	Status Register	
			Bit	Name
72	SYSTIMER_TARGET1_INT	INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	8	
73	SYSTIMER_TARGET2_INT	INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	9	
74	ASSIST_DEBUG_INTR	INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	10	
75	PMS_PRO_IRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	11	
76	PMS_PRO_DRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	12	
77	PMS_PRO_DPORT_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	13	
78	PMS_PRO_AHB_ILG_INTR	INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	14	
79	PMS_PRO_CACHE_ILG_INTR	INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	15	
80	PMS_DMA_APB_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG	16	
81	PMS_DMA_RX_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG	17	
82	PMS_DMA_TX_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG	18	
83	SPI_MEM_REJECT_INTR	INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG	19	
84	DMA_COPY_INTR	INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG	20	
85	reserved	reserved	21	
86	reserved	reserved	22	
87	DCACHE_PRELOAD_INT	INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	23	INTERRUPT_PRO_INTR_STATUS_REG_2_REG
88	ICACHE_PRELOAD_INT	INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG	24	
89	APB_ADC_INT	INTERRUPT_PRO_APB_ADC_INT_MAP_REG	25	
90	CRYPTO_DMA_INT	INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG	26	
91	CPU_PERI_ERROR_INT	INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG	27	
92	APB_PERI_ERROR_INT	INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG	28	
93	DCACHE_SYNC_INT	INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG	29	
94	ICACHE_SYNC_INT	INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG	30	

8.3.2 CPU Interrupts

The CPU has 32 interrupts, including 26 peripheral interrupts and six internal interrupts. Table 56 lists all the interrupts.

- Peripheral Interrupts:
  - Level-triggered interrupts: triggered by high level signal. The interrupt sources should hold the level till the CPU handles the interrupts.
  - Edge-triggered interrupts: triggered on rising edge. The CPU responds to this kind of interrupts immediately.
  - NMI interrupt: once triggered, the NMI interrupt can not be masked by software using the CPU registers.
- Internal Interrupts:
  - Timer interrupts: triggered by internal timers and are used to generate periodic interrupts.
  - Software interrupts: triggered when software writes to special registers.
  - Profiling interrupt: triggered for performance monitoring and analysis.

ESP32-S2 supports the above-mentioned 32 interrupts at six levels as shown in the table below. A higher level corresponds to a higher priority. NMI has the highest interrupt priority and once triggered, the CPU must handle such interrupt.

Table 56: CPU Interrupts

No.	Category	Type	Priority Level
0	Peripheral	Level-triggered	1
1	Peripheral	Level-triggered	1
2	Peripheral	Level-triggered	1
3	Peripheral	Level-triggered	1
4	Peripheral	Level-triggered	1
5	Peripheral	Level-triggered	1
6	Internal	Timer.0	1
7	Internal	Software	1
8	Peripheral	Level-triggered	1
9	Peripheral	Level-triggered	1
10	Peripheral	Edge-triggered	1
11	Internal	Profiling	3
12	Peripheral	Level-triggered	1
13	Peripheral	Level-triggered	1
14	Peripheral	NMI	NMI
15	Internal	Timer.1	3
16	Internal	Timer.2	5
17	Peripheral	Level-triggered	1
18	Peripheral	Level-triggered	1
19	Peripheral	Level-triggered	2
20	Peripheral	Level-triggered	2

No.	Category	Type	Priority Level
21	Peripheral	Level-triggered	2
22	Peripheral	Edge-triggered	3
23	Peripheral	Level-triggered	3
24	Peripheral	Level-triggered	4
25	Peripheral	Level-triggered	4
26	Peripheral	Level-triggered	5
27	Peripheral	Level-triggered	3
28	Peripheral	Edge-triggered	4
29	Internal	Software	3
30	Peripheral	Edge-triggered	4
31	Peripheral	Level-triggered	5

8.3.3 Allocate Peripheral Interrupt Source to CPU Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- Source_X: stands for a particular peripheral interrupt source, wherein, X means the number of this interrupt source in Table 55.
- INTERRUPT_PRO_X_MAP_REG: stands for a peripheral interrupt configuration register, corresponding to the peripheral interrupt source Source_X. In Table 55, the registers listed in column "Configuration Register" correspond to the peripheral interrupt sources listed in column "Source". For example, the configuration register for source UHCI0_INTR is INTERRUPT_PRO_UHCI0_INTR_MAP_REG.
- Interrupt_P: stands for the CPU peripheral interrupt numbered as Num_P. The value of Num_P can be 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28 and 30 ~ 31 (see Table 56).
- Interrupt_I: stands for the CPU internal interrupt numbered as Num_I. The value of Num_I can be 6, 7, 11, 15, 16 and 29 (see Table 56).

8.3.3.1 Allocate one peripheral interrupt source Source_X to CPU

Setting the corresponding configuration register INTERRUPT_PRO_X_MAP_REG of Source_X to Num_P will allocate this interrupt source to Interrupt_P. Num_P here can be any value from 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28 and 30 ~ 31. Note that one CPU interrupt can be shared by multiple peripherals.

8.3.3.2 Allocate multiple peripheral interrupt sources Source_X*n* to CPU

Setting the corresponding configuration register INTERRUPT_PRO_X*n*_MAP_REG of each interrupt source to the same Num_P will allocate all the sources to the same Interrupt_P. Any of these sources will trigger CPU Interrupt_P. When an interrupt signal is generated, software should check the interrupt status registers to figure out which peripheral the signal comes from.

8.3.3.3 Disable CPU peripheral interrupt source Source_X

Setting the corresponding configuration register INTERRUPT_PRO_X_MAP_REG of the source to any Num_I will disable this interrupt Source_X. The choice of Num_I does not matter, as none of Num_I is connected to the CPU. Therefore this functionality can be used to disable peripheral interrupt sources.

8.3.4 Disable CPU NMI Interrupt Sources

The interrupt matrix is able to mask all peripheral interrupt sources allocated to CPU No.14 NMI interrupt using hardware, depending on the internal signal INTERRUPT_PRO_NMI_MASK_HW. The signal comes from "Interrupt Reg" register configuration submodule inside interrupt matrix, see Figure 8-1. If the signal is set to high level, CPU will not respond to NMI interrupt.

8.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

Current interrupt status of a peripheral interrupt source can be read via the bit value in INTERRUPT_PRO_INTR_STATUS_REG_*n*. For the mapping between INTERRUPT_PRO_INTR_STATUS_REG_*n* and peripheral interrupt sources, please refer to Table 55.

8.4 Base Address

Users can access interrupt matrix with the base address, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 57: Interrupt Matrix Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C2000

8.5 Register Summary

The address in the following table represents the address offset (relative address) with the respect to the peripheral base address, not the absolute address. For detailed information about the interrupt matrix base address, please refer to Section 8.4.

Name	Description	Address	Access
Configuration registers			
INTERRUPT_PRO_PWR_INTR_MAP_REG	PWR_INTR interrupt configuration register	0x0008	R/W
INTERRUPT_PRO_UHCI0_INTR_MAP_REG	UHCI0_INTR interrupt configuration register	0x0034	R/W
INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG	TG_T0_LEVEL_INT interrupt configuration register	0x003C	R/W
INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG	TG_T1_LEVEL_INT interrupt configuration register	0x0040	R/W
INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG	TG_WDT_LEVEL_INT interrupt configuration register	0x0044	R/W
INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG	TG_LACT_LEVEL_INT interrupt configuration register	0x0048	R/W
INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG	TG1_T0_LEVEL_INT interrupt configuration register	0x004C	R/W
INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG	TG1_T1_LEVEL_INT interrupt configuration register	0x0050	R/W
INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	TG1_WDT_LEVEL_INT interrupt configuration register	0x0054	R/W
INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	TG1_LACT_LEVEL_INT interrupt configuration register	0x0058	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO interrupt configuration register	0x005C	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	GPIO_INTERRUPT_PRO_NMI interrupt configuration register	0x0060	R/W
INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG	DEDICATED_GPIO_IN_INTR interrupt configuration register	0x006C	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 interrupt configuration register	0x0070	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 interrupt configuration register	0x0074	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 interrupt configuration register	0x0078	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 interrupt configuration register	0x007C	R/W
INTERRUPT_PRO_SPI_INTR_1_MAP_REG	SPI_INTR_1 interrupt configuration register	0x0080	R/W
INTERRUPT_PRO_SPI_INTR_2_MAP_REG	SPI_INTR_2 interrupt configuration register	0x0084	R/W
INTERRUPT_PRO_SPI_INTR_3_MAP_REG	SPI_INTR_3 interrupt configuration register	0x0088	R/W
INTERRUPT_PRO_I2S0_INT_MAP_REG	I2S0_INT interrupt configuration register	0x008C	R/W
INTERRUPT_PRO_UART_INTR_MAP_REG	UART_INT interrupt configuration register	0x0094	R/W
INTERRUPT_PRO_UART1_INTR_MAP_REG	UART1_INT interrupt configuration register	0x0098	R/W
INTERRUPT_PRO_LEDC_INT_MAP_REG	LEDC_INTR interrupt configuration register	0x00B4	R/W
INTERRUPT_PRO_EFUSE_INT_MAP_REG	EFUSE_INT interrupt configuration register	0x00B8	R/W
INTERRUPT_PRO_CAN_INT_MAP_REG	CAN_INT interrupt configuration register	0x00BC	R/W
INTERRUPT_PRO_USB_INTR_MAP_REG	USB_INT interrupt configuration register	0x00C0	R/W
INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG	RTC_CORE_INTR interrupt configuration register	0x00C4	R/W

Name	Description	Address	Access
INTERRUPT_PRO_RMT_INTR_MAP_REG	RMT_INTR interrupt configuration register	0x00C8	R/W
INTERRUPT_PRO_PCNT_INTR_MAP_REG	PCNT_INTR interrupt configuration register	0x00CC	R/W
INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG	I2C_EXT0_INTR interrupt configuration register	0x00D0	R/W
INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG	I2C_EXT1_INTR interrupt configuration register	0x00D4	R/W
INTERRUPT_PRO_RSA_INTR_MAP_REG	RSA_INTR interrupt configuration register	0x00D8	R/W
INTERRUPT_PRO_SHA_INTR_MAP_REG	SHA_INTR interrupt configuration register	0x00DC	R/W
INTERRUPT_PRO_AES_INTR_MAP_REG	AES_INTR interrupt configuration register	0x00E0	R/W
INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG	SPI2_DMA_INT interrupt configuration register	0x00E4	R/W
INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG	SPI3_DMA_INT interrupt configuration register	0x00E8	R/W
INTERRUPT_PRO_TIMER_INT1_MAP_REG	TIMER_INT1 interrupt configuration register	0x00F0	R/W
INTERRUPT_PRO_TIMER_INT2_MAP_REG	TIMER_INT2 interrupt configuration register	0x00F4	R/W
INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	TG_T0_EDGE_INT interrupt configuration register	0x00F8	R/W
INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	TG_T1_EDGE_INT interrupt configuration register	0x00FC	R/W
INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	TG_WDT_EDGE_INT interrupt configuration register	0x0100	R/W
INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	TG_LACT_EDGE_INT interrupt configuration register	0x0104	R/W
INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	TG1_T0_EDGE_INT interrupt configuration register	0x0108	R/W
INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	TG1_T1_EDGE_INT interrupt configuration register	0x010C	R/W
INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	TG1_WDT_EDGE_INT interrupt configuration register	0x0110	R/W
INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	TG1_LACT_EDGE_INT interrupt configuration register	0x0114	R/W
INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	CACHE_IA_INT interrupt configuration register	0x0118	R/W
INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INT interrupt configuration register	0x011C	R/W
INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INT interrupt configuration register	0x0120	R/W
INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2 interrupt configuration register	0x0124	R/W
INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR interrupt configuration register	0x0128	R/W
INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	PMS_PRO_IRAM0_ILG interrupt configuration register	0x012C	R/W
INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	PMS_PRO_DRAM0_ILG interrupt configuration register	0x0130	R/W
INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	PMS_PRO_DPORT_ILG interrupt configuration register	0x0134	R/W
INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	PMS_PRO_AHB_ILG interrupt configuration register	0x0138	R/W
INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	PMS_PRO_CACHE_ILG interrupt configuration register	0x013C	R/W

Name	Description	Address	Access
INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG	PMS_DMA_APB_I_ILG interrupt configuration register	0x0140	R/W
INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG	PMS_DMA_RX_I_ILG interrupt configuration register	0x0144	R/W
INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG	PMS_DMA_TX_I_ILG interrupt configuration register	0x0148	R/W
INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG	SPI_MEM_REJECT_INTR interrupt configuration register	0x014C	R/W
INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG	DMA_COPY_INTR interrupt configuration register	0x0150	R/W
INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	DCACHE_PRELOAD_INT interrupt configuration register	0x015C	R/W
INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG	ICACHE_PRELOAD_INT interrupt configuration register	0x0160	R/W
INTERRUPT_PRO_APB_ADC_INT_MAP_REG	APB_ADC_INT interrupt configuration register	0x0164	R/W
INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG	CRYPTO_DMA_INT interrupt configuration register	0x0168	R/W
INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG	CPU_PERI_ERROR_INT interrupt configuration register	0x016C	R/W
INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG	APB_PERI_ERROR_INT interrupt configuration register	0x0170	R/W
INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG	DCACHE_SYNC_INT interrupt configuration register	0x0174	R/W
INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG	ICACHE_SYNC_INT interrupt configuration register	0x0178	R/W
INTERRUPT_CLOCK_GATE_REG	NMI interrupt signals mask register	0x0188	R/W
Interrupt status registers			
INTERRUPT_PRO_INTR_STATUS_REG_0_REG	Interrupt status register 0	0x017C	RO
INTERRUPT_PRO_INTR_STATUS_REG_1_REG	Interrupt status register 1	0x0180	RO
INTERRUPT_PRO_INTR_STATUS_REG_2_REG	Interrupt status register 2	0x0184	RO
Version register			
INTERRUPT_DATE_REG	Version control register	0x0FFC	R/W

8.6 Registers

The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the interrupt matrix base address, please refer to Section 8.4.

Register 8.1: INTERRUPT_PRO_PWR_INTR_MAP_REG (0x0008)

(reserved)	INTERRUPT_PRO_PWR_INTR_MAP	
31 … 5	4 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

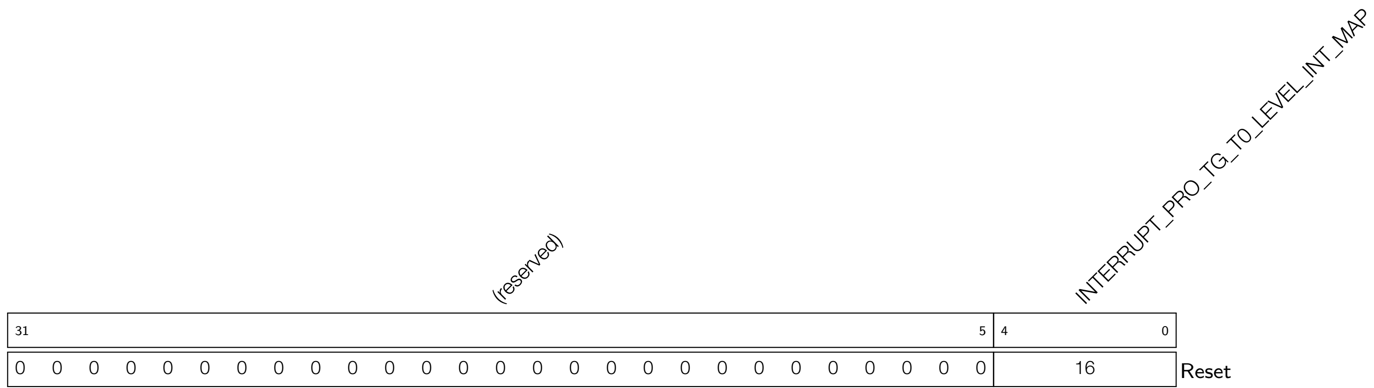
INTERRUPT_PRO_PWR_INTR_MAP This register is used to map PWR_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.2: INTERRUPT_PRO_UHCI0_INTR_MAP_REG (0x0034)

(reserved)	INTERRUPT_PRO_UHCI0_INTR_MAP	
31 … 5	4 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

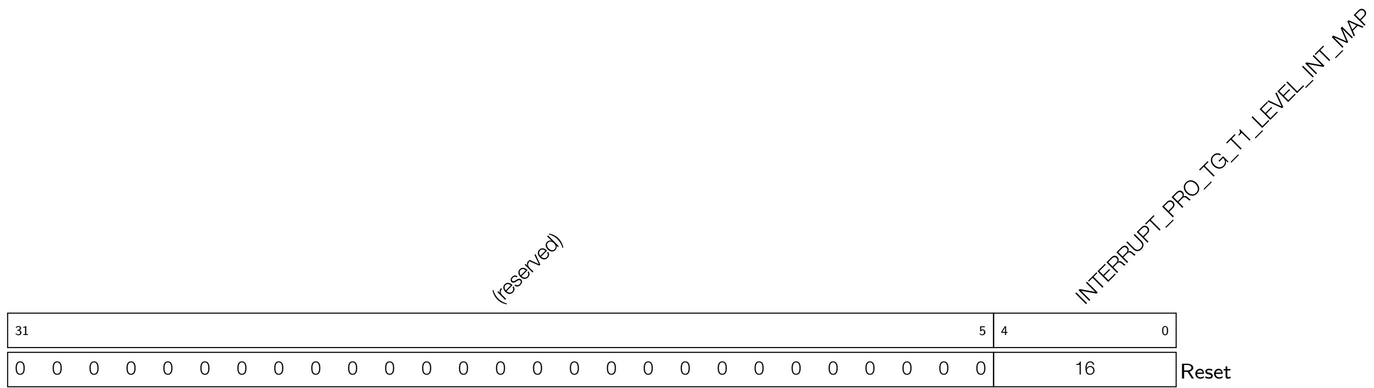
INTERRUPT_PRO_UHCI0_INTR_MAP This register is used to map UHCI0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.3: INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG (0x003C)



INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP This register is used to map TG_T0_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.4: INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG (0x0040)



INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP This register is used to map TG_T1_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.5: INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG (0x0044)

(reserved)	INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

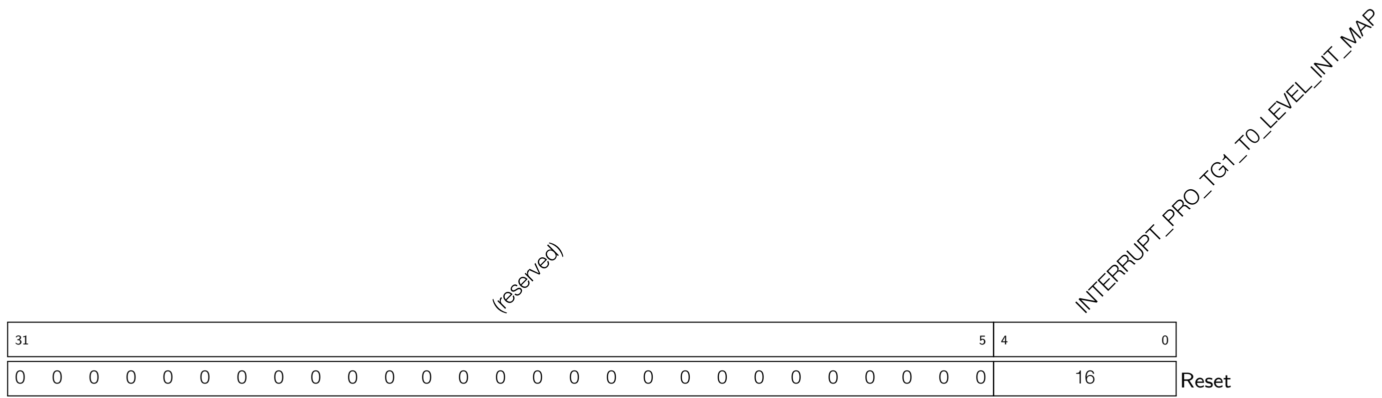
INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP This register is used to map TG_WDT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.6: INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG (0x0048)

(reserved)	INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

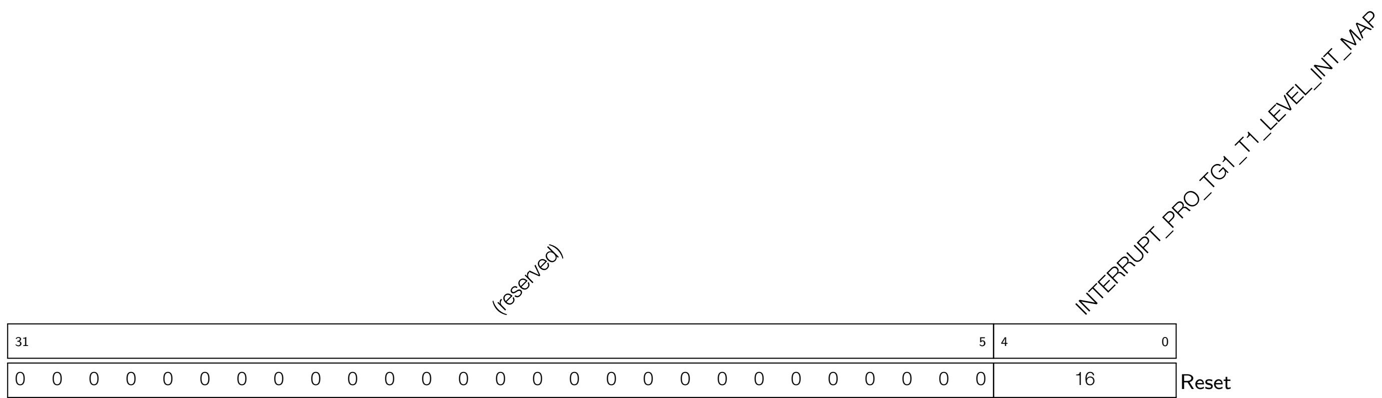
INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP This register is used to map TG_LACT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.7: INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG (0x004C)



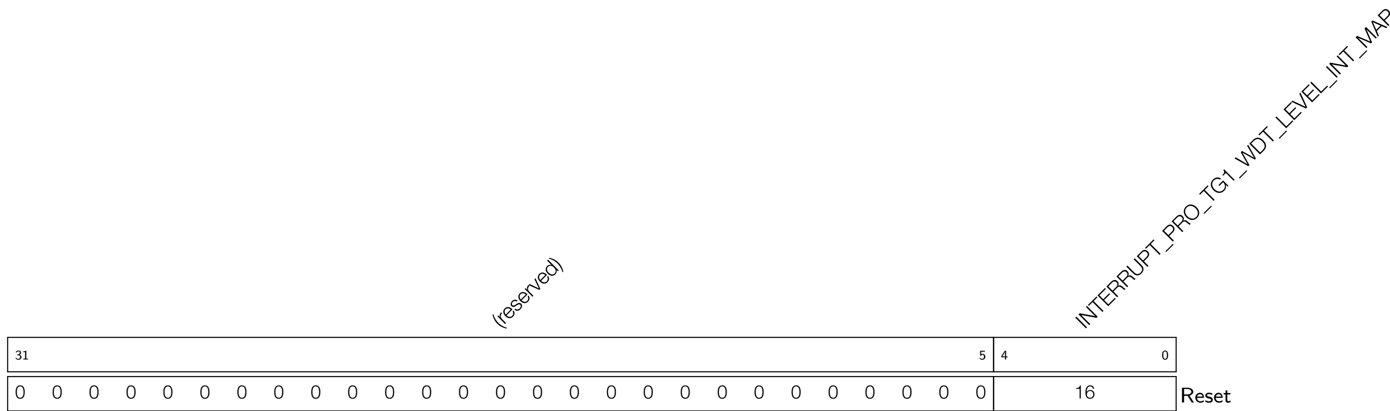
INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP This register is used to map TG1_T0_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.8: INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG (0x0050)



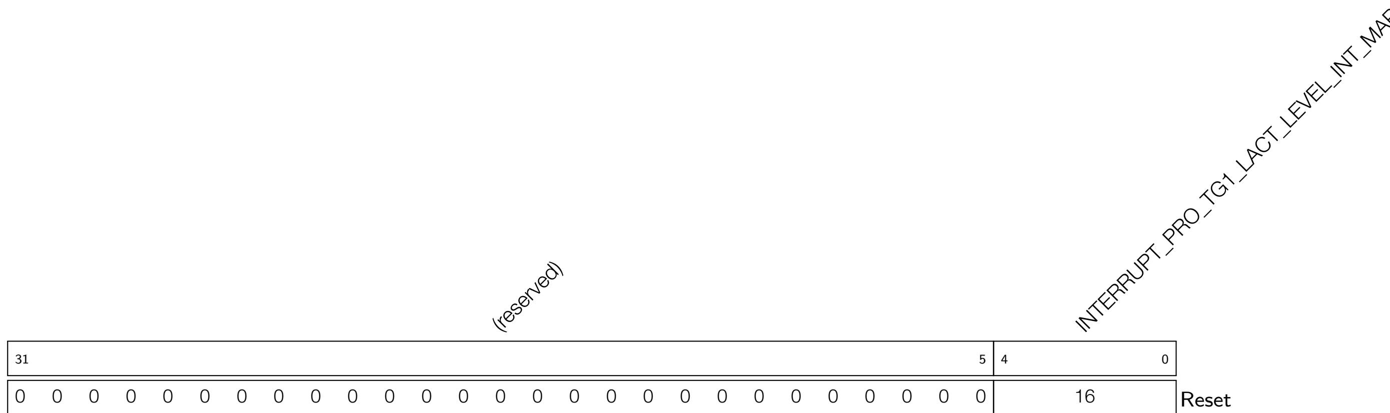
INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP This register is used to map TG1_T1_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.9: INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG (0x0054)



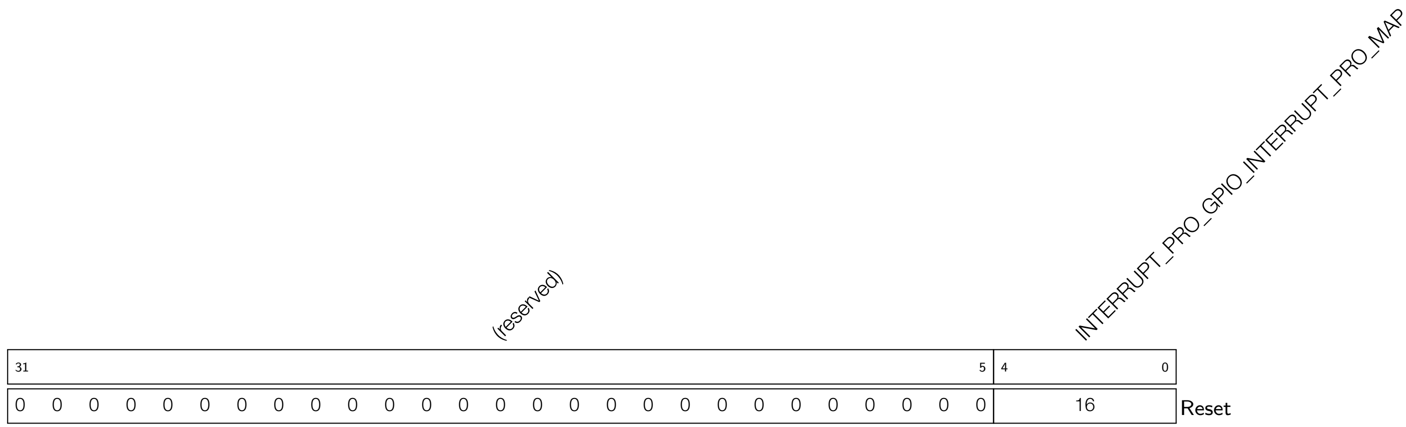
INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP This register is used to map TG1_WDT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.10: INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG (0x0058)



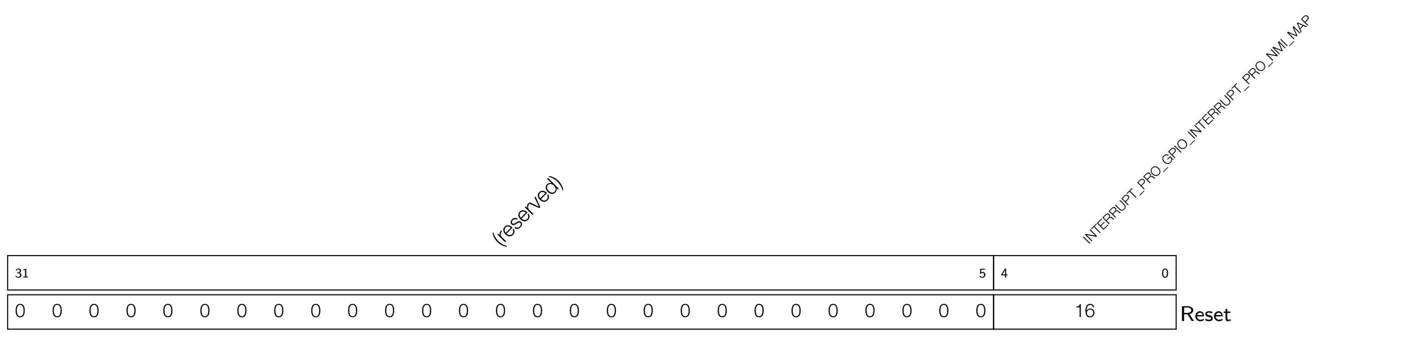
INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP This register is used to map TG1_LACT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.11: INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG (0x005C)



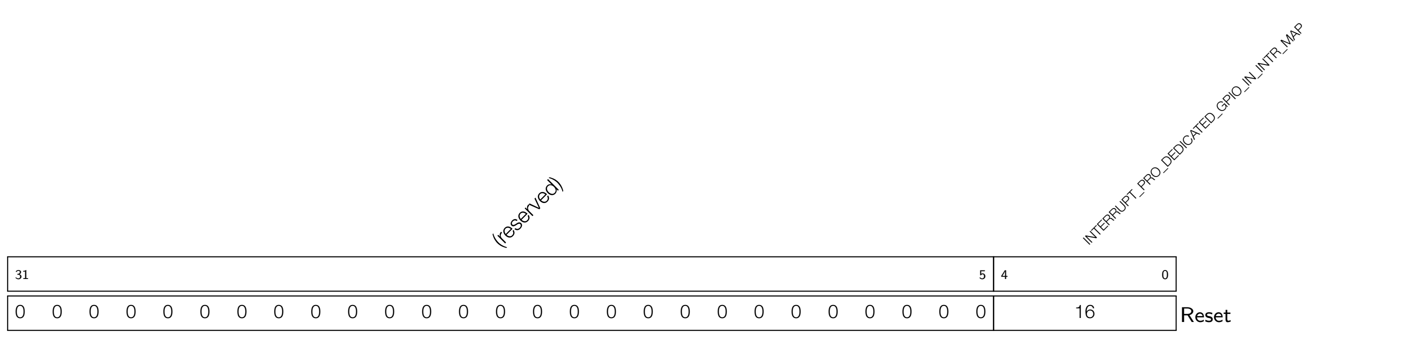
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP This register is used to map GPIO_INTERRUPT_PRO interrupt signal to one of the CPU interrupts. (R/W)

Register 8.12: INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG (0x0060)



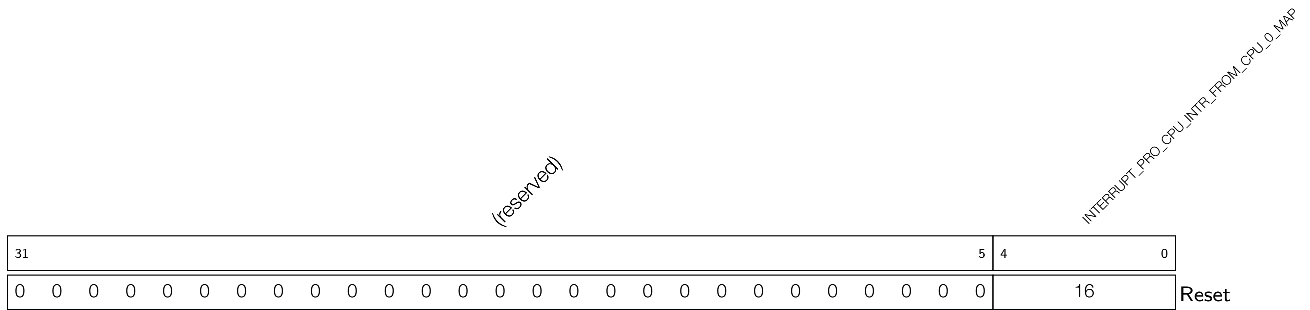
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP This register is used to map GPIO_INTERRUPT_PRO_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 8.13: INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG (0x006C)



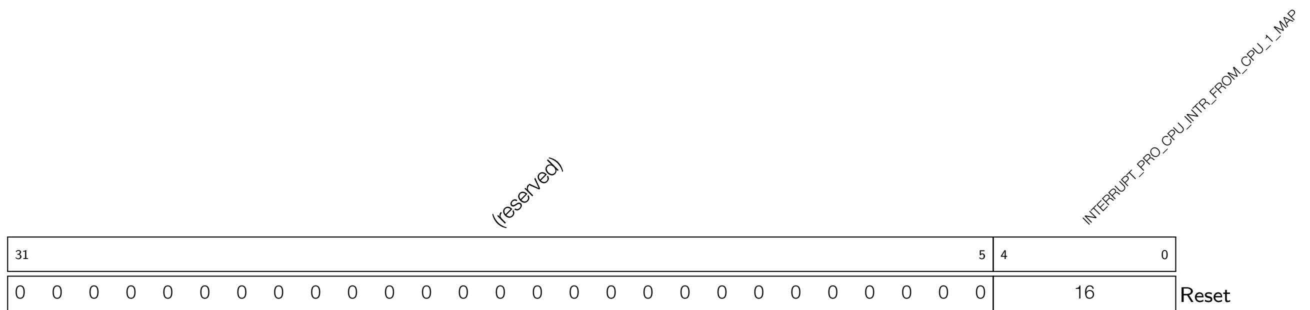
INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP This register is used to map DEDICATED_GPIO_IN_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.14: INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG (0x0070)



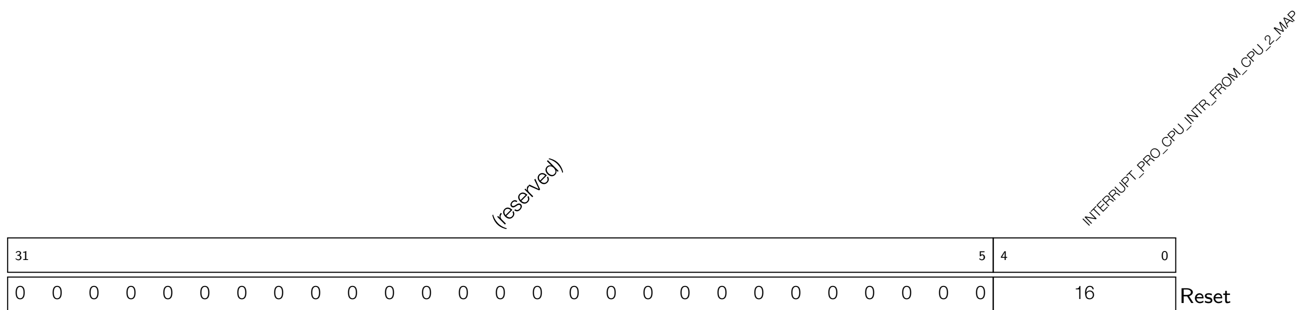
INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP This register is used to map CPU_INTR_FROM_CPU_0 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.15: INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG (0x0074)



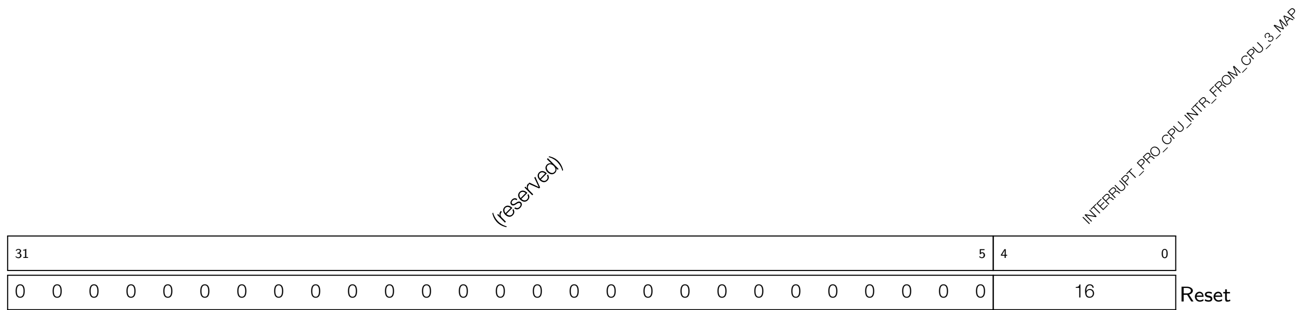
INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP This register is used to map CPU_INTR_FROM_CPU_1 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.16: INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG (0x0078)



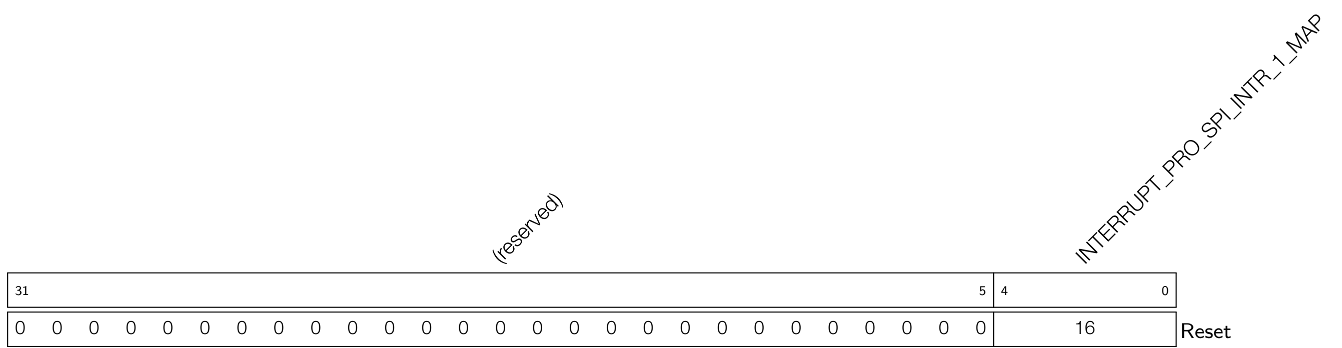
INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP This register is used to map CPU_INTR_FROM_CPU_2 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.17: INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG (0x007C)



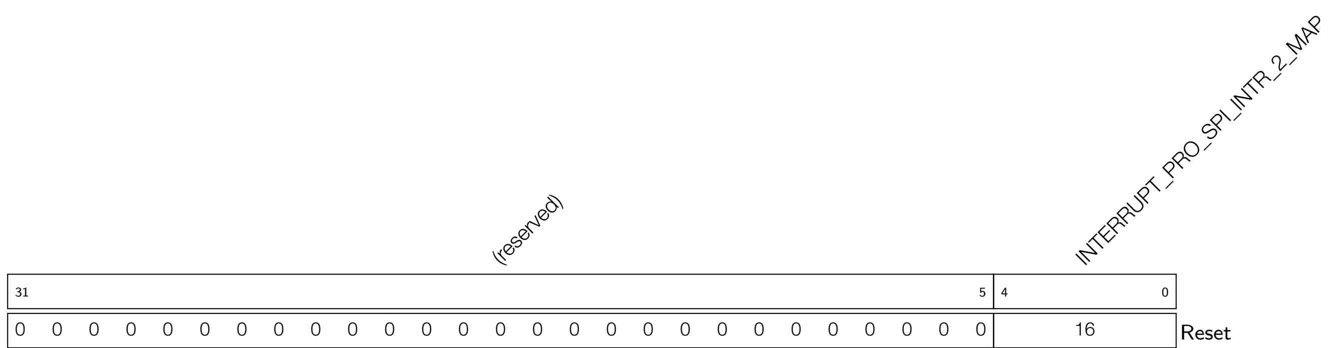
INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP This register is used to map CPU_INTR_FROM_CPU_3 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.18: INTERRUPT_PRO_SPI_INTR_1_MAP_REG (0x0080)



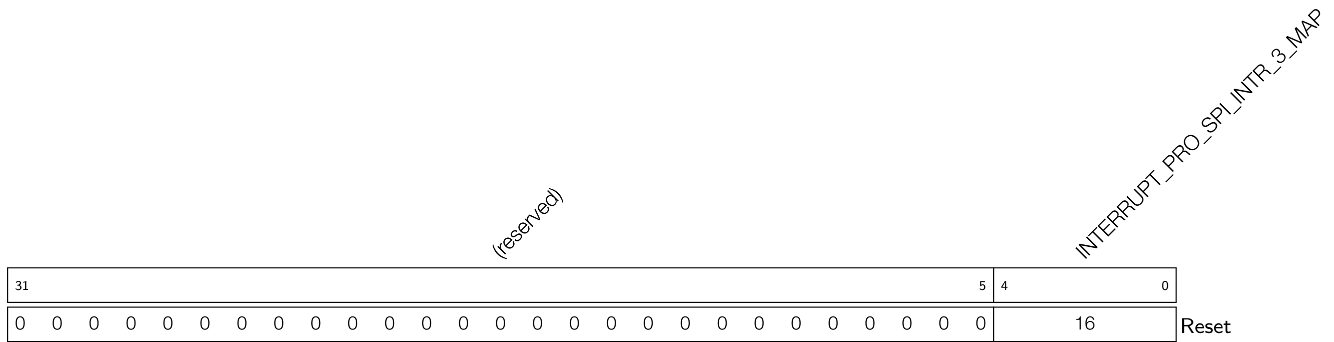
INTERRUPT_PRO_SPI_INTR_1_MAP This register is used to map SPI_INTR_1 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.19: INTERRUPT_PRO_SPI_INTR_2_MAP_REG (0x0084)



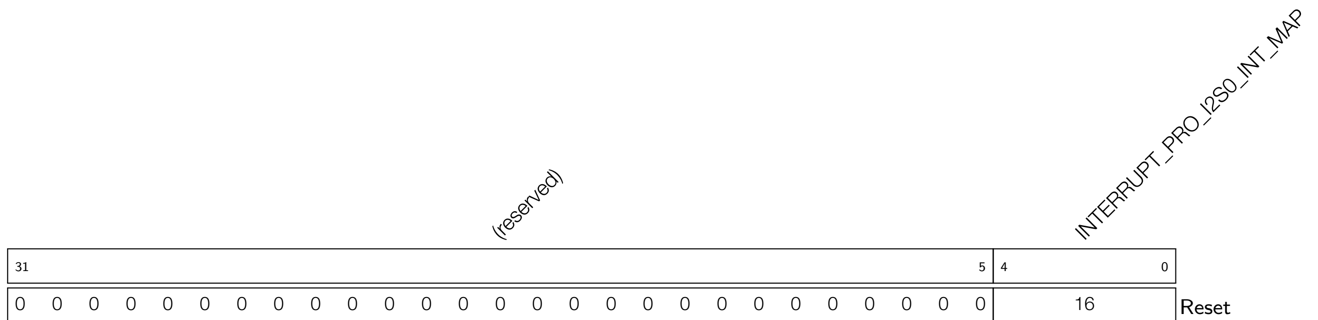
INTERRUPT_PRO_SPI_INTR_2_MAP This register is used to map SPI_INTR_2 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.20: INTERRUPT_PRO_SPI_INTR_3_MAP_REG (0x0088)



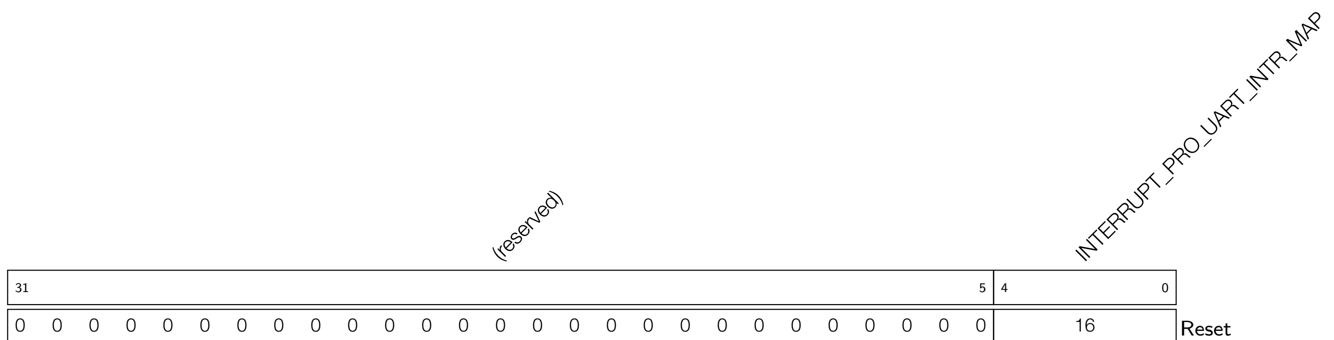
INTERRUPT_PRO_SPI_INTR_3_MAP This register is used to map SPI_INTR_3 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.21: INTERRUPT_PRO_I2S0_INT_MAP_REG (0x008C)



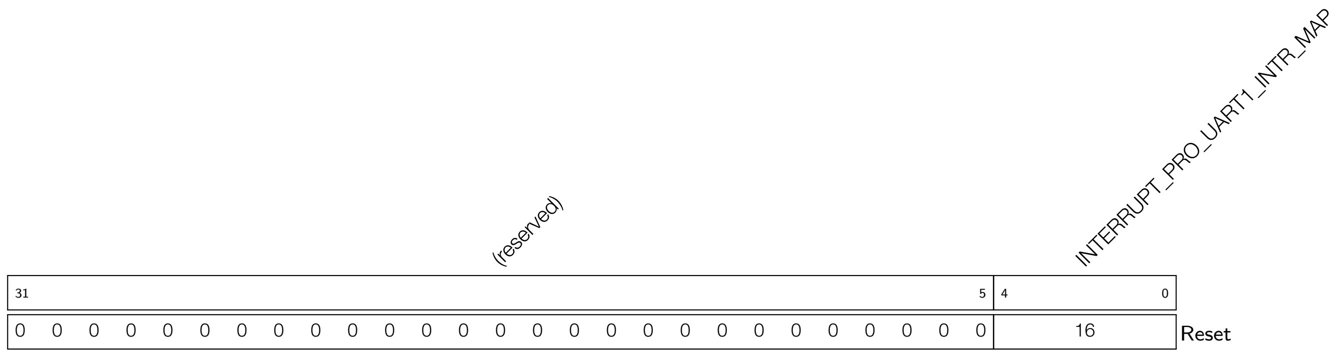
INTERRUPT_PRO_I2S0_INT_MAP This register is used to map I2S0_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.22: INTERRUPT_PRO_UART_INTR_MAP_REG (0x0094)



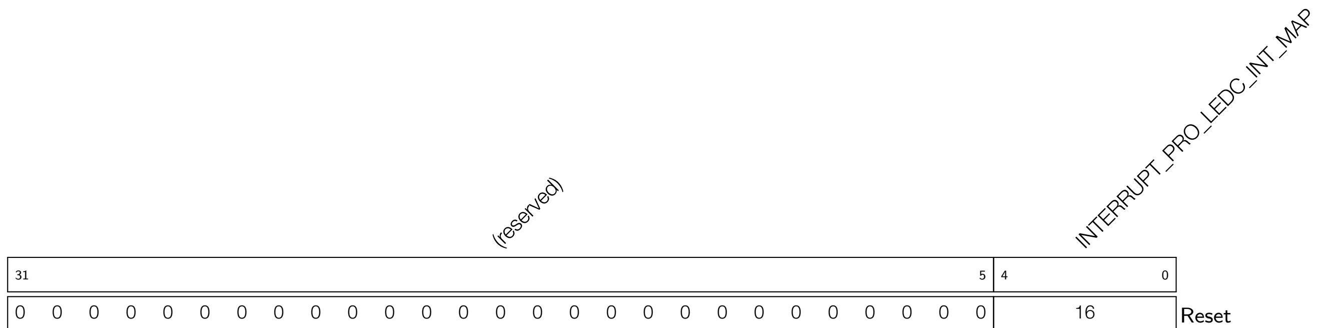
INTERRUPT_PRO_UART_INTR_MAP This register is used to map UART_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.23: INTERRUPT_PRO_UART1_INTR_MAP_REG (0x0098)



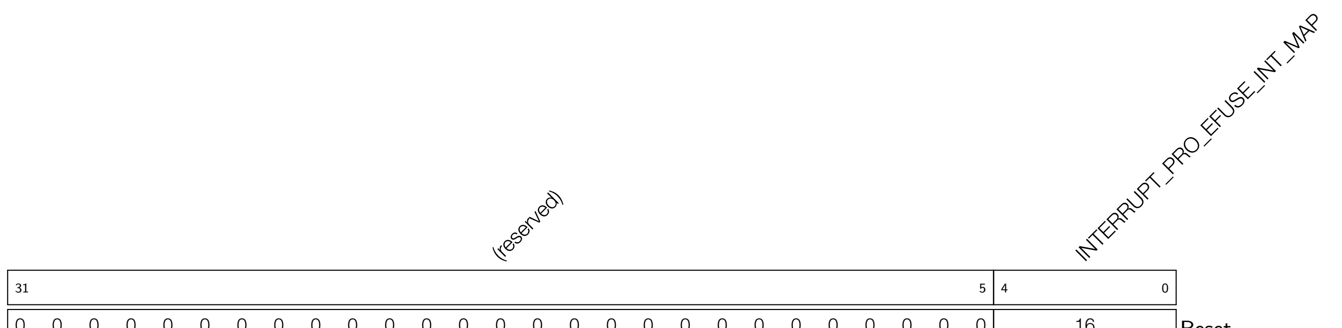
INTERRUPT_PRO_UART1_INTR_MAP This register is used to map UART1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.24: INTERRUPT_PRO_LEDC_INT_MAP_REG (0x00B4)



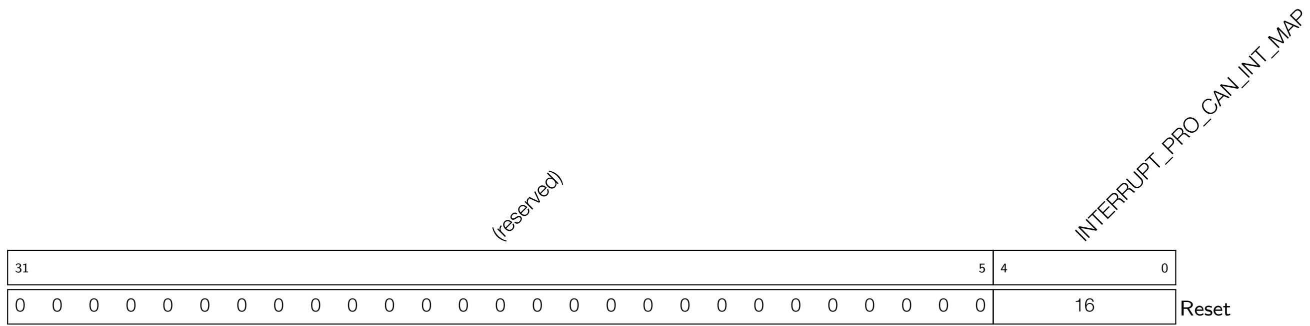
INTERRUPT_PRO_LEDC_INT_MAP This register is used to map LEDC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.25: INTERRUPT_PRO_EFUSE_INT_MAP_REG (0x00B8)



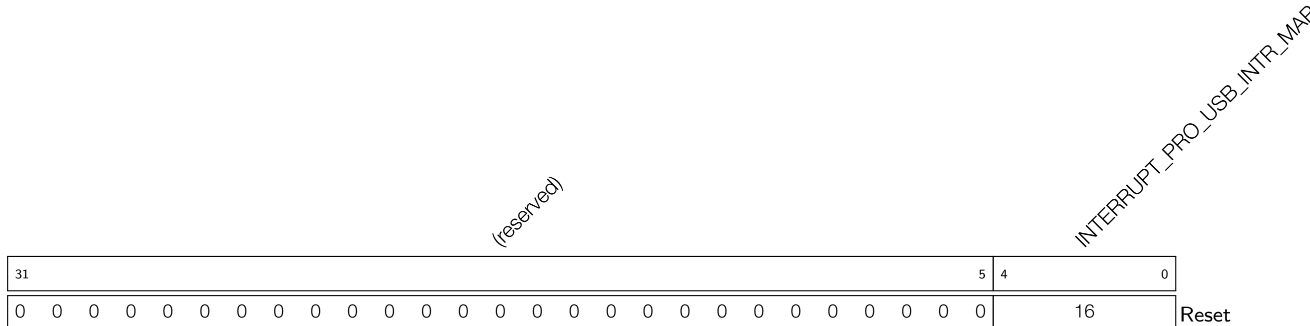
INTERRUPT_PRO_EFUSE_INT_MAP This register is used to map EFUSE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.26: INTERRUPT_PRO_CAN_INT_MAP_REG (0x00BC)



INTERRUPT_PRO_CAN_INT_MAP This register is used to map CAN_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.27: INTERRUPT_PRO_USB_INTR_MAP_REG (0x00C0)

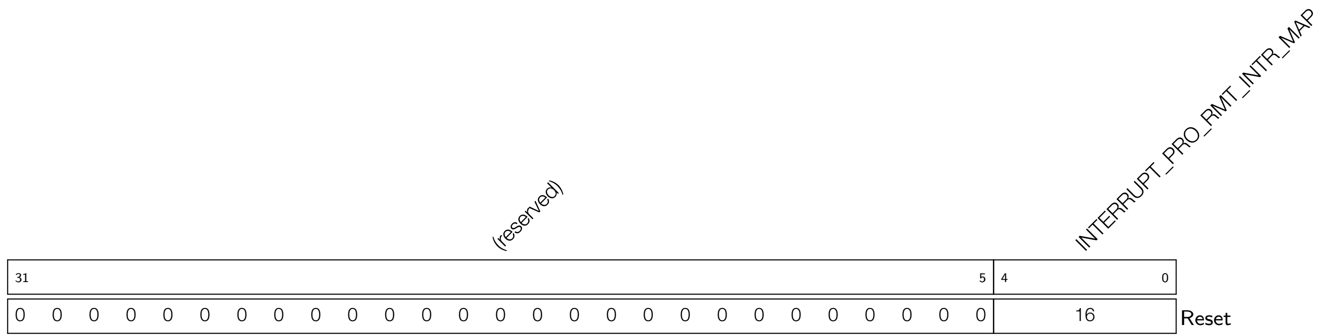


INTERRUPT_PRO_USB_INTR_MAP This register is used to map USB_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.28: INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG (0x00C4)

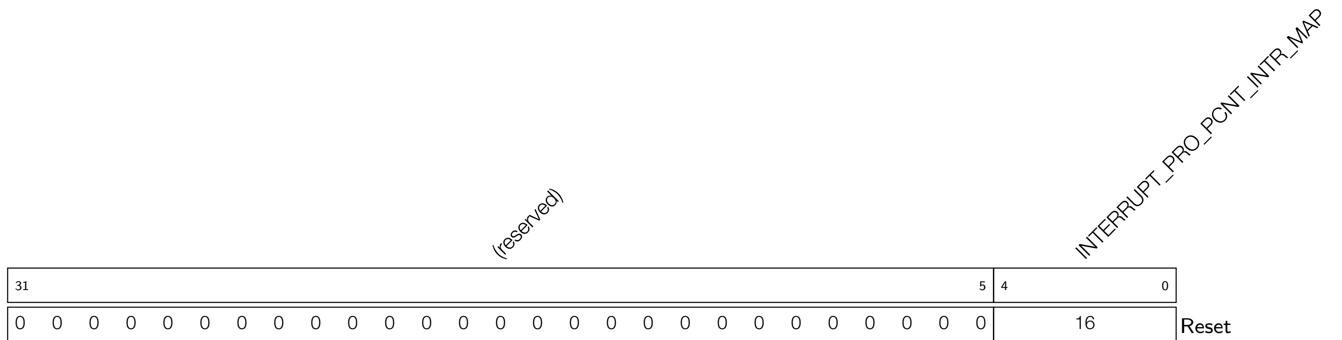
INTERRUPT_PRO_RTC_CORE_INTR_MAP This register is used to map RTC_CORE_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.29: INTERRUPT_PRO_RMT_INTR_MAP_REG (0x00C8)



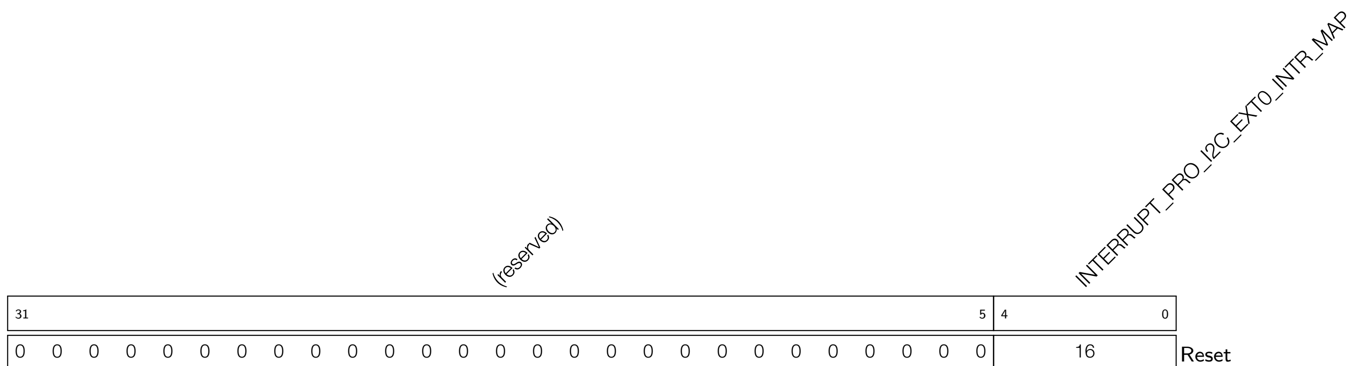
INTERRUPT_PRO_RMT_INTR_MAP This register is used to map RMT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.30: INTERRUPT_PRO_PCNT_INTR_MAP_REG (0x00CC)



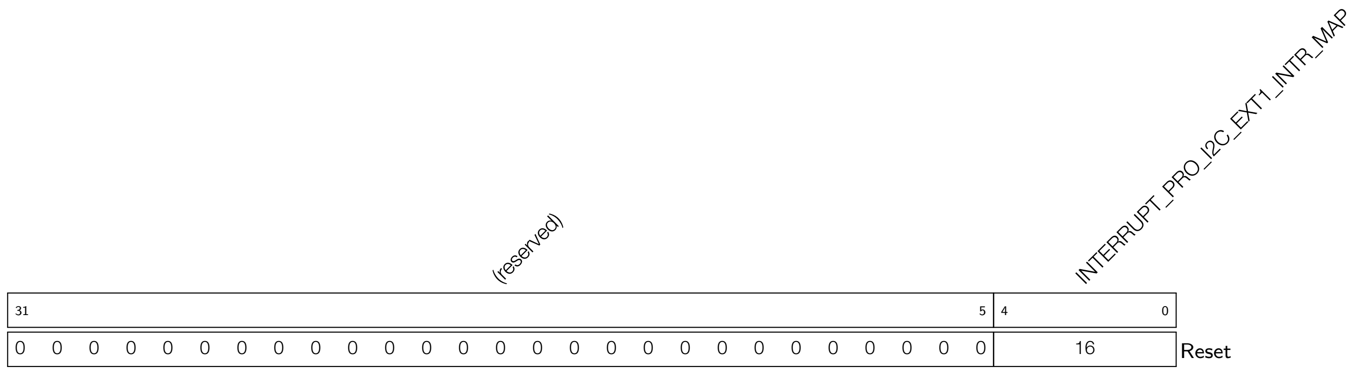
INTERRUPT_PRO_PCNT_INTR_MAP This register is used to map PCNT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.31: INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG (0x00D0)



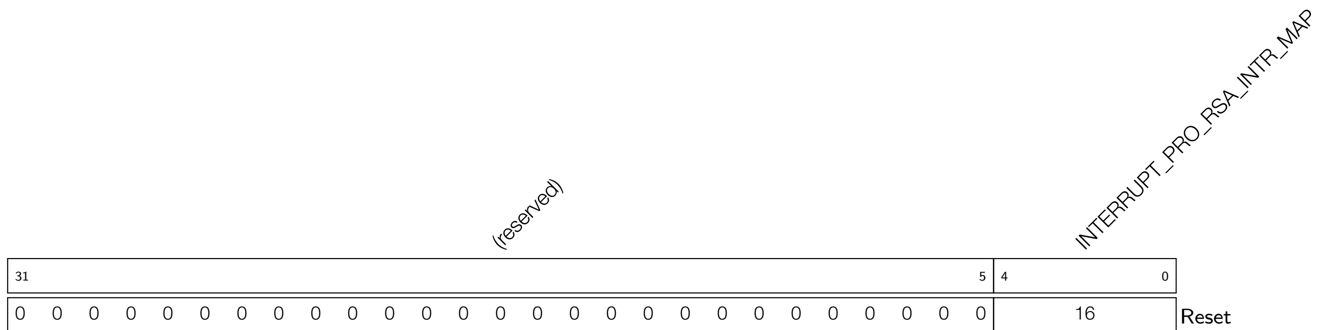
INTERRUPT_PRO_I2C_EXT0_INTR_MAP This register is used to map I2C_EXT0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.32: INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG (0x00D4)



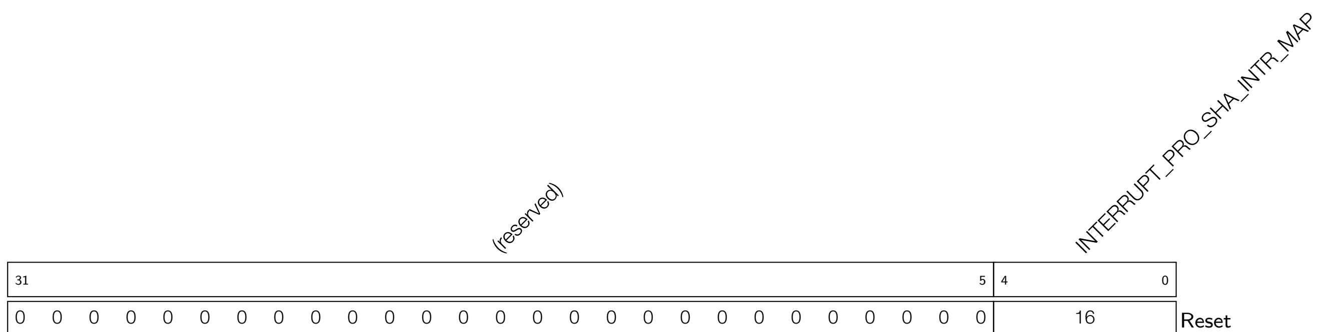
INTERRUPT_PRO_I2C_EXT1_INTR_MAP This register is used to map I2C_EXT1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.33: INTERRUPT_PRO_RSA_INTR_MAP_REG (0x00D8)



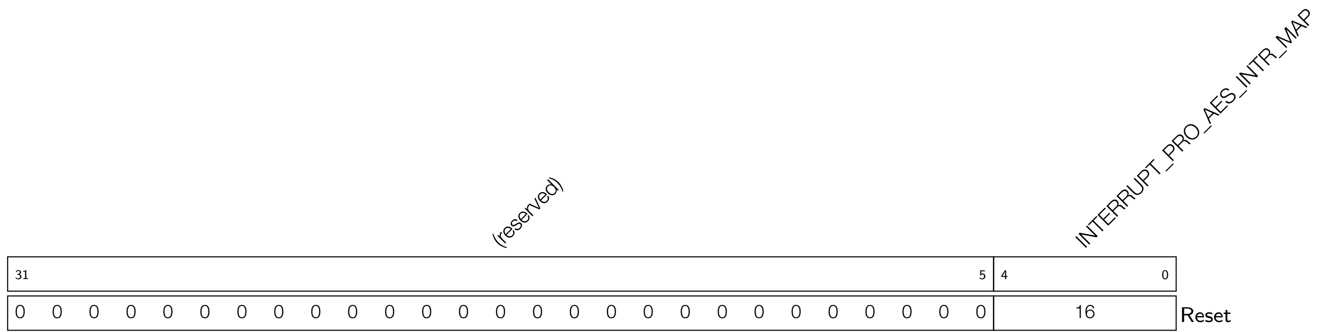
INTERRUPT_PRO_RSA_INTR_MAP This register is used to map RSA_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.34: INTERRUPT_PRO_SHA_INTR_MAP_REG (0x00DC)



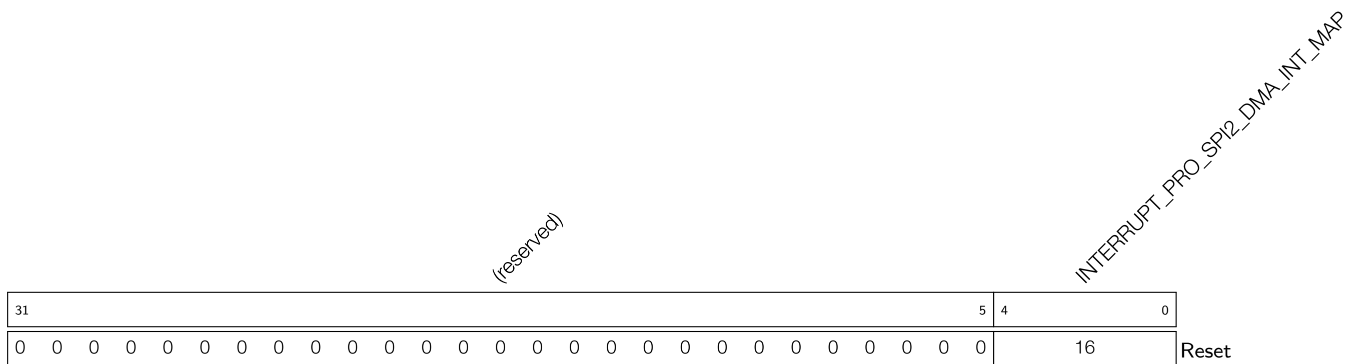
INTERRUPT_PRO_SHA_INTR_MAP This register is used to map SHA_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.35: INTERRUPT_PRO_AES_INTR_MAP_REG (0x00E0)



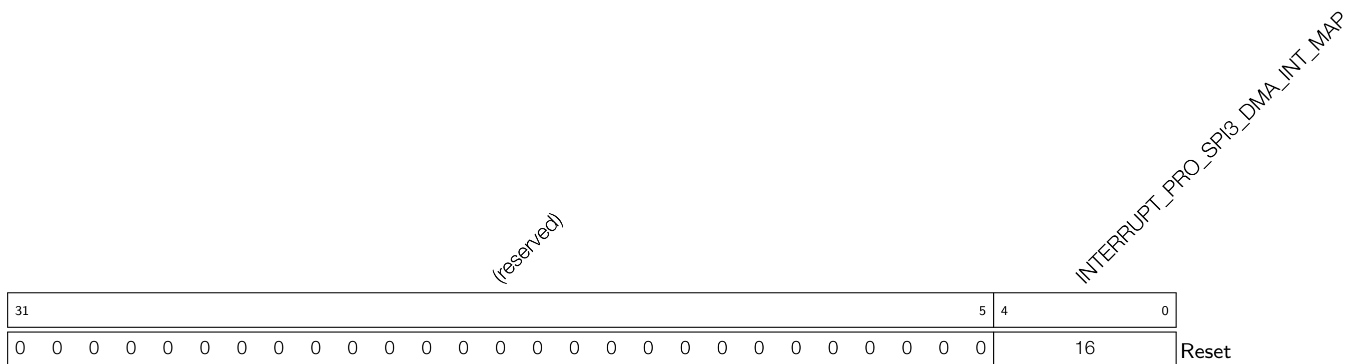
INTERRUPT_PRO_AES_INTR_MAP This register is used to map AES_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.36: INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG (0x00E4)



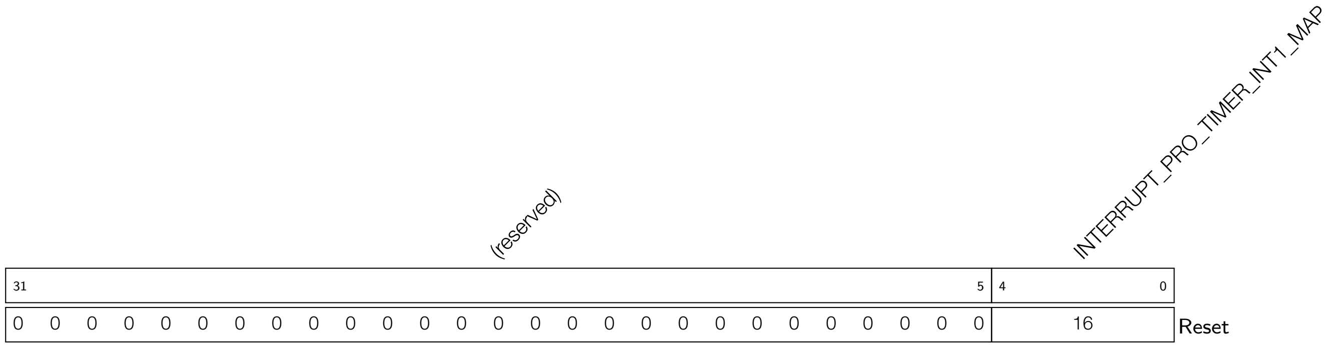
INTERRUPT_PRO_SPI2_DMA_INT_MAP This register is used to map SPI2_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.37: INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG (0x00E8)



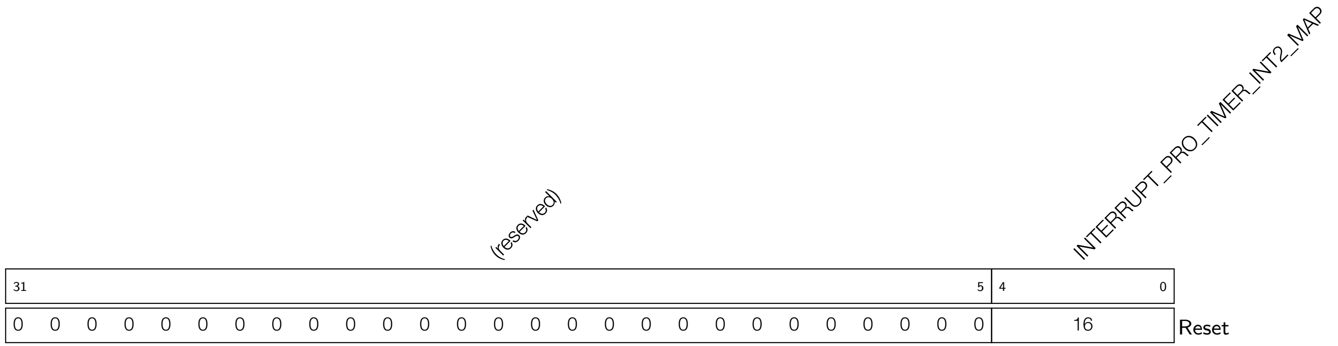
INTERRUPT_PRO_SPI3_DMA_INT_MAP This register is used to map SPI3_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.38: INTERRUPT_PRO_TIMER_INT1_MAP_REG (0x00F0)



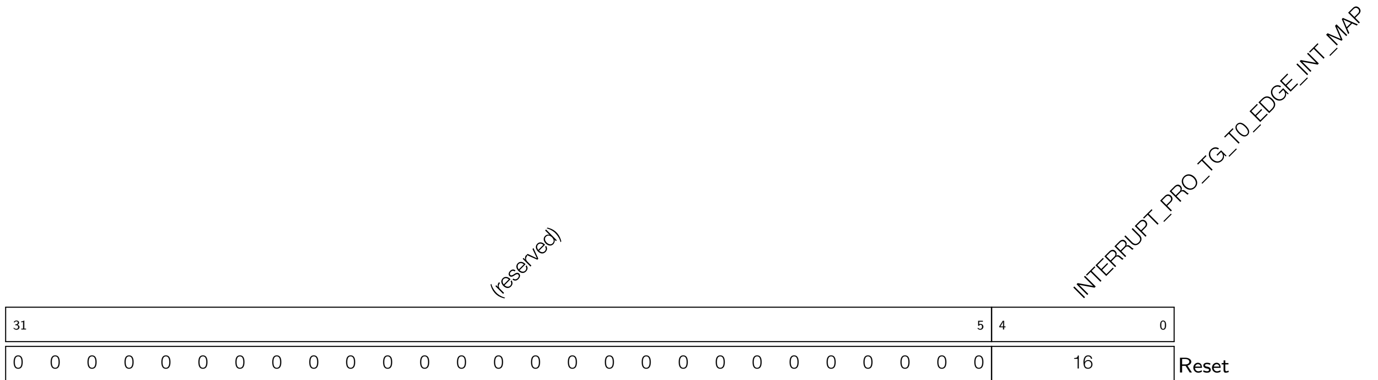
INTERRUPT_PRO_TIMER_INT1_MAP This register is used to map TIMER_INT1 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.39: INTERRUPT_PRO_TIMER_INT2_MAP_REG (0x00F4)



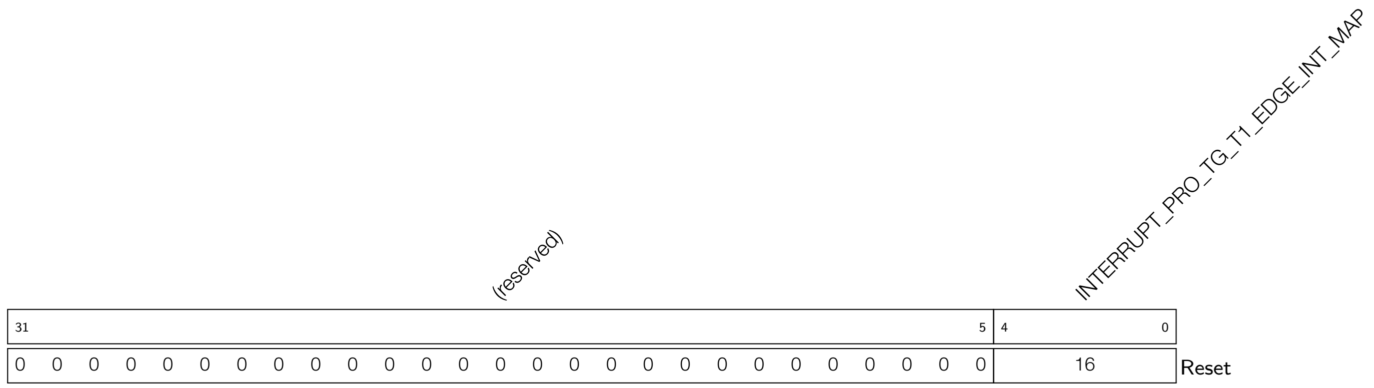
INTERRUPT_PRO_TIMER_INT2_MAP This register is used to map TIMER_INT2 interrupt signal to one of the CPU interrupts. (R/W)

Register 8.40: INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG (0x00F8)



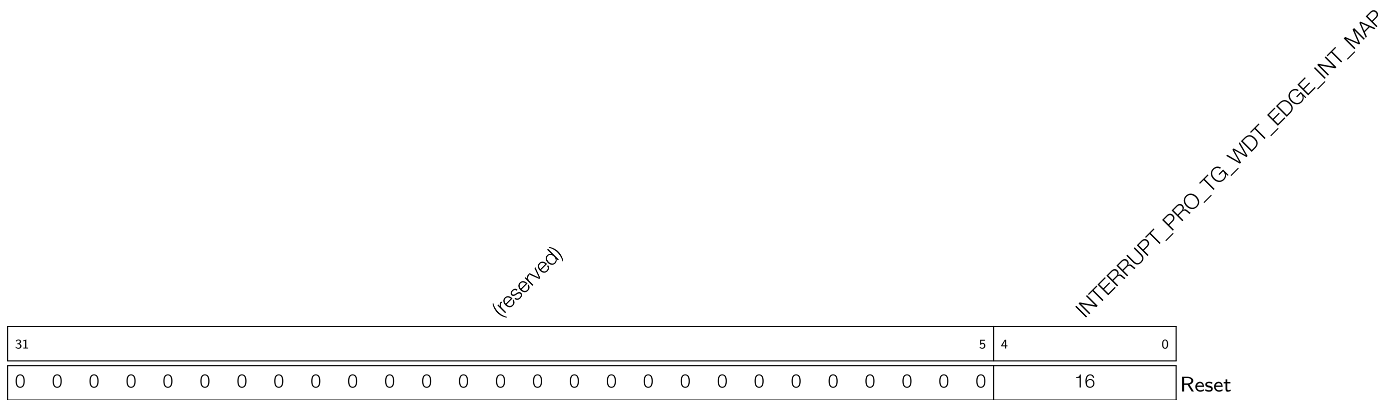
INTERRUPT_PRO_TG_T0_EDGE_INT_MAP This register is used to map TG_T0_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.41: INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG (0x00FC)



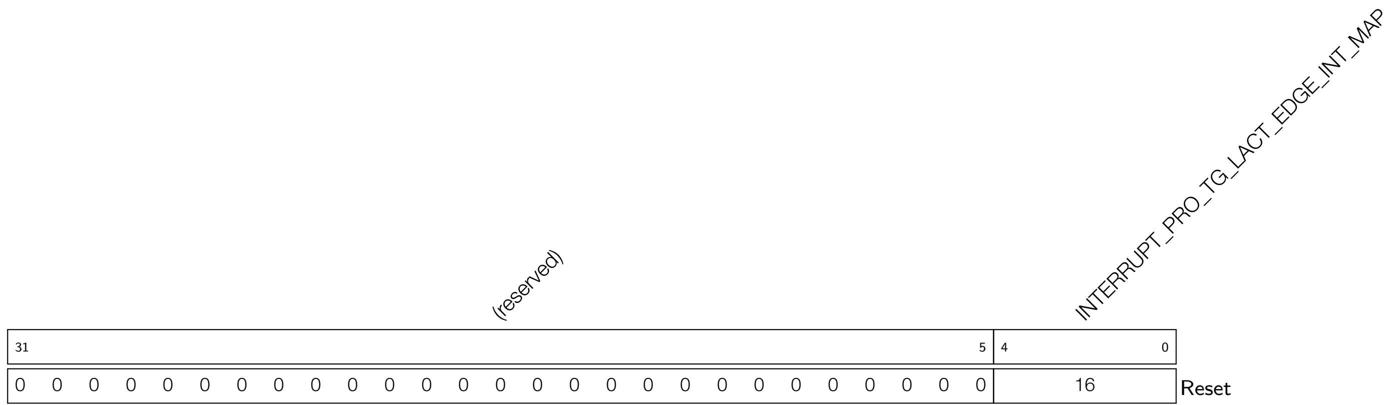
INTERRUPT_PRO_TG_T1_EDGE_INT_MAP This register is used to map TG_T1_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.42: INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG (0x0100)



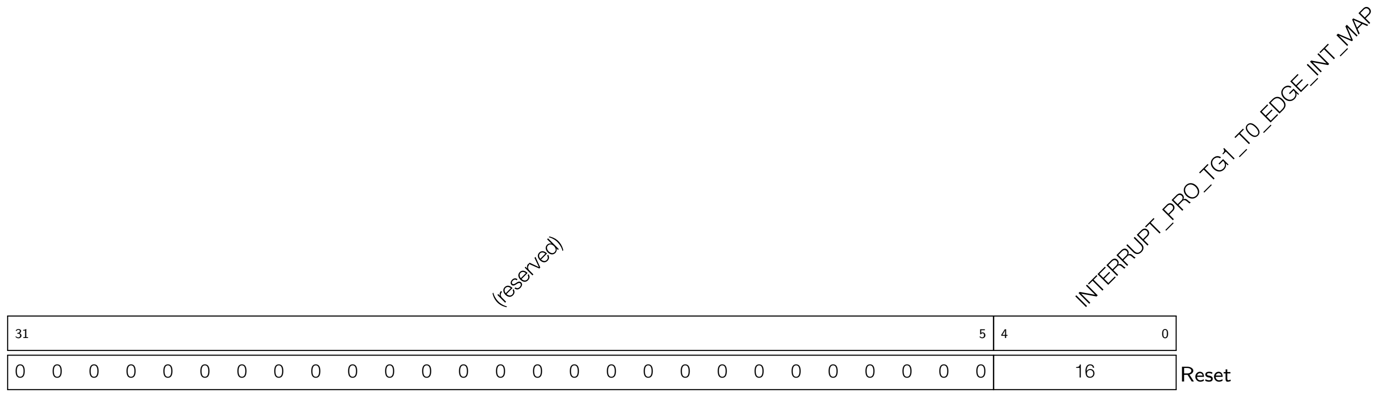
INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP This register is used to map TG_WDT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.43: INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG (0x0104)



INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP This register is used to map TG_LACT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.44: INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG (0x0108)



INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP This register is used to map TG1_T0_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.45: INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG (0x010C)

(reserved) 31–5	INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP 4–0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP This register is used to map TG1_T1_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.46: INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG (0x0110)

(reserved) 31–5	INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP 4–0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP This register is used to map TG1_WDT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.47: INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG (0x0114)

31 ... 5	4 ... 0	
(reserved)	INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP This register is used to map TG1_LACT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.48: INTERRUPT_PRO_CACHE_IA_INT_MAP_REG (0x0118)

31 ... 5	4 ... 0	
(reserved)	INTERRUPT_PRO_CACHE_IA_INT_MAP	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

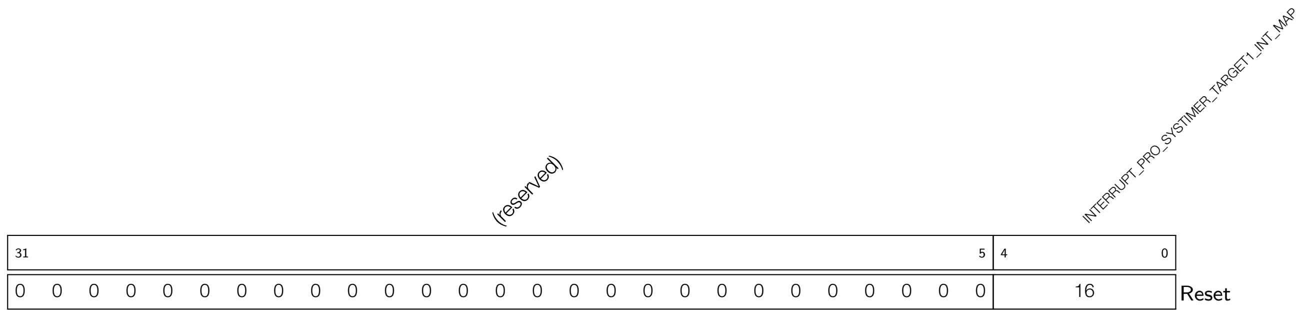
INTERRUPT_PRO_CACHE_IA_INT_MAP This register is used to map CACHE_IA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.49: INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG (0x011C)

31 ... 5	4 ... 0	
(reserved)	INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

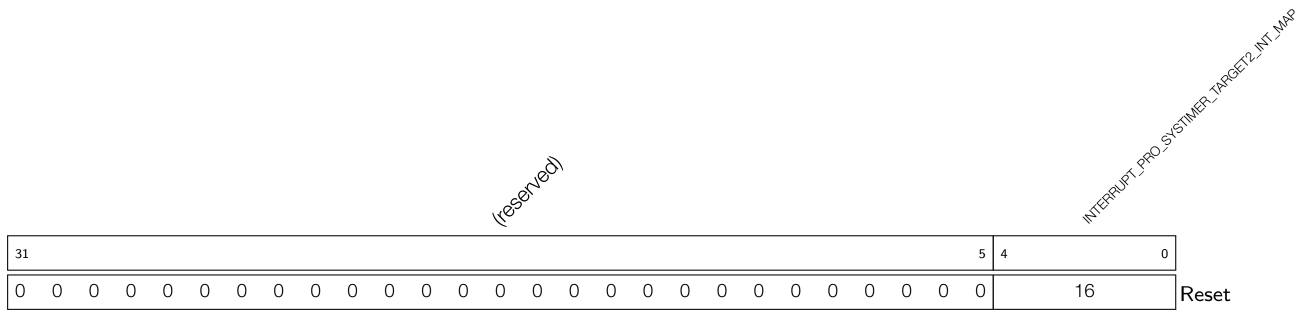
INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP This register is used to map SYSTIMER_TARGET0_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.50: INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG (0x0120)



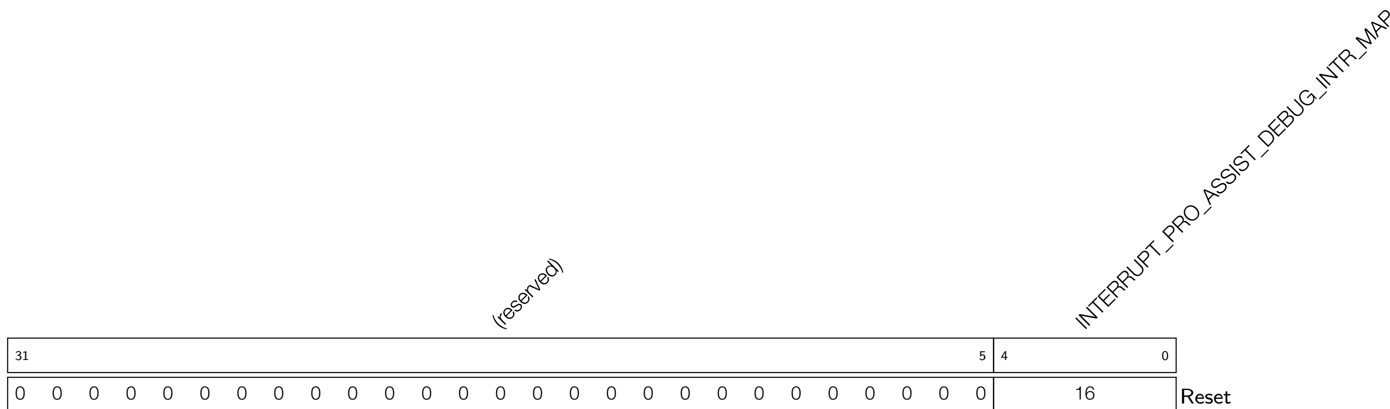
INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP This register is used to map SYSTIMER_TARGET1_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.51: INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG (0x0124)



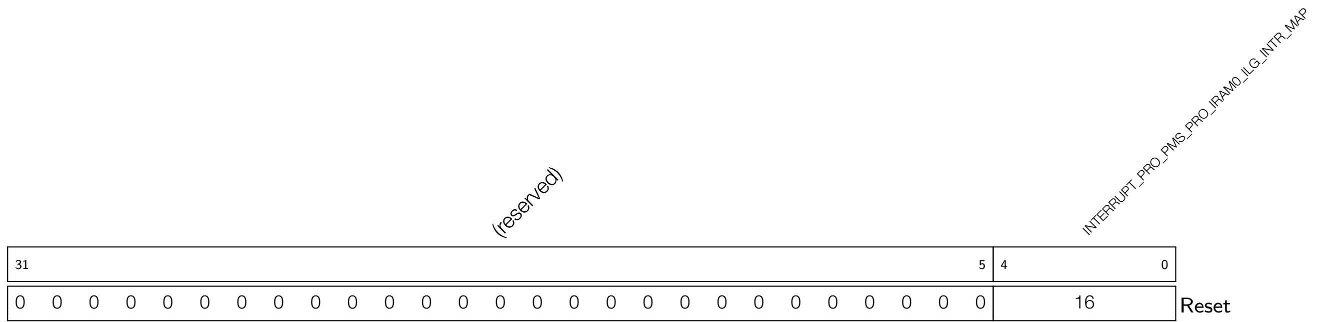
INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP This register is used to map SYSTIMER_TARGET2_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.52: INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG (0x0128)



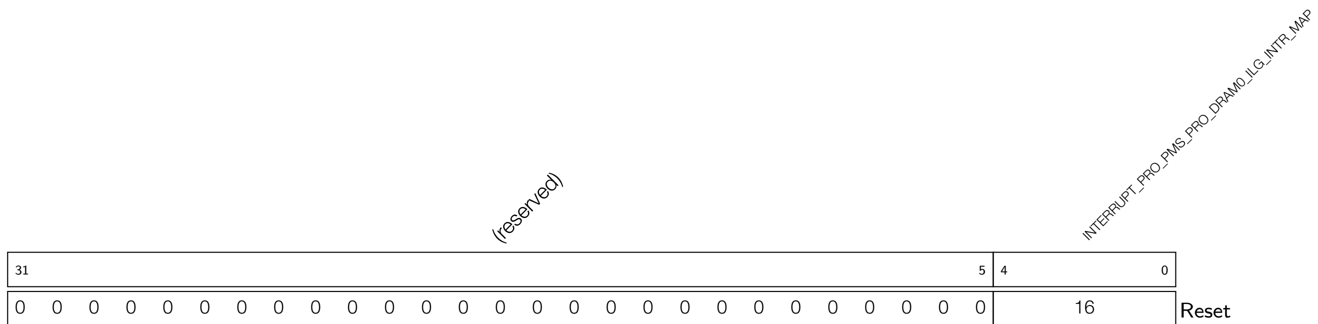
INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP This register is used to map ASSIST_DEBUG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.53: INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG (0x012C)



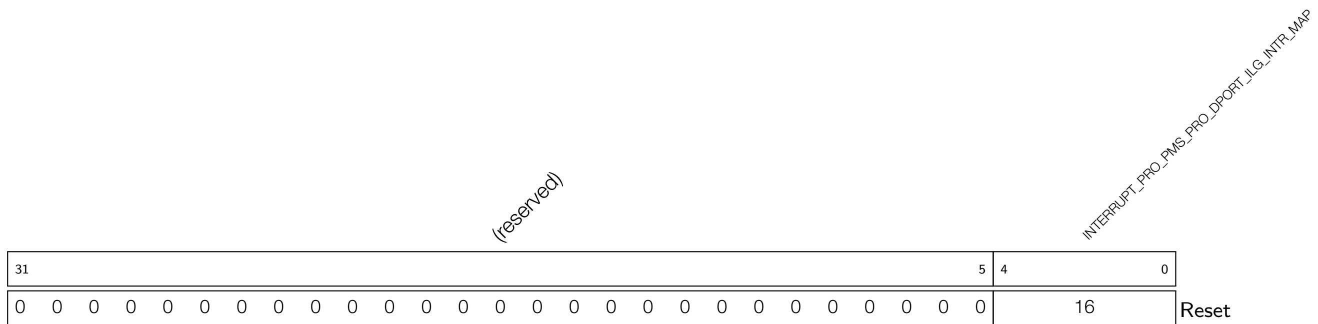
INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP This register is used to map PMS_PRO_IRAM0_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.54: INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG (0x0130)



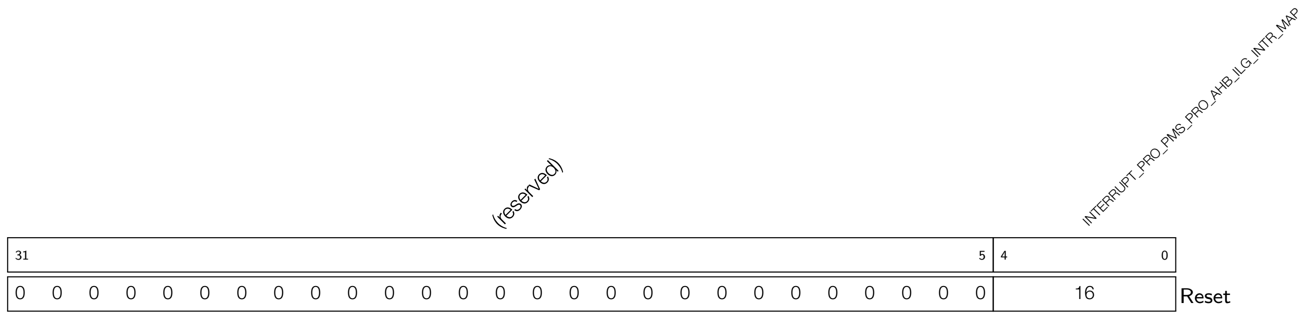
INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP This register is used to map PMS_PRO_DRAM0_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.55: INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG (0x0134)



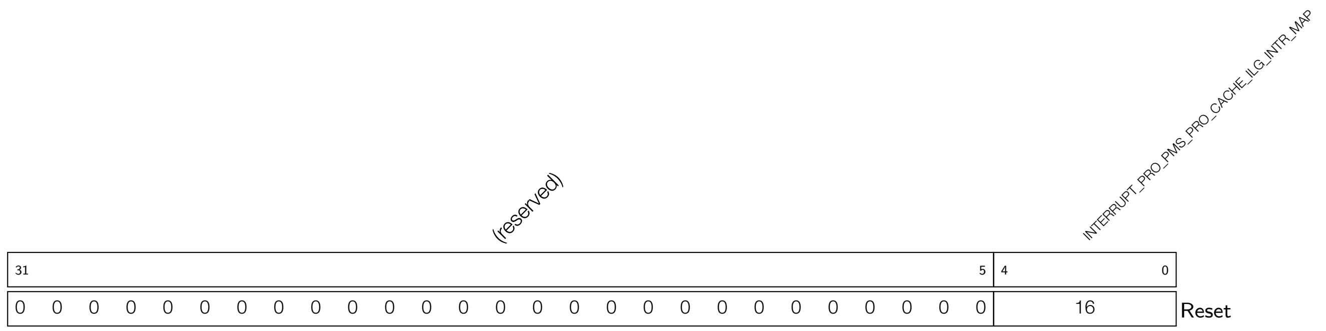
INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP This register is used to map PMS_PRO_DPORT_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.56: INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG (0x0138)



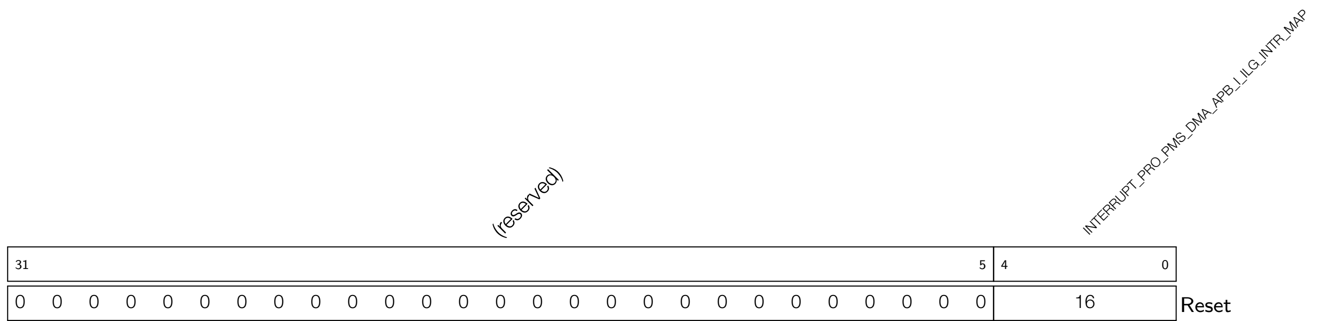
INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP This register is used to map PMS_PRO_AHB_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.57: INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG (0x013C)



INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP This register is used to map PMS_PRO_CACHE_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.58: INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG (0x0140)



INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP This register is used to map PMS_DMA_APB_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.59: INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG (0x0144)

(reserved)	INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP This register is used to map PMS_DMA_RX_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.60: INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG (0x0148)

(reserved)	INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

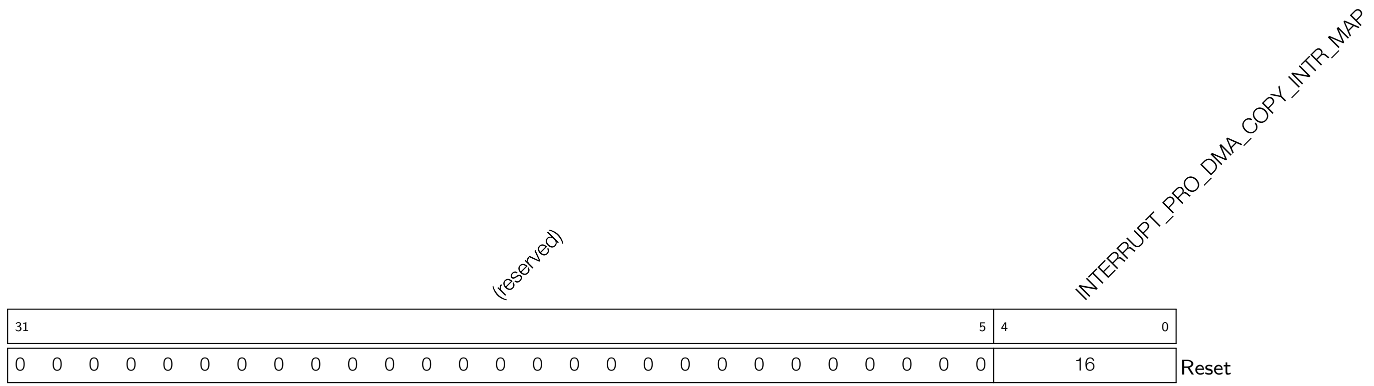
INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP This register is used to map PMS_DMA_TX_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.61: INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG (0x014C)

(reserved)	INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

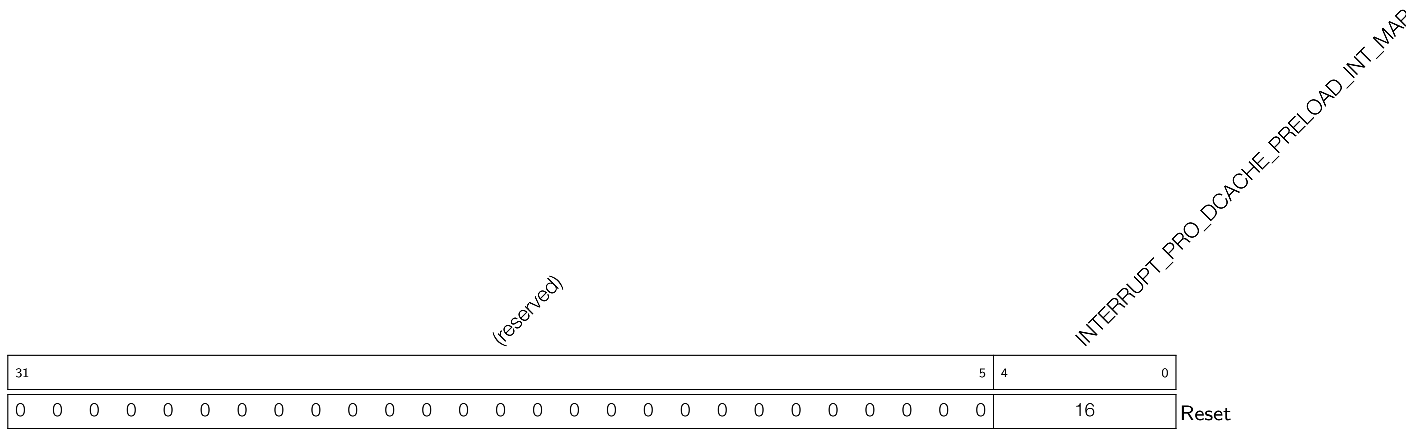
INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP This register is used to map SPI_MEM_REJECT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.62: INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG (0x0150)



INTERRUPT_PRO_DMA_COPY_INTR_MAP This register is used to map DMA_COPY_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 8.63: INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG (0x015C)



INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP This register is used to map DCACHE_PRELOAD_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.64: INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG (0x0160)

(reserved)	INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

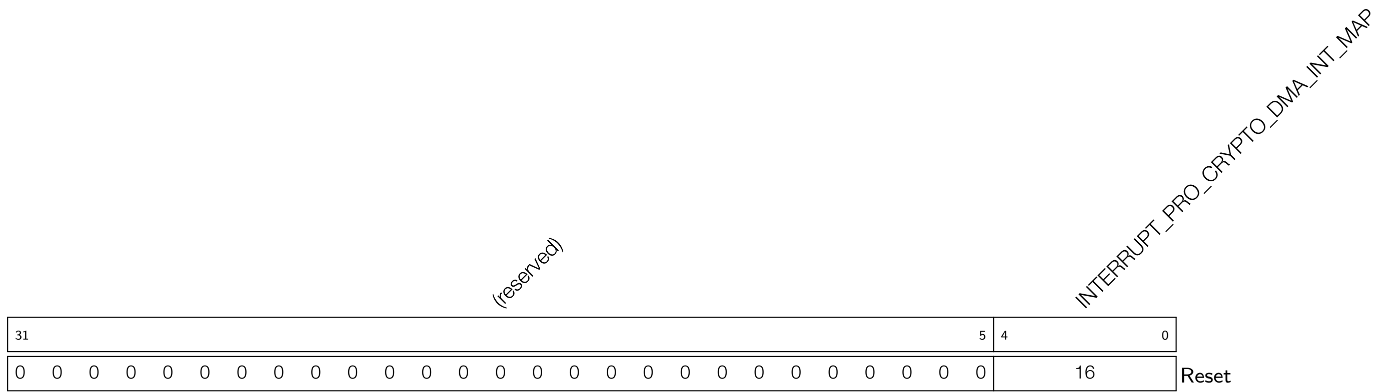
INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP This register is used to map ICACHE_PRELOAD_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.65: INTERRUPT_PRO_APB_ADC_INT_MAP_REG (0x0164)

(reserved)	INTERRUPT_PRO_APB_ADC_INT_MAP	
31 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

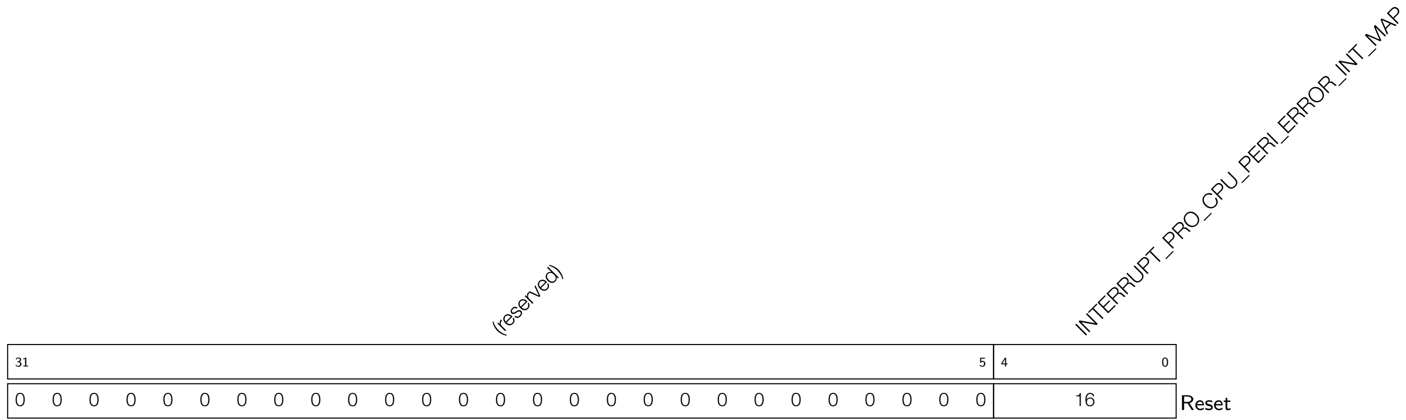
INTERRUPT_PRO_APB_ADC_INT_MAP This register is used to map APB_ADC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.66: INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG (0x0168)



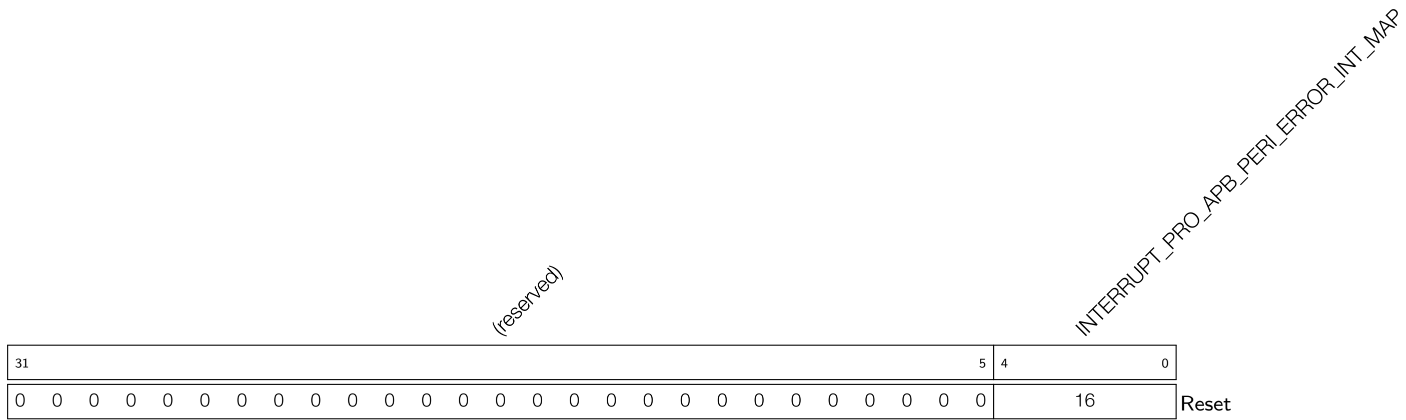
INTERRUPT_PRO_CRYPTO_DMA_INT_MAP This register is used to map CRYPTO_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.67: INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG (0x016C)



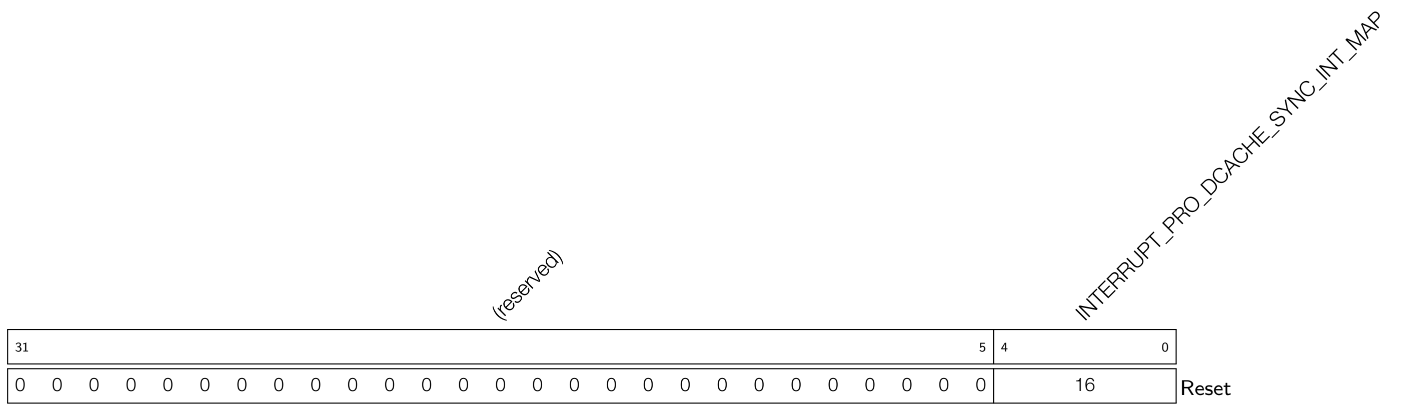
INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP This register is used to map CPU_PERI_ERROR_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.68: INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG (0x0170)



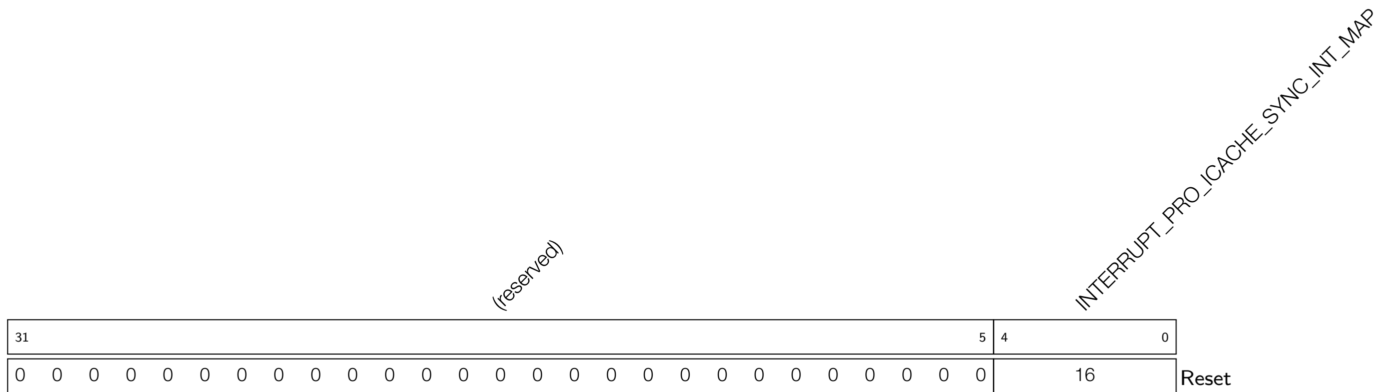
INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP This register is used to map APB_PERI_ERROR_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.69: INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG (0x0174)



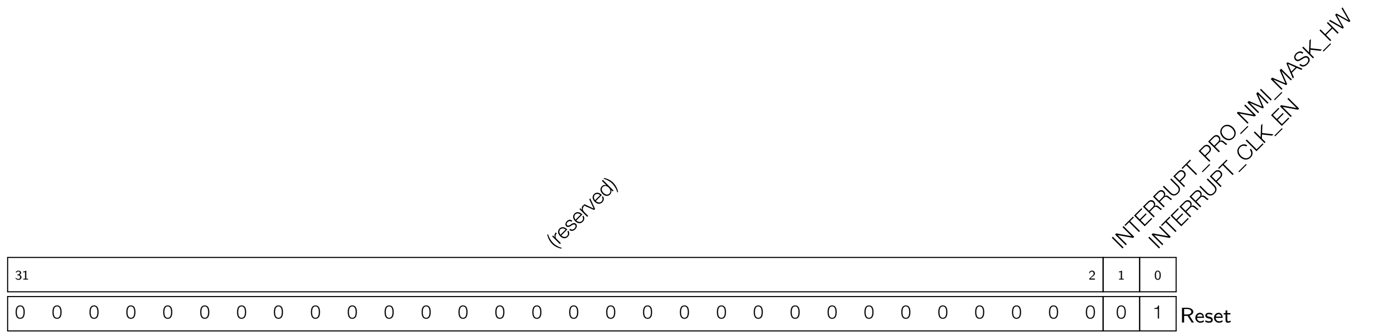
INTERRUPT_PRO_DCACHE_SYNC_INT_MAP This register is used to map DCACHE_SYNC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 8.70: INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG (0x0178)



INTERRUPT_PRO_ICACHE_SYNC_INT_MAP This register is used to map ICACHE_SYNC_INT interrupt signal to one of the CPU interrupts. (R/W)

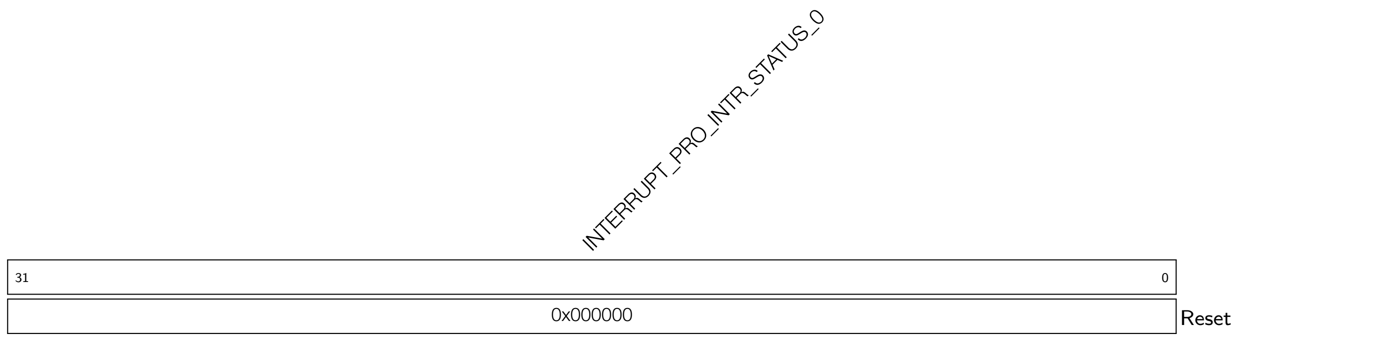
Register 8.71: INTERRUPT_CLOCK_GATE_REG (0x0188)



INTERRUPT_CLK_EN This bit is used to enable or disable the clock of interrupt matrix. 1: enable the clock; 0: disable the clock. (R/W)

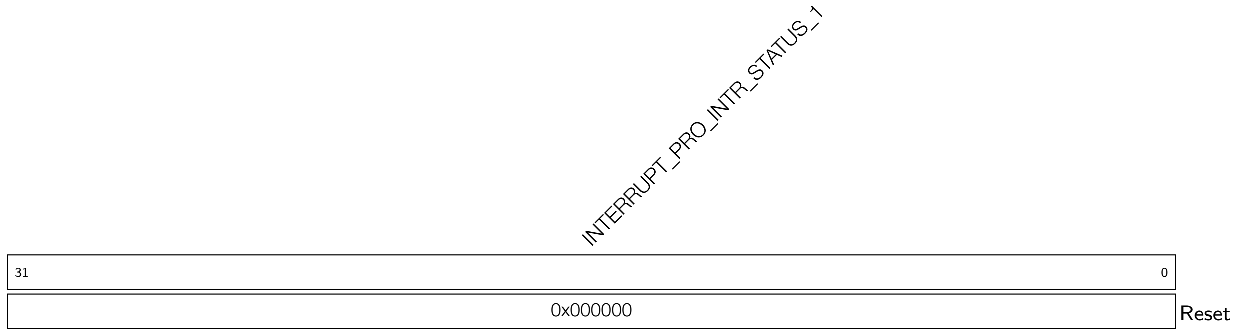
INTERRUPT_PRO_NMI_MASK_HW This bit is used to disable all NMI interrupt signals to CPU. (R/W)

Register 8.72: INTERRUPT_PRO_INTR_STATUS_REG_0_REG (0x017C)



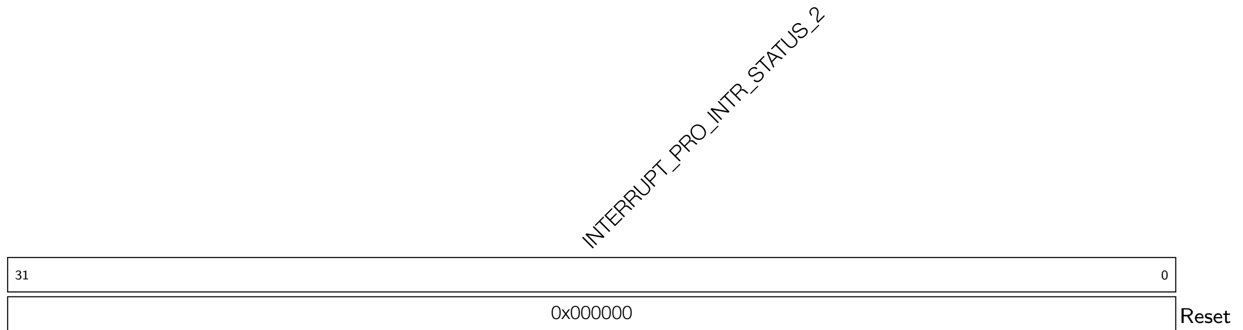
INTERRUPT_PRO_INTR_STATUS_0 This register stores the status of the first 32 input interrupt sources. (RO)

Register 8.73: INTERRUPT_PRO_INTR_STATUS_REG_1_REG (0x0180)



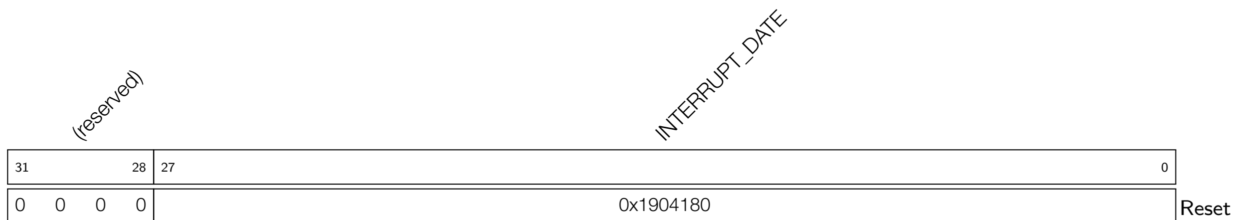
INTERRUPT_PRO_INTR_STATUS_1 This register stores the status of the second 32 input interrupt sources. (RO)

Register 8.74: INTERRUPT_PRO_INTR_STATUS_REG_2_REG (0x0184)



INTERRUPT_PRO_INTR_STATUS_2 This register stores the status of the last 31 input interrupt sources. (RO)

Register 8.75: INTERRUPT_DATE_REG (0x0FFC)



INTERRUPT_DATE Version control register. (R/W)

9. Low-Power Management (RTC_CNTL)

9.1 Introduction

ESP32-S2 has an advanced Power Management Unit (PMU), which can flexibly power up different power domains of the chip, to achieve the best balance among chip performance, power consumption, and wakeup latency. To simplify power management for typical scenarios, ESP32-S2 has predefined five power modes, which are preset configurations that power up different combinations of power domains. On top of that, the chip also allows the users to independently power up any particular power domain to meet more complex requirements. ESP32-S2 has integrated two Ultra-Low-Power co-processors (ULP co-processors), which allow the chip to work when most of the power domains are powered down, thus achieving extremely low-power consumption.

9.2 Features

ESP32-S2's low-power management supports the following features:

- ULP co-processors supported in all power modes
- Five predefined power modes to simplify power management for typical scenarios
- Up to 16 KB of retention memory
- 8 x 32-bit retention registers
- RTC Boot supported for reduced wakeup latency

In this chapter, we first introduce the working process of ESP32-S2's low-power management, then introduce the predefined power modes of the chip, and at last, introduce the RTC boot of the chip.

9.3 Functional Description

ESP32-S2's low-power management involves the following components:

- Power management unit: controls the power supply to Analog, RTC and Digital power domains.
- Power isolation unit: isolates different power domains, so any powered down power domain does not affect the powered up ones.
- Low-power clocks: provide clocks to power domains working in low-power modes.
- Timers:
  - RTC timer: logs the status of the RTC main state machine in dedicated registers.
  - ULP timer: wakes up the ULP co-processors at a predefined time. For details, please refer to Chapter 1 *ULP Coprocessor (ULP)*.
  - Touch sensor timer: wakes up the touch sensor at a predefined time. For details, please refer to Chapter 32 *On-Chip Sensor and Analog Signal Processing*.
- 8 x 32-bit "always-on" retention registers: These registers are always powered up and can be used for storing data that cannot be lost.
- 22 x "always-on" pins: These pins are always powered up and can be used as wakeup sources when the chip is working in the low-power modes (for details, please refer to Section 9.4.4), or can be used as

regular GPIOs (for details, please refer to Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*).

- RTC slow memory: supports 8 KB SRAM, which can be used as retention memory or to store ULP directives and data.
- RTC fast memory: supports 8 KB SRAM, which can be used as retention memory.
- Regulators: regulate the power supply to different power domains.

The schematic diagram of ESP32-S2's low-power management is shown in Figure 9-1.

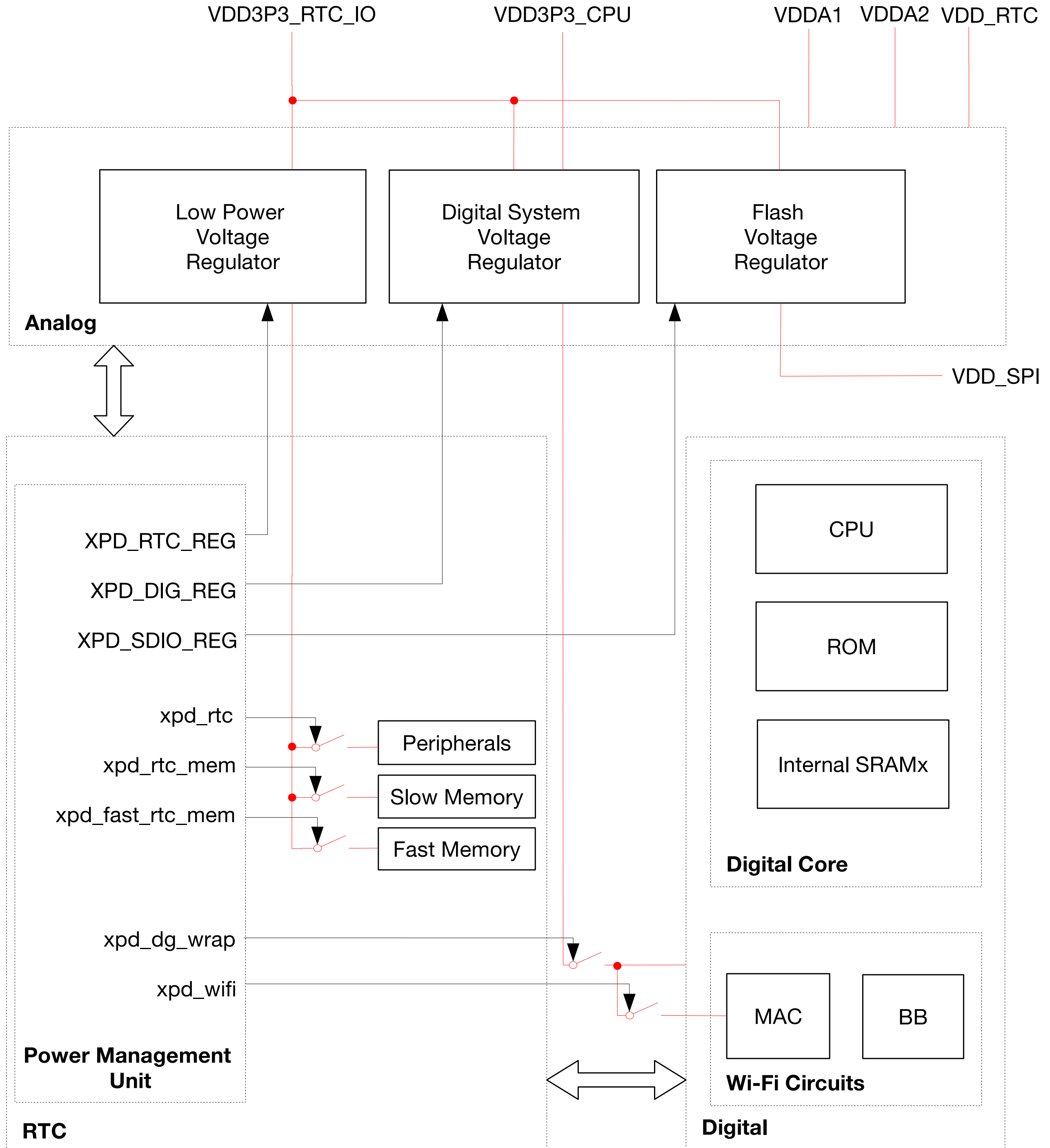


Figure 9-1. Low-power Management Schematics

9.3.1 Power Management Unit

ESP32-S2's power management unit controls the power supply to different power domains. The main components of the power management unit include:

- RTC main state machine: generates power gating, clock gating, and reset signals.
- Power controllers: power up and power down different power domains, according to the power gating signals from the main state machine.
- Sleep / wakeup controllers: send sleep or wakeup requests to the RTC main state machine.
- Clock controller: selects and powers up / down clock sources.

In ESP32-S2's power management unit, the sleep / wakeup controllers send sleep or wakeup requests to the RTC main state machine, which then generates power gating, clock gating, and reset signals. Then, the power controller and clock controller power up and power down different power domains and clock sources, according to the signals generated by the RTC main state machine, so that the chip enters or exits the low-power modes. The main workflow is shown in Figure 9-2.

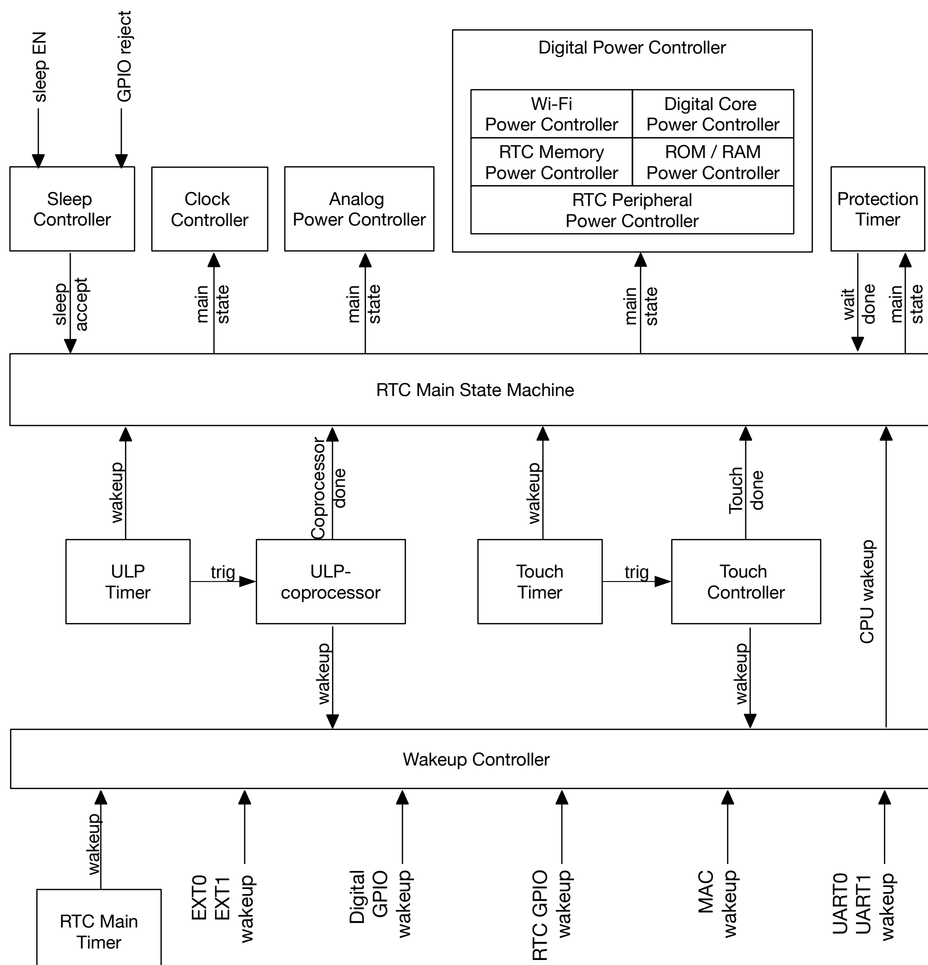


Figure 9-2. Power Management Unit Workflow

Note:
For more detailed description about power domains, please refer to 9.4.2.

9.3.2 Low-Power Clocks

In general, ESP32-S2 powers down its 40 MHz crystal oscillator and PLL to reduce power consumption when working in low-power modes. During this time, the chip's low-power clocks remain on to provide clocks to different power domains, such as the power management unit, RTC peripherals, RTC fast memory, RTC slow memory, and Wi-Fi digital circuits in the digital domain.

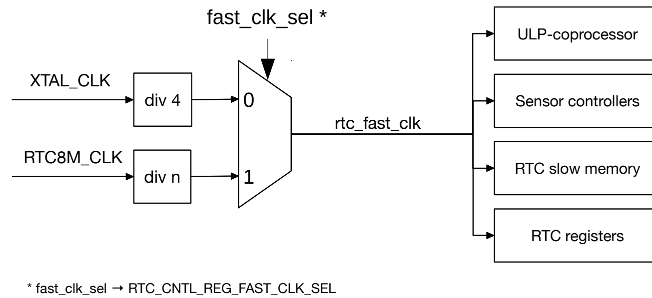


Figure 9-3. Fast Clocks for RTC Power Domains

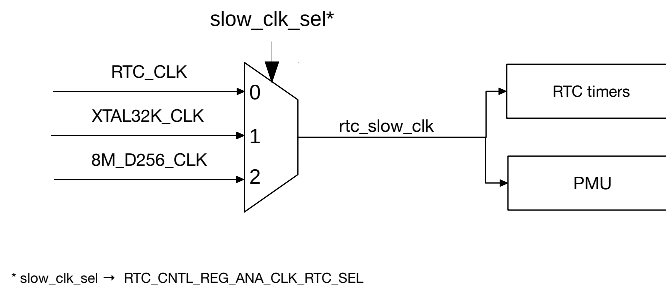


Figure 9-4. Slow Clocks for RTC Power Domains

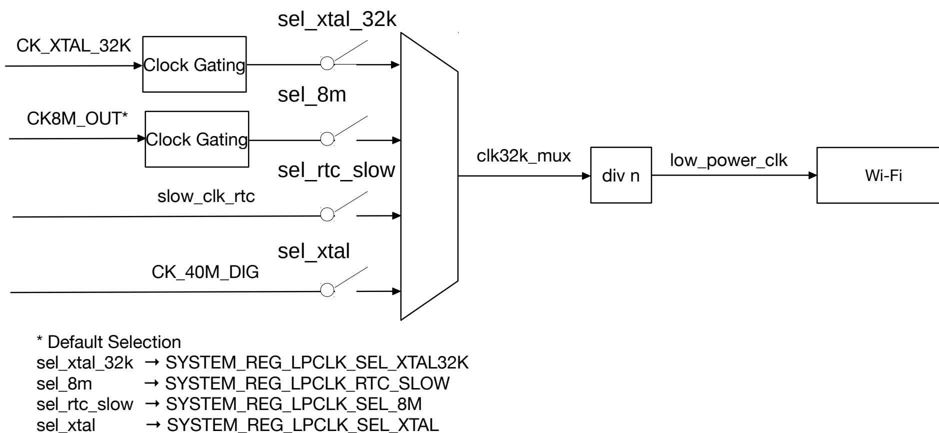


Figure 9-5. Low-Power Clocks for RTC Power Domains

Table 59: Low-Power Clocks

Clock Type	Clock Source	Selection Option	Power Domain
RTC fast clock	RTC8M_CLK divided by n [1]	RTC_CNTL_FAST_CLK_RTC_SEL	RTC peripherals
	XTAL_CLK divided by 4		RTC fast memory
			RTC slow memory
RTC slow clock	XTAL32K_CLK	RTC_CNTL_ANA_CLK_RTC_SEL	Power management unit
	RTC8M_D256_CLK		
	RTC_CLK [2]		
Low-Power clocks	XTAL32K_CLK	SYSTEM_LPCLK_SEL_XTAL32K	Digital system (Wi-Fi) in low-power modes
	RTC8M_CLK	SYSTEM_LPCLK_SEL_8M	
	RTC_CLK	SYSTEM_LPCLK_RTC_SLOW	
	XTAL_CLK	SYSTEM_LPCLK_SEL_XTAL	

Note:

1. The default RTC fast clock source.
2. The default RTC slow clock source.

For more detailed description about clocks, please refer to 6 *Reset and Clock*.

9.3.3 Timers

ESP32-S2's low-power management uses 3 timers:

- RTC timer
- ULP timer
- Touch sensor timer

This section only introduces the RTC timer. For detailed description of ULP timer and touch sensor timer, please refer to Chapters 1 *ULP Coprocessor (ULP)* and 32 *On-Chip Sensor and Analog Signal Processing*.

The readable 48-bit RTC timer is a real-time rr (using RTC slow clock) that can be configured to log the time when one of the following events happens. For details, see Table 60.

Table 60: The Triggering Conditions for the RTC Timer

Enabling Options	Descriptions
RTC_CNTL_TIMER_XTL_OFF	1. RTC main state machine powers down; 2. 40 MHz crystal powers up.
RTC_CNTL_TIMER_SYS_STALL	CPU enters or exits the stall state. This is to ensure the SYS_TIMER is continuous in time.
RTC_CNTL_TIMER_SYS_RST	Resetting digital system completes.
RTC_CNTL_TIME_UPDATE	Register RTC_CNTL_RTC_TIME_UPDATE is configured by CPU (i.e. users).

The RTC timer updates two groups of registers upon any new trigger. The first group logs the information of the current trigger, and the other logs the previous trigger. Detailed information about these two register groups is

shown below:

- Register group 0: logs the status of RTC timer at the current trigger.
  - RTC_CNTL_TIME_HIGH0_REG
  - RTC_CNTL_TIME_LOW0_REG
- Register group 1: logs the status of RTC timer at the previous trigger.
  - RTC_CNTL_TIME_HIGH1_REG
  - RTC_CNTL_TIME_LOW1_REG

On a new trigger, information on previous trigger is moved from register group 0 to register group 1 (and the original trigger logged in register group 1 is overrode), and this new trigger is logged in register group 0. Therefore, only the last two triggers can be logged at any time.

It should be noted that any reset / sleep other than power-up reset will not stop or reset the RTC timer.

Also, the RTC timer can be used as a wakeup source. For details, see Section 9.4.4.

9.3.4 Regulators

ESP32-S2 has three regulators to regulate the power supply to different power domains:

- Digital system voltage regulator for digital power domains;
- Low-power voltage regulator for RTC power domains;
- Flash voltage regulator for the rest of power domains.

> **Note:**
> For more detailed description about power domains, please refer to Section 9.4.2.

9.3.4.1 Digital System Voltage Regulator

ESP32-S2's built-in digital system voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for digital power domains. This regulator has an adjustable output. For the architecture of the ESP32-S2 digital system voltage regulator, see Figure 9-6.

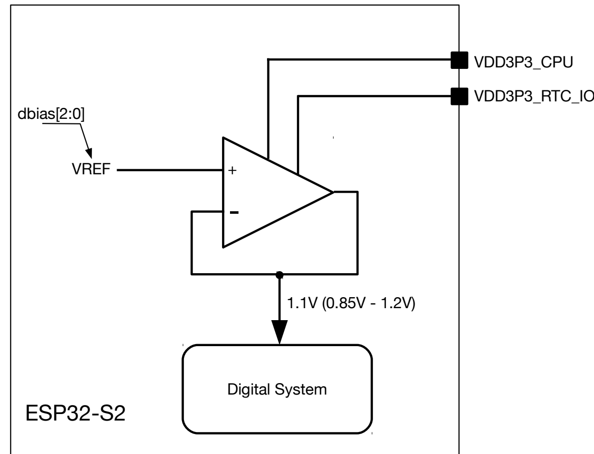


Figure 9-6. Digital System Regulator

1. When XPD_DIG_REG == 1, the regulator outputs a 1.1 V voltage and the power domains in digital system are able to run; when XPD_DIG_REG == 0, both the regulator and the power domains in digital system stop running.
2. DIG_DBIAS[2:0] tunes the supply voltage of the digital system:

$$\text{VDD_DIG} = 0.90 + \text{DBIAS} \times 0.05\text{V}$$

3. The current to power domains in digital system comes from pin VDD3P3_CPU and pin VDD3P3_RTC_IO.

9.3.4.2 Low-power Voltage Regulator

ESP32-S2's built-in low-power voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for RTC power domains. This regulator has an adjustable output to achieve lower power consumption and can output even lower voltage in Deep-sleep mode and Hibernation mode. For the architecture of the ESP32-S2 low-power voltage regulator, see Figure 9-7.

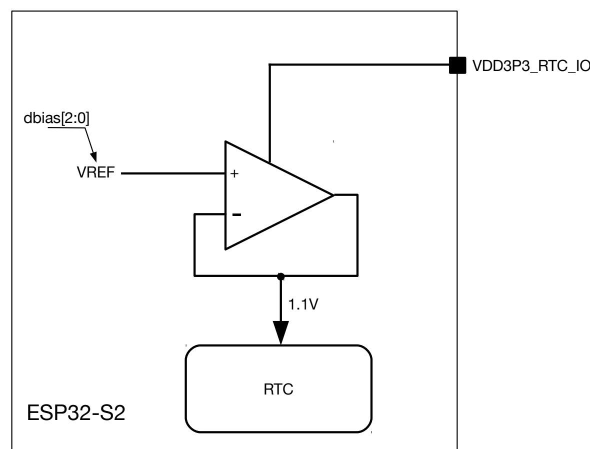


Figure 9-7. Low-power voltage regulator

1. When the pin CHIP_PU is at a high level, the low-power voltage regulator cannot be turned off, but only switching between normal-work mode and Deep-sleep mode.
2. RTC_DBIAS[2:0] can be used to tune the output voltage:

$$VDD_RTC = 0.90 + DBIAS \times 0.05V$$

3. The current to the RTC power domains comes from pin VDD3P3_RTC_IO.

9.3.4.3 Flash Voltage Regulator

ESP32-S2's built-in flash voltage regulator can supply a voltage of 3.3 V or 1.8 V to other components outside of digital system and RTC, such as flash. For the architecture of the ESP32-S2 flash voltage regulator, see Figure 9-8.

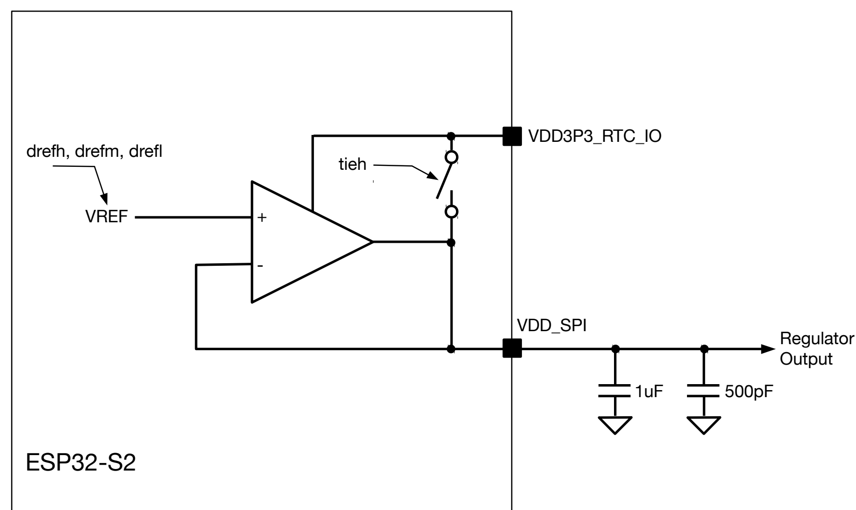


Figure 9-8. Flash voltage regulator

1. When XPD_SDIO_VREG == 1, the regulator outputs a voltage of 3.3 V or 1.8 V; when XPD_SDIO_VREG == 0, the output is high-impedance. In this case, the voltage is provided by the external power supply.
2. When SDIO_TIEH == 1, the regulator shorts pin VDD_SDIO to pin VDD3P3_RTC, and outputs a voltage of 3.3 V, which is the voltage of pin VDD3P3_RTC. When SDIO_TIEH == 0, the regulator outputs the reference voltage VREF, which is typically 1.8 V.
3. DREFH_SDIO, DREFM_SDIO and DREFL_SDIO could be used to fine tune the reference voltage VREF, but are not recommended because this fine tuning may jeopardize the stability of the inner loop.
4. When the regulator output is 3.3 V or 1.8 V, the output current comes from the pin VDD3P3_RTC_IO.

The flash voltage regulator can be configured via RTC registers or eFuse controller registers.

- The configuration of XPD_SDIO_VREG:
  - When the chip is in active mode, RTC_CNTL_SDIO_FORCE == 0 and VDD_SPI_FORCE(EFUSE) == 1, the XPD_SDIO_VREG voltage is defined by XPD_VDD_SPI_REG(EFUSE);
  - When the chip is in sleep modes and RTC_CNTL_SDIO_PD_EN == 1, XPD_SDIO_VREG is 0;

    - When RTC_CNTL_SDIO_FORCE == 1, XPD_SDIO_VREG is defined by RTC_CNTL_XPD_SDIO_REG.

- The configuration of SDIO_TIEH:
    - When RTC_CNTL_SDIO_FORCE == 0 and VDD_SPI_FORCE(EFUSE) == 1, SDIO_TIEH = VDD_SDIO_TIEH(EFUSE)
    - Otherwise, SDIO_TIEH = RTC_CNTL_SDIO_TIEH.

9.3.4.4 Brownout Detector

The brownout detector checks the voltage of pins VDD3P3_RTC_IO, VDD3P3_CPU, VDDA1, and VDDA2. If the voltage of these pins drops rapidly and becomes too low, the detector would trigger a signal to shut down some power-consuming blocks (such as LNA, PA, etc.) to allow extra time for the digital system to save and transfer important data.

The brownout detector has ultra-low power consumption and remains enabled whenever the chip is powered up. For the architecture of the ESP32-S2 brownout detector, see Figure 9-9.

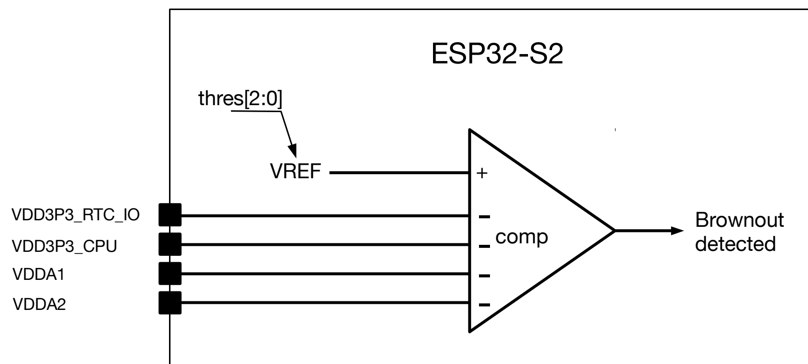


Figure 9-9. Brown-out detector

1. RTC_CNTL_RTC_BROWN_OUT_DET indicates the output level of brown-out detector. This register is low level by default, and outputs high level when the voltage of the detected pin drops below the predefined threshold;
2. I2C register ULP_CAL_REG5[2:0] configures the trigger threshold of the brown-out detector;

Table 61: Brown-out Detector Configuration

ULP_CAL_REG5[2:0]	VDD (Unit: V)
0	2.67
1	3.30
2	3.19
3	2.98
4	2.84
5	2.67
6	2.56
7	2.44

3. RTC_CNTL_BROWN_OUT_RST_SEL configures the reset type.
    - 0: resets the chip
    - 1: resets the system

Note:
For more information regarding chip reset and system reset, please refer to 6 *Reset and Clock*.

9.4 Power Modes Management

9.4.1 Power Domain

ESP32-S2 has 10 power domains in three power domain categories:

- RTC
  - Power management unit
  - RTC peripherals
  - RTC slow memory
  - RTC fast memory
- Digital
  - Digital core
  - Wi-Fi digital circuits
- Analog
  - 8 MHz crystals
  - 40 MHz crystals
  - PLL
  - RF circuits

9.4.2 RTC States

ESP32-S2 has three main RTC states: Active, Monitor, and Sleep. The transition process among these states can be seen in Figure 9-10.

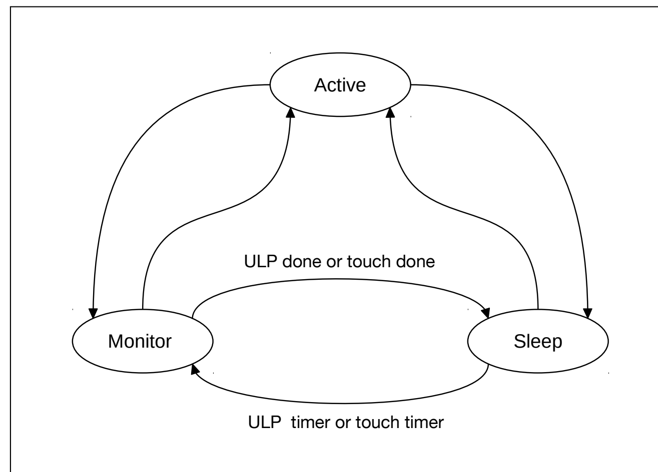


Figure 9-10. RTC States Transition

Under different RTC states, different power domains are powered up or down by default, but can also be force-powered-up (FPU) or force-powered-down (FPD) individually based on actual requirements. For details, please refer to Table 62.

Table 62: RTC Statues Transition

Power Domain		RTC States			Notes
Category	Sub-category	Active	Monitor	Sleep	
RTC	Power Management Unit	ON	ON	ON	1
	RTC Peripherals	ON	ON	OFF	2
	RTC Slow Memory	ON	OFF	OFF	3
	RTC Fast Memory	ON	OFF	OFF	4
Digital	Digital Core	ON	OFF	OFF	5
	Wi-Fi Digital Circuits	ON	OFF	OFF	6
Analog	8 MHz Crystals	ON	ON	OFF	-
	40 MHz Crystals	ON	OFF	OFF	-
	PLL	ON	OFF	OFF	-
	RF Circuits	-	-	-	-

Note:

1. ESP32-S2's power management unit is specially designed to be "always-on", which means it is always on when the chip is powered up. Therefore, users cannot FPU or FPD the power management unit.
2. The RTC peripherals include 1 *ULP Coprocessor (ULP)* and 32 *On-Chip Sensor and Analog Signal Processing* (i.e. temperature sensor controller, touch sensor, SAR ADC controller).
3. RTC slow memory supports 8 KB SRAM, which can be used to reserve memory or to store ULP instructions and / or data. This memory can be accessed by CPU via PeriBUS2 (starting address is 0x50000000), and should be force-power-on.
4. RTC fast memory supports 8 KB SRAM, which can be used to reserve memory. This memory can be accessed by CPU via IRAM0 / DRAM0, and should be forced-power-up.
5. When the digital core of the digital system is powered down, all components in the digital system are turned off.

It's worth noting that, ESP32-S2's ROM and SRAM are no longer controlled as independent power domains, thus cannot be force-powered-up or force-powered-down when the digital core is powered down.

6. Power domain Wi-Fi digital circuits includes Wi-Fi MAC and BB (Base Band).

9.4.3 Pre-defined Power Modes

As mentioned earlier, ESP32-S2 has five power modes, which are predefined configurations that power up different combinations of power domains. For details, please refer to Table 63.

Table 63: Predefined Power Modes

Power Modes	Power Domain									
	PMU	RTC Peripherals	RTC Slow Memory	RTC Fast Memory	Digital Core	Wi-Fi Digital Circuits	8 MHz Crystals	40 MHz Crystals	PLL	RF Circuits
Active	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	ON *	ON	ON	ON	OFF
Light-sleep	ON	ON	ON	ON	ON	ON *	OFF	OFF	OFF	OFF
Deep-sleep	ON	ON	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF
Hibernation	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Note:

* Configurable

By default, ESP32-S2 first enters the Active mode after system resets, then enters different low-power modes (including Modem-sleep, Light-sleep, Deep-sleep, and Hibernation) to save power after the CPU stalls for a specific time (For example, when CPU is waiting to be wakened up by an external event). From modes Active to Hibernation, the performance[1] and power consumption[2] decreases and wakeup latency increases. Also, the supported wakeup sources for different power modes are different[3]. Users can choose a power mode based on their requirements of performance, power consumption, wakeup latency, and available wakeup sources.

Note:

1. For details, please refer to Table 63.
2. For details on power consumption, please refer to the Current Consumption Characteristics in *ESP32-S2 Datasheet*.
3. For details on the supported wakeup sources, please refer to Section 9.4.4.

9.4.4 Wakeup Source

The ESP32-S2 supports various wakeup sources, which could wake up the CPU in different sleep modes. The wakeup source is determined by RTC_CNTL_RTC_WAKEUP_ENA as shown in Table 64.

Table 64: Wakeup Source

WAKEUP_ENA	Wakeup Source	Light-sleep	Deep-sleep	Hibernation	Notes*
0x1	EXT0	Y	Y	-	1
0x2	EXT1	Y	Y	Y	2
0x4	GPIO	Y	Y	-	3
0x8	RTC timer	Y	Y	Y	-
0x20	Wi-Fi	Y	-	-	4
0x40	UART0	Y	-	-	5
0x80	UART1	Y	-	-	5
0x100	TOUCH	Y	Y	-	6
0x800	ULP-FSM	Y	Y	-	7
0x1000	XTAL_32K	Y	Y	Y	8
0x2000	ULP-RISCV Trap	Y	Y	-	9
0x8000	USB	Y	-	-	10

Note:

1. EXT0 can only wake up the chip from Light-sleep / Deep-sleep modes. If RTC_CNTL_EXT_WAKEUP0_LV is 1, it's triggered when the pin level is high. Otherwise, it's triggered when the pin level is low. Users can configure RTCIO_EXT_WAKEUP0_SEL to select an RTC pin as a wakeup source.
2. EXT1 is especially designed to wake up the chip from any sleep modes, and can be triggered by a combination of pins. Users should define the combination of wakeup sources by configuring RTC_CNTL_EXT_WAKEUP1_SEL[17:0] according to the bitmap of selected wakeup source. When RTC_CNTL_EXT_WAKEUP1_LV == 1, the chip is waken up if any pin in the combination is high level. When RTC_CNTL_EXT_WAKEUP1_LV == 0, the chip is only waken up if all pins in the combination is low level.
3. In Deep-sleep mode, only the RTC GPIOs (not regular GPIOs) can work as a wakeup source.
4. To wake up the chip with a Wi-Fi source, the chip switches between the Active, Modem-sleep, and Light-sleep modes. The CPU and RF modules are woken up at predetermined intervals to keep Wi-Fi connections active.
5. A wakeup is triggered when the number of RX pulses received exceeds the setting in the threshold register.
6. A wakeup is triggered when any touch event is detected by the touch sensor.
7. A wakeup is triggered when RTC_CNTL_RTC_SW_CPU_INT is configured by the ULP co-processor.
8. When the 32 kHz crystal is working as RTC slow clock, a wakeup is triggered upon any detection of any crystal stops by the 32 kHz watchdog timer.
9. A wakeup is triggered when the ULP co-processor starts capturing exceptions (e.g., stack overflow).
10. A wakeup is triggered when the USB host sends USB into the RESUMING state.

9.5 RTC Boot

The wakeup time for Deep-sleep and Hibernation modes are much longer, compared to the Light-sleep and Modem-sleep, because the ROMs and RAMs are both powered down in this case, and the CPU needs more time for ROM unpacking and data-copying from the flash (SPI booting). However, it's worth noting that both RTC fast memory and RTC slow memory remain powered up in the Deep-sleep mode. Therefore, users can store codes (so called "deep sleep wake stub" of up to 8 KB) either in RTC fast memory or RTC slow memory to avoid the above-mentioned ROM unpacking and SPI booting, thus speeding up the wakeup process.

Method one: Using RTC slow memory

1. Set RTC_CNTL_PROCPU_STAT_VECTOR_SEL to 0.
2. Send the chip into sleep.
3. After the CPU is powered up, the reset vector starts resetting from 0x50000000 instead of 0x40000400, which does not involve any ROM unpacking and SPI booting. The codes stored in RTC slow memory starts running immediately after the CPU reset. However, note that the content in the RTC slow memory should be initialized before sending the chip into sleep.

Method two: Using RTC fast memory

1. Set RTC_CNTL_PROCPU_STAT_VECTOR_SEL to 1.
2. Calculate CRC for the RTC fast memory, and save the result in RTC_CNTL_STORE6_REG[31:0].
3. Set RTC_CNTL_STORE7_REG[31:0] to the entry address of RTC fast memory.
4. Send the chip into sleep.
5. ROM unpacking and some of the initialization starts after the CPU is powered up. After that, calculate the CRC for the RTC fast memory again. If the result matches with register RTC_CNTL_STORE6_REG[31:0], the CPU jumps to the entry address.

The boot flow is shown in Figure 9-11.

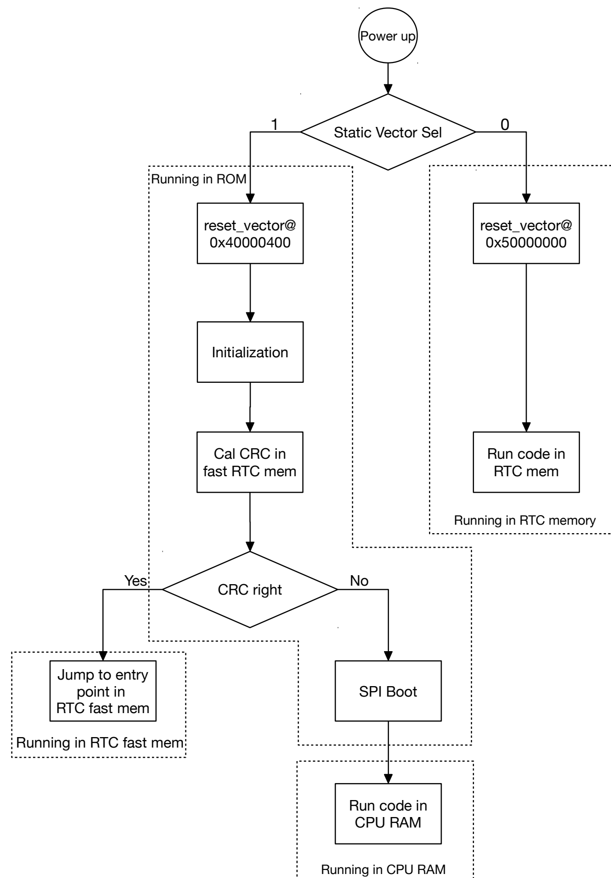


Figure 9-11. ESP32-S2 Boot Flow

When working under low-power modes, ESP32-S2's 40 MHz crystal oscillator and PLL are usually powered down to reduce power consumption. However, the low-power clock remains on so the chip can operate properly under low-power modes.

9.6 Base Address

Users can access the low-power management module of ESP32-S2 with two base addresses, which can be seen in Table 65. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 65: Low-power Management Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3f408000
PeriBUS2	0x60008000

9.7 Register Summary

The addresses in the following table are relative to the Low Power Management base addresses provided in Section 9.6.

Name	Description	Address	Access
RTC Option Registers			
RTC_CNTL_OPTIONS0_REG	Sets the power options of crystal and PLL clocks, and initiates reset by software	0x0000	varies
RTC_CNTL_OPTION1_REG	RTC option register	0x0128	R/W
RTC Timer Registers			
RTC_CNTL_SLP_TIMER0_REG	RTC timer threshold register 0	0x0004	R/W
RTC_CNTL_SLP_TIMER1_REG	RTC timer threshold register 1	0x0008	varies
RTC_CNTL_TIME_UPDATE_REG	RTC timer update control register	0x000C	varies
RTC_CNTL_TIME_LOW0_REG	Stores the lower 32 bits of RTC timer 0	0x0010	RO
RTC_CNTL_TIME_HIGH0_REG	Stores the higher 16 bits of RTC timer 0	0x0014	RO
RTC_CNTL_STATE0_REG	Configures the sleep / reject / wakeup state	0x0018	varies
RTC_CNTL_TIMER1_REG	Configures CPU stall options	0x001C	R/W
RTC_CNTL_TIMER2_REG	Configures RTC slow clock and touch controller	0x0020	R/W
RTC_CNTL_TIMER5_REG	Configures the minimal sleep cycles	0x002C	R/W
RTC_CNTL_TIME_LOW1_REG	Stores the lower 32 bits of RTC timer 1	0x00E8	RO
RTC_CNTL_TIME_HIGH1_REG	Stores the higher 16 bits of RTC timer 1	0x00EC	RO
Internal Power Control Register			
RTC_CNTL_ANA_CONF_REG	Configures the power options for I2C and PLLA	0x0034	R/W
RTC_CNTL_REG	Low-power voltage and digital voltage regulators configuration register	0x0084	R/W
RTC_CNTL_PWC_REG	RTC power configuration register	0x0088	R/W
RTC_CNTL_DIG_PWC_REG	Digital system power configuraiton register	0x008C	R/W
RTC_CNTL_DIG_ISO_REG	Digital system isolation configuration register	0x0090	varies
RTC_CNTL_LOW_POWER_ST_REG	RTC main state machine state register	0x00CC	RO
Reset Control Register			
RTC_CNTL_RESET_STATE_REG	Indicates the CPU reset source	0x0038	varies
Sleep and Wake-up Control Register			
RTC_CNTL_WAKEUP_STATE_REG	Wakeup bitmap enabling register	0x003C	R/W
RTC_CNTL_EXT_WAKEUP_CONF_REG	GPIO wakeup configuration register	0x0064	R/W
RTC_CNTL_SLP_REJECT_CONF_REG	Configures sleep / reject options	0x0068	R/W
RTC_CNTL_EXT_WAKEUP1_REG	EXT1 wakeup configuration register	0x00DC	varies
RTC_CNTL_EXT_WAKEUP1_STATUS_REG	EXT1 wakeup source register	0x00E0	RO
RTC_CNTL_SLP_REJECT_CAUSE_REG	Stores the reject-to-sleep cause	0x0124	RO
RTC_CNTL_SLP_WAKEUP_CAUSE_REG	Stores the sleep-to-wakeup cause	0x012C	RO
Interrupt Registers			
RTC_CNTL_INT_ENA_RTC_REG	RTC interrupt enabling register	0x0040	R/W
RTC_CNTL_INT_RAW_RTC_REG	RTC interrupt raw register	0x0044	RO
RTC_CNTL_INT_ST_RTC_REG	RTC interrupt state register	0x0048	RO
RTC_CNTL_INT_CLR_RTC_REG	RTC interrupt clear register	0x004C	WO
Reservation Registers			
RTC_CNTL_STORE0_REG	Reservation register 0	0x0050	R/W

Name	Description	Address	Access
RTC_CNTL_STORE1_REG	Reservation register 1	0x0054	R/W
RTC_CNTL_STORE2_REG	Reservation register 2	0x0058	R/W
RTC_CNTL_STORE3_REG	Reservation register 3	0x005C	R/W
RTC_CNTL_STORE4_REG	Reservation register 4	0x00BC	R/W
RTC_CNTL_STORE5_REG	Reservation register 5	0x00C0	R/W
RTC_CNTL_STORE6_REG	Reservation register 6	0x00C4	R/W
RTC_CNTL_STORE7_REG	Reservation register 7	0x00C8	R/W
Clock Control Registers			
RTC_CNTL_EXT_XTL_CONF_REG	32 kHz crystal oscillator configuration register	0x0060	varies
RTC_CNTL_CLK_CONF_REG	RTC clock configuration register	0x0074	R/W
RTC_CNTL_SLOW_CLK_CONF_REG	RTC slow clock configuration register	0x0078	R/W
RTC_CNTL_XTAL32K_CLK_FACTOR_REG	Configures the divider for the backup clock of 32 kHz crystal oscillator	0x00F0	R/W
RTC_CNTL_XTAL32K_CONF_REG	32 kHz crystal oscillator configuration register	0x00F4	R/W
RTC Watchdog Control Register			
RTC_CNTL_WDTCONFIG0_REG	RTC watchdog configuration register	0x0094	R/W
RTC_CNTL_WDTCONFIG1_REG	Configures the hold time of RTC watchdog at level 1	0x0098	R/W
RTC_CNTL_WDTCONFIG2_REG	Configures the hold time of RTC watchdog at level 2	0x009C	R/W
RTC_CNTL_WDTCONFIG3_REG	Configures the hold time of RTC watchdog at level 3	0x00A0	R/W
RTC_CNTL_WDTCONFIG4_REG	Configures the hold time of RTC watchdog at level 4	0x00A4	R/W
RTC_CNTL_WDTFEED_REG	RTC watchdog SW feed configuration register	0x00A8	WO
RTC_CNTL_WDTWPROTECT_REG	RTC watchdog write protection configuration register	0x00AC	R/W
RTC_CNTL_SWD_CONF_REG	Super watchdog configuration register	0x00B0	varies
RTC_CNTL_SWD_WPROTECT_REG	Super watchdog write protection configuration register	0x00B4	R/W
Other Registers			
RTC_CNTL_SW_CPU_STALL_REG	CPU stall configuration register	0x00B8	R/W
RTC_CNTL_PAD_HOLD_REG	Configures the hold options for RTC GPIOs	0x00D4	R/W
RTC_CNTL_DIG_PAD_HOLD_REG	Configures the hold options for digital GPIOs	0x00D8	R/W
RTC_CNTL_BROWN_OUT_REG	Brownout configuration register	0x00E4	varies

9.8 Registers

The addresses in this section are relative to the Low Power Management base addresses provided in Section 9.6.

Register 9.1: RTC_CNTL_OPTIONS0_REG (0x0000)

31	30	29	28 ... 14	13	12	11	10	9	8	7	6	5	4	3 ... 2	1 ... 0	
RTC_CNTL_SW_SYS_RST	RTC_CNTL_DG_WRAP_FORCE_NORST	RTC_CNTL_DG_WRAP_FORCE_RST	(reserved)	RTC_CNTL_XTL_FORCE_PU	RTC_CNTL_XTL_FORCE_PD	RTC_CNTL_BBPLL_FORCE_PU	RTC_CNTL_BBPLL_FORCE_PD	RTC_CNTL_BBPLL_I2C_FORCE_PU	RTC_CNTL_BBPLL_I2C_FORCE_PD	RTC_CNTL_BB_I2C_FORCE_PU	RTC_CNTL_BB_I2C_FORCE_PD	RTC_CNTL_SW_PROCPU_RST	(reserved)	RTC_CNTL_SW_STALL_PROCPU_C0	(reserved)	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	0	0	0	0	0	0	0	0	0	0 0	Reset

RTC_CNTL_SW_STALL_PROCPU_C0 When RTC_CNTL_SW_STALL_PROCPU_C1 is configured to 0x21, setting this bit to 0x2 stalls the CPU by SW.

RTC_CNTL_SW_PROCPU_RST Set this bit to reset the CPU by SW. (WO)

RTC_CNTL_BB_I2C_FORCE_PD Set this bit to FPD BB_I2C. (R/W)

RTC_CNTL_BB_I2C_FORCE_PU Set this bit to FPU BB_I2C. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PD Set this bit to FPD BB_PLL _I2C. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PU Set this bit to FPU BB_PLL _I2C. (R/W)

RTC_CNTL_BBPLL_FORCE_PD Set this bit to FPD BB_PLL. (R/W)

RTC_CNTL_BBPLL_FORCE_PU Set this bit to FPU BB_PLL. (R/W)

RTC_CNTL_XTL_FORCE_PD Set this bit to FPD the crystal oscillator. (R/W)

RTC_CNTL_XTL_FORCE_PU Set this bit to FPU the crystal oscillator. (R/W)

RTC_CNTL_DG_WRAP_FORCE_RST Set this bit to force reset the digital system in deep-sleep. (R/W)

RTC_CNTL_DG_WRAP_FORCE_NORST Set this bit to disable force reset to digital system in deep-sleep. (R/W)

RTC_CNTL_SW_SYS_RST Set this bit to reset the system via SW. (WO)

Register 9.2: RTC_CNTL_OPTION1_REG (0x0128)

(reserved) 31–1	RTC_CNTL_FORCE_DOWNLOAD_BOOT 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RTC_CNTL_FORCE_DOWNLOAD_BOOT Set this bit to force the chip to boot from the download mode. (R/W)

Register 9.3: RTC_CNTL_SLP_TIMER0_REG (0x0004)

RTC_CNTL_SLP_VAL_LO 31–0	
0x000000	Reset

RTC_CNTL_SLP_VAL_LO Sets the lower 32 bits of the trigger threshold for the RTC timer. (R/W)

Register 9.4: RTC_CNTL_SLP_TIMER1_REG (0x0008)

(reserved) 31–17	RTC_CNTL_RTC_MAIN_TIMER_ALARM_EN 16	RTC_CNTL_SLP_VAL_HI 15–0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0x00	Reset

RTC_CNTL_SLP_VAL_HI Sets the higher 16 bits of the trigger threshold for the RTC timer. (R/W)

RTC_CNTL_RTC_MAIN_TIMER_ALARM_EN Sets this bit to enable the timer alarm. (WO)

Register 9.5: RTC_CNTL_TIME_UPDATE_REG (0x000C)

RTC_CNTL_RTC_TIME_UPDATE	(reserved)	RTC_CNTL_TIMER_SYS_RST	RTC_CNTL_TIMER_XTL_OFF	RTC_CNTL_TIMER_SYS_STALL	(reserved)	
31	30	29	28	27	26 ... 0	
0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_TIMER_SYS_STALL Selects the triggering condition for the RTC timer. (R/W)

RTC_CNTL_TIMER_XTL_OFF Selects the triggering condition for the RTC timer. (R/W)

RTC_CNTL_TIMER_SYS_RST Selects the triggering condition for the RTC timer. (R/W)

RTC_CNTL_RTC_TIME_UPDATE Selects the triggering condition for the RTC timer. (WO)

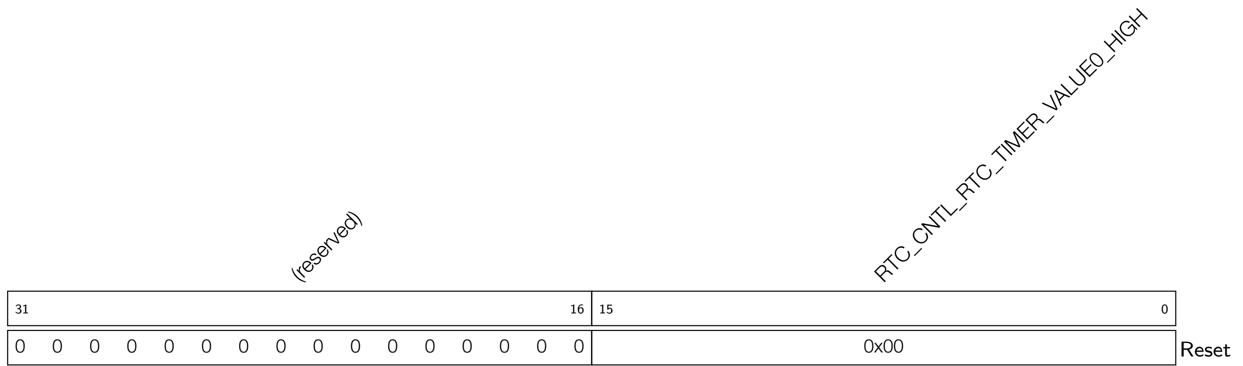
Note:
For details, please refer to Table 60.

Register 9.6: RTC_CNTL_TIME_LOW0_REG (0x0010)

RTC_CNTL_RTC_TIMER_VALUE0_LOW	
31 ... 0	
0x000000	Reset

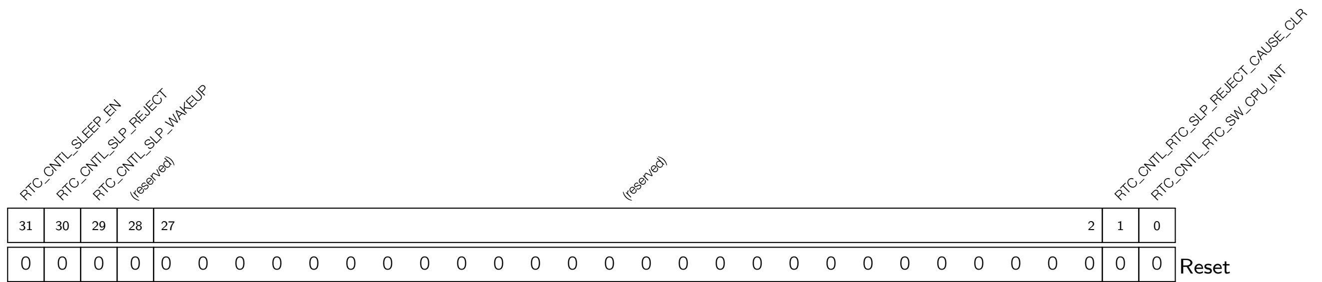
RTC_CNTL_RTC_TIMER_VALUE0_LOW Stores the lower 32 bits of RTC timer 0. (RO)

Register 9.7: RTC_CNTL_TIME_HIGH0_REG (0x0014)



RTC_CNTL_RTC_TIMER_VALUE0_HIGH Stores the higher 16 bits of RTC timer 0. (RO)

Register 9.8: RTC_CNTL_STATE0_REG (0x0018)



RTC_CNTL_RTC_SW_CPU_INT Sends a SW RTC interrupt to CPU. (WO)

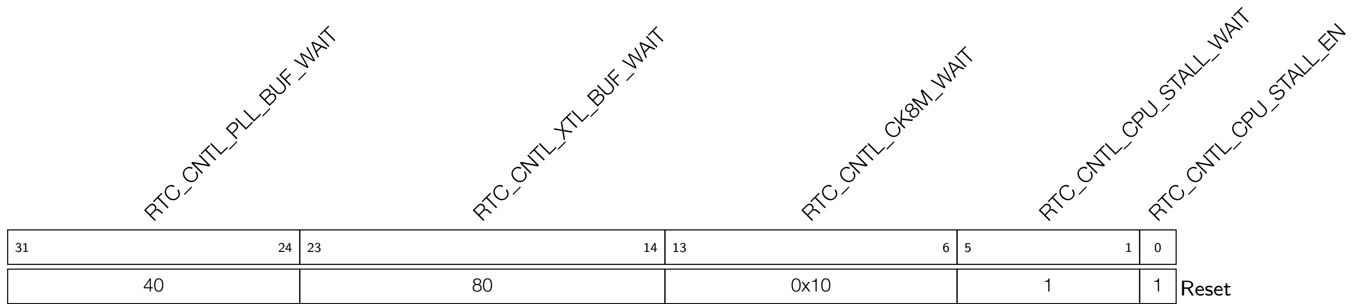
RTC_CNTL_RTC_SLP_REJECT_CAUSE_CLR Clears the RTC reject-to-sleep cause. (WO)

RTC_CNTL_SLP_WAKEUP Sleep wakeup bit. (R/W)

RTC_CNTL_SLP_REJECT Sleep reject bit. (R/W)

RTC_CNTL_SLEEP_EN Sends the chip to sleep. (R/W)

Register 9.9: RTC_CNTL_TIMER1_REG (0x001C)



RTC_CNTL_CPU_STALL_EN Enables the CPU stalling. (R/W)

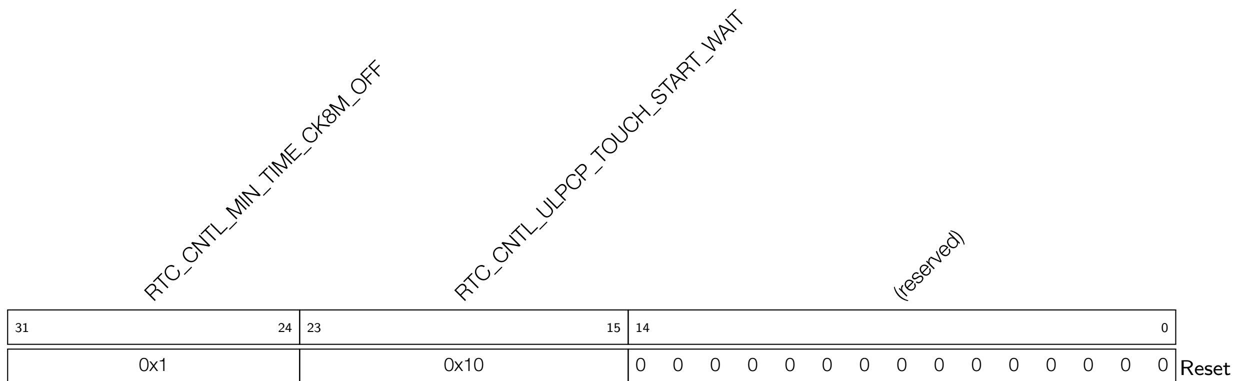
RTC_CNTL_CPU_STALL_WAIT Sets the CPU stall waiting cycles (using the RTC fast clock). (R/W)

RTC_CNTL_CK8M_WAIT Sets the 8 MHz clock waiting cycles (using the RTC slow clock). (R/W)

RTC_CNTL_XTL_BUF_WAIT Sets the XTAL waiting cycles (using the RTC slow clock). (R/W)

RTC_CNTL_PLL_BUF_WAIT Sets the PLL waiting cycles (using the RTC slow clock). (R/W)

Register 9.10: RTC_CNTL_TIMER2_REG (0x0020)



RTC_CNTL_ULPCP_TOUCH_START_WAIT Sets the waiting cycles (using the RTC slow clock) before the ULP co-processor or touch controller starts to work. (R/W)

RTC_CNTL_MIN_TIME_CK8M_OFF Sets the minimal cycles for 8 MHz clock (using the RTC slow clock) when powered down. (R/W)

Register 9.11: RTC_CNTL_TIMER5_REG (0x002C)

31 ... 16	15 ... 8	7 ... 0	
(reserved)	RTC_CNTL_MIN_SLP_VAL	(reserved)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x80	0 0 0 0 0 0 0 0	Reset

RTC_CNTL_MIN_SLP_VAL Sets the minimal sleep cycles (using the RTC slow clock). (R/W)

Register 9.12: RTC_CNTL_TIME_LOW1_REG (0x00E8)

31 ... 0	
RTC_CNTL_RTC_TIMER_VALUE1_LOW	
0x000000	Reset

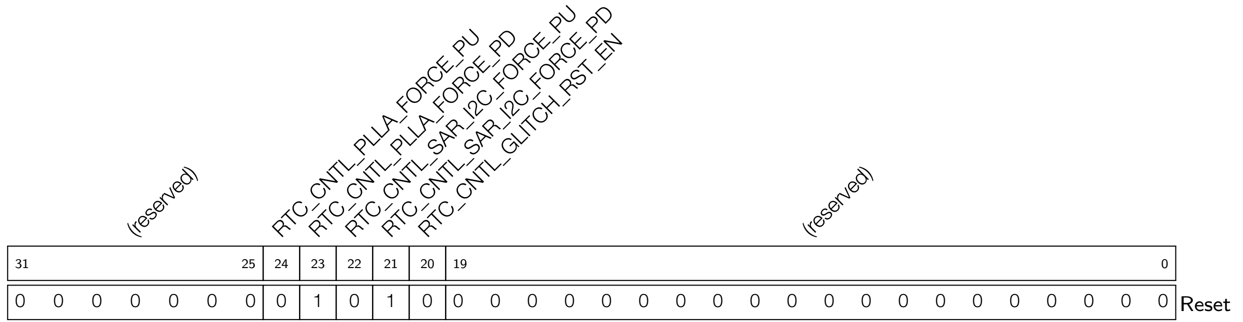
RTC_CNTL_RTC_TIMER_VALUE1_LOW Stores the lower 32 bits of RTC timer 1. (RO)

Register 9.13: RTC_CNTL_TIME_HIGH1_REG (0x00EC)

31 ... 16	15 ... 0	
(reserved)	RTC_CNTL_RTC_TIMER_VALUE1_HIGH	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

RTC_CNTL_RTC_TIMER_VALUE1_HIGH Stores the higher 16 bits of RTC timer. (RO)

Register 9.14: RTC_CNTL_ANA_CONF_REG (0x0034)



RTC_CNTL_GLITCH_RST_EN Set this bit to enable a reset when the system detects a glitch. (R/W)

RTC_CNTL_SAR_I2C_FORCE_PD Set this bit to FPD the SAR_I2C. (R/W)

RTC_CNTL_SAR_I2C_FORCE_PU Set this bit to FPU the SAR_I2C. (R/W)

RTC_CNTL_PLLA_FORCE_PD Set this bit to FPD the PLLA. (R/W)

RTC_CNTL_PLLA_FORCE_PU Sets this bit to FPU the PLLA. (R/W)

Register 9.15: RTC_CNTL_REG (0x0084)

RTC_CNTL_RTC_REGULATOR_FORCE_PU	RTC_CNTL_RTC_REGULATOR_FORCE_PD	(reserved)	RTC_CNTL_RTC_DBIAS_WAK	RTC_CNTL_RTC_DBIAS_SLP	RTC_CNTL_SCK_DCAP	RTC_CNTL_DIG_DBIAS_WAK	RTC_CNTL_DIG_DBIAS_SLP	(reserved)	
31	30	29 28	27 25	24 22	21 14	13 11	10 8	7 0	
1	0	0 0	4	4	0	4	4	0 0 0 0 0 0 0 0	Reset

RTC_CNTL_DIG_DBIAS_SLP Configures the regulation factor for the digital system voltage regulator when the CPU is in sleep state. (R/W)

RTC_CNTL_DIG_DBIAS_WAK Configures the regulation factor for the digital system voltage regulator when the CPU is in active status. (R/W)

RTC_CNTL_SCK_DCAP Configures the frequency of the RTC clocks. (R/W)

RTC_CNTL_RTC_DBIAS_SLP Configures the regulation factor for the low-power voltage regulator when the CPU is in sleep status. (R/W)

RTC_CNTL_RTC_DBIAS_WAK Configures the regulation factor for the low-power voltage regulator when the CPU is in active status. (R/W)

RTC_CNTL_RTC_REGULATOR_FORCE_PD Set this bit to FPD the low-power voltage regulator, which means decreasing its voltage to 0.8 V or lower. (R/W)

RTC_CNTL_RTC_REGULATOR_FORCE_PU Set this bit to FPU the low-power voltage regulator, which means increasing its voltage to higher than 0.8 V. (R/W)

Register 9.16: RTC_CNTL_PWC_REG (0x0088)

Bits	Field	Reset
31:22	(reserved)	0 0 0 0 0 0 0 0 0 0
21	RTC_CNTL_RTC_PAD_FORCE_HOLD	0
20	RTC_CNTL_RTC_PD_EN	0
19	RTC_CNTL_RTC_FORCE_PU	0
18	RTC_CNTL_RTC_FORCE_PD	0
17	RTC_CNTL_RTC_SLOWMEM_PD_EN	0
16	RTC_CNTL_RTC_SLOWMEM_FORCE_PU	1
15	RTC_CNTL_RTC_SLOWMEM_FORCE_PD	0
14	RTC_CNTL_RTC_FASTMEM_PD_EN	0
13	RTC_CNTL_RTC_FASTMEM_FORCE_PU	1
12	RTC_CNTL_RTC_FASTMEM_FORCE_PD	0
11	RTC_CNTL_RTC_SLOWMEM_FORCE_LPU	1
10	RTC_CNTL_RTC_SLOWMEM_FORCE_LPD	0
9	RTC_CNTL_RTC_SLOWMEM_FOLW_CPU	0
8	RTC_CNTL_RTC_FASTMEM_FORCE_LPU	1
7	RTC_CNTL_RTC_FASTMEM_FORCE_LPD	0
6	RTC_CNTL_RTC_FASTMEM_FOLW_CPU	0
5	RTC_CNTL_RTC_FORCE_NOISO	1
4	RTC_CNTL_RTC_FORCE_ISO	0
3	RTC_CNTL_RTC_SLOWMEM_FORCE_ISO	0
2	RTC_CNTL_RTC_SLOWMEM_FORCE_NOISO	1
1	RTC_CNTL_RTC_FASTMEM_FORCE_ISO	0
0	RTC_CNTL_RTC_FASTMEM_FORCE_NOISO	1

RTC_CNTL_RTC_FASTMEM_FORCE_NOISO Set this bit to disable the force isolation of the RTC fast memory. (R/W)

RTC_CNTL_RTC_FASTMEM_FORCE_ISO Set this bit to force isolation of the RTC fast memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_FORCE_NOISO Set this bit to disable the force isolation of the RTC slow memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_FORCE_ISO Set this bit to force isolation of the RTC slow memory. (R/W)

RTC_CNTL_RTC_FORCE_ISO Set this bit to force isolation of the RTC peripherals. (R/W)

RTC_CNTL_RTC_FORCE_NOISO Set this bit to disable the force isolation of the RTC peripherals. (R/W)

RTC_CNTL_RTC_FASTMEM_FOLW_CPU Set this bit to FPD the RTC fast memory when the CPU is powered down. Reset this bit to FPD the RTC fast memory when the RTC main state machine is powered down. (R/W)

RTC_CNTL_RTC_FASTMEM_FORCE_LPD Set this bit to force not retain the RTC fast memory. (R/W)

RTC_CNTL_RTC_FASTMEM_FORCE_LPU Set this bit to force retain the RTC fast memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_FOLW_CPU Set this bit to FPD the RTC slow memory when the CPU is powered down. Reset this bit to FPD the RTC slow memory when the RTC main state machine is powered down. (R/W)

RTC_CNTL_RTC_SLOWMEM_FORCE_LPD Set this bit to force not retain the RTC slow memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_FORCE_LPU Set this bit to force retain the RTC slow memory. (R/W)

RTC_CNTL_RTC_FASTMEM_FORCE_PD Set this bit to FPD the RTC fast memory. (R/W)

RTC_CNTL_RTC_FASTMEM_FORCE_PU Set this bit to FPU the RTC fast memory. (R/W)

RTC_CNTL_RTC_FASTMEM_PD_EN Set this bit to enable PD for the RTC fast memory in sleep. (R/W)

Continued on the next page...

Register 9.16: RTC_CNTL_PWC_REG (0x0088)

Continued from the previous page...

RTC_CNTL_RTC_SLOWMEM_FORCE_PD Set this bit to FPD the RTC slow memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_FORCE_PU Set this bit to FPU the RTC slow memory. (R/W)

RTC_CNTL_RTC_SLOWMEM_PD_EN Set this bit to enable PD for the RTC slow memory in sleep. (R/W)

RTC_CNTL_RTC_FORCE_PD Set this bit to FPD the RTC peripherals. (R/W)

RTC_CNTL_RTC_FORCE_PU Set this bit to FPU the RTC peripherals. (R/W)

RTC_CNTL_RTC_PD_EN Set this bit to enable PD for the RTC peripherals in sleep. (R/W)

RTC_CNTL_RTC_PAD_FORCE_HOLD Set this bit the force hold the RTC GPIOs. (R/W)

Register 9.17: RTC_CNTL_DIG_PWC_REG (0x008C)

Bits	31	30	29–24	23	22	21	20	19	18	17	16–5	4	3	2–0
Field	RTC_CNTL_DG_WRAP_PD_EN	RTC_CNTL_WIFI_PD_EN	(reserved)	(reserved)	(reserved)	(reserved)	RTC_CNTL_DG_WRAP_FORCE_PU	RTC_CNTL_DG_WRAP_FORCE_PD	RTC_CNTL_WIFI_FORCE_PU	RTC_CNTL_WIFI_FORCE_PD	(reserved)	RTC_CNTL_LSLP_MEM_FORCE_PU	RTC_CNTL_LSLP_MEM_FORCE_PD	(reserved)
Reset	0	0	0 0 0 0 0 0	0	1	0	1	0	1	0	0 0 0 0 0 0 0 0 0 0 0 0	1	0	0 0 0

RTC_CNTL_LSLP_MEM_FORCE_PD Set this bit to FPD the memories in the digital system in sleep. (R/W)

RTC_CNTL_LSLP_MEM_FORCE_PU Set this bit to FPU the memories in the digital system. (R/W)

RTC_CNTL_WIFI_FORCE_PD Set this bit to FPD the Wi-Fi digital circuit in the digital system. (R/W)

RTC_CNTL_WIFI_FORCE_PU Set this bit to FPU the Wi-Fi digital circuit in the digital system. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PD Set this bit to FPD the digital system. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PU Set this bit to FPU the digital system. (R/W)

RTC_CNTL_WIFI_PD_EN Set this bit to enable PD for the Wi-Fi digital circuit in sleep. (R/W)

RTC_CNTL_DG_WRAP_PD_EN Set this bit to enable PD for the digital system in sleep. (R/W)

Register 9.18: RTC_CNTL_DIG_ISO_REG (0x0090)

31	30	29	28	27 ... 16	15	14	13	12	11	10	9	8 ... 0	
RTC_CNTL_DG_WRAP_FORCE_NOISO	RTC_CNTL_DG_WRAP_FORCE_ISO	RTC_CNTL_WIFI_FORCE_NOISO	RTC_CNTL_WIFI_FORCE_ISO	(reserved)	RTC_CNTL_DG_PAD_FORCE_HOLD	RTC_CNTL_DG_PAD_FORCE_UNHOLD	RTC_CNTL_DG_PAD_FORCE_ISO	RTC_CNTL_DG_PAD_FORCE_NOISO	RTC_CNTL_DG_PAD_AUTOHOLD_EN	RTC_CNTL_CLR_DG_PAD_AUTOHOLD	RTC_CNTL_DG_PAD_AUTOHOLD	(reserved)	
1	0	1	0	0 0 0 0 0 0 0 0 0 0 0 0	0	1	0	1	0	0	0	0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_DG_PAD_AUTOHOLD Indicates the auto-hold status of the digital GPIOs. (RO)

RTC_CNTL_CLR_DG_PAD_AUTOHOLD Set this bit to clear the auto-hold enabler for the digital GPIOs. (WO)

RTC_CNTL_DG_PAD_AUTOHOLD_EN Set this bit to allow the digital GPIOs to enter the auto-hold status. (R/W)

RTC_CNTL_DG_PAD_FORCE_NOISO Set this bit to disable the force isolation of the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_FORCE_ISO Set this bit to force isolation of the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_FORCE_UNHOLD Set this bit the force unhold the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_FORCE_HOLD Set this bit the force hold the digital GPIOs. (R/W)

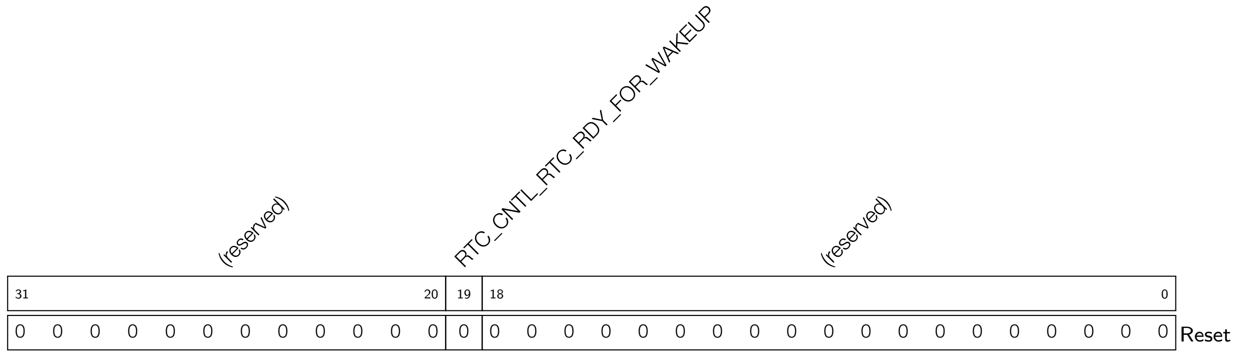
RTC_CNTL_WIFI_FORCE_ISO Set this bit to force isolation of the Wi-Fi digital circuits. (R/W)

RTC_CNTL_WIFI_FORCE_NOISO Set this bit to disable the force isolation of the Wi-Fi digital circuits. (R/W)

RTC_CNTL_DG_WRAP_FORCE_ISO Set this bit to force isolation of the digital system. (R/W)

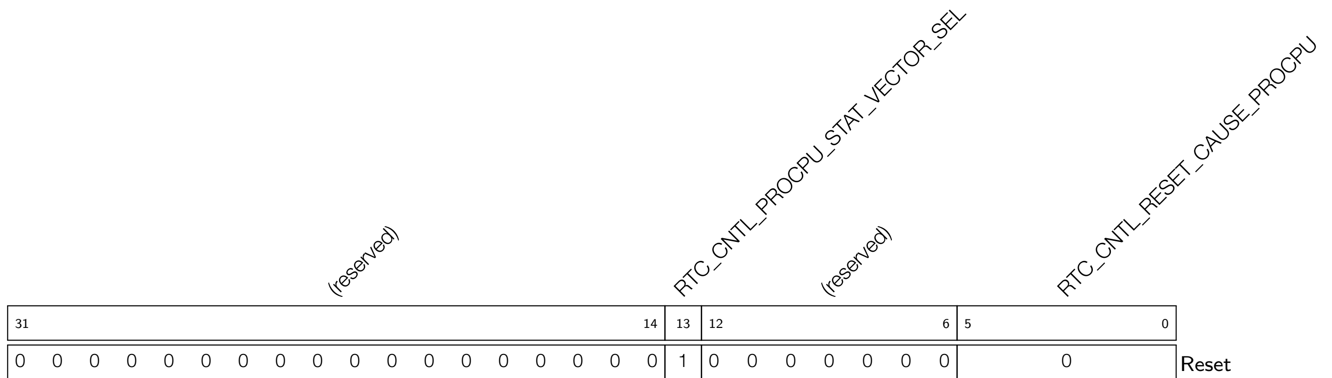
RTC_CNTL_DG_WRAP_FORCE_NOISO Set this bit to disable the force isolation of the digital system. (R/W)

Register 9.19: RTC_CNTL_LOW_POWER_ST_REG (0x00CC)



RTC_CNTL_RTC_RDY_FOR_WAKEUP Indicates the RTC is ready to be triggered by any wakeup source. (RO)

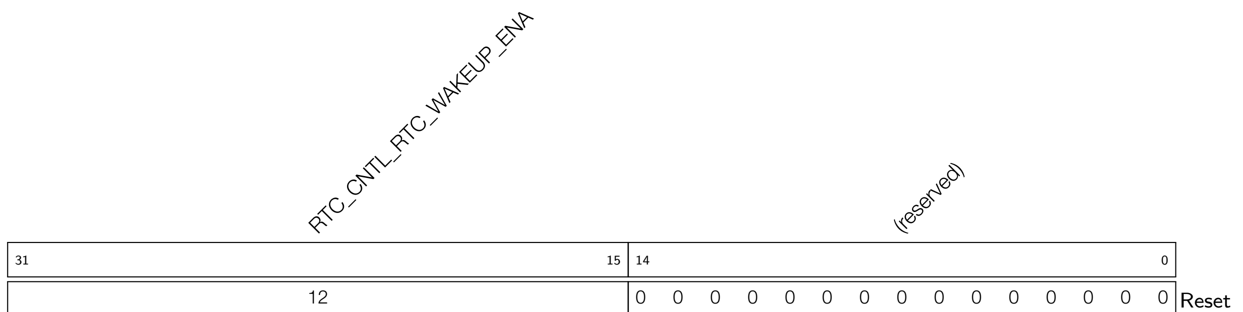
Register 9.20: RTC_CNTL_RESET_STATE_REG (0x0038)



RTC_CNTL_RESET_CAUSE_PROCPU Stores the CPU reset cause. (RO)

RTC_CNTL_PROCPU_STAT_VECTOR_SEL Selects the CPU static vector. (R/W)

Register 9.21: RTC_CNTL_WAKEUP_STATE_REG (0x003C)



RTC_CNTL_RTC_WAKEUP_ENA Selects the wakeup source. For details, please refer to Table 64. (R/W)

Register 9.22: RTC_CNTL_EXT_WAKEUP_CONF_REG (0x0064)

RTC_CNTL_EXT_WAKEUP1_LV	RTC_CNTL_EXT_WAKEUP0_LV	RTC_CNTL_GPIO_WAKEUP_FILTER	(reserved)	
31	30	29	28 ... 0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_GPIO_WAKEUP_FILTER Set this bit to enable the GPIO wakeup event filter. (R/W)

RTC_CNTL_EXT_WAKEUP0_LV 0: EXT0 at low level, 1: EXT0 at high level. (R/W)

RTC_CNTL_EXT_WAKEUP1_LV 0: EXT1 at low level, 1: EXT1 at high level. (R/W)

Register 9.23: RTC_CNTL_SLP_REJECT_CONF_REG (0x0068)

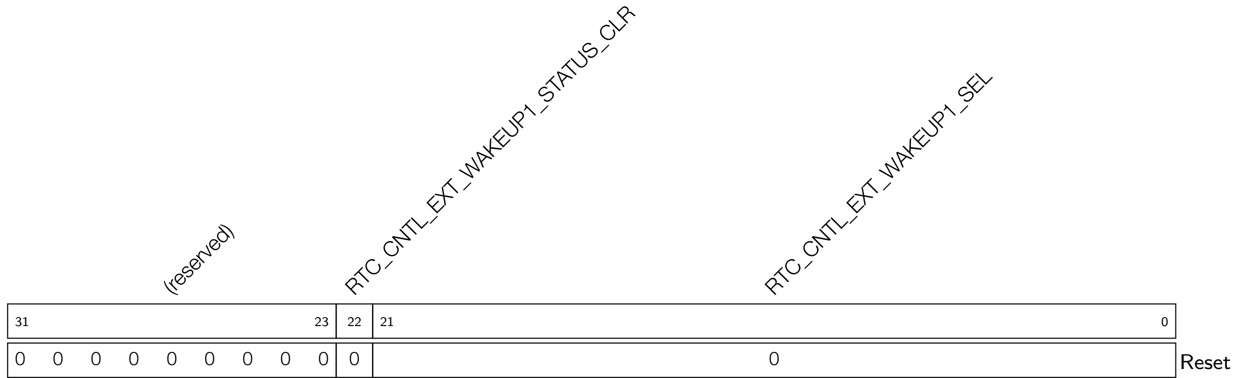
RTC_CNTL_DEEP_SLP_REJECT_EN	RTC_CNTL_LIGHT_SLP_REJECT_EN	RTC_CNTL_RTC_SLEEP_REJECT_ENA	(reserved)	
31	30	29 ... 13	12 ... 0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_RTC_SLEEP_REJECT_ENA Set this bit to enable reject-to-sleep. (R/W)

RTC_CNTL_LIGHT_SLP_REJECT_EN Set this bit to enable reject-to-light-sleep. (R/W)

RTC_CNTL_DEEP_SLP_REJECT_EN Set this bit to enable reject-to-deep-sleep. (R/W)

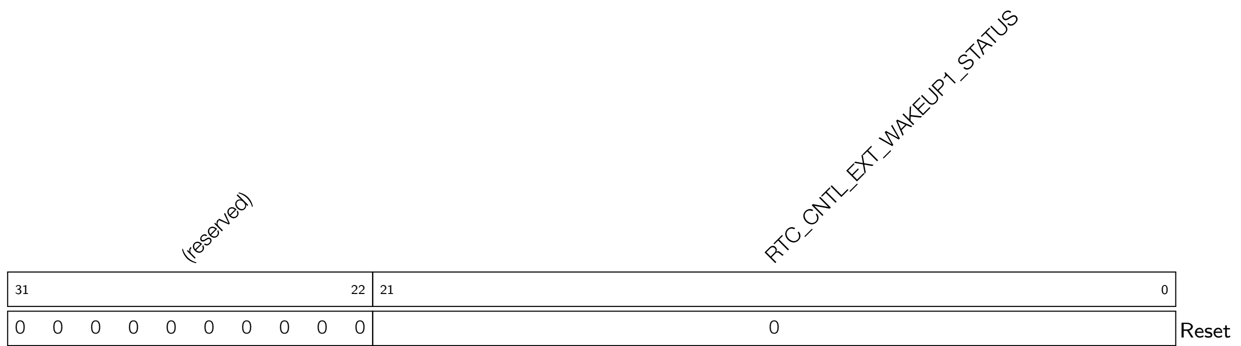
Register 9.24: RTC_CNTL_EXT_WAKEUP1_REG (0x00DC)



RTC_CNTL_EXT_WAKEUP1_SEL Selects a RTC GPIO to be the EXT1 wakeup source. (R/W)

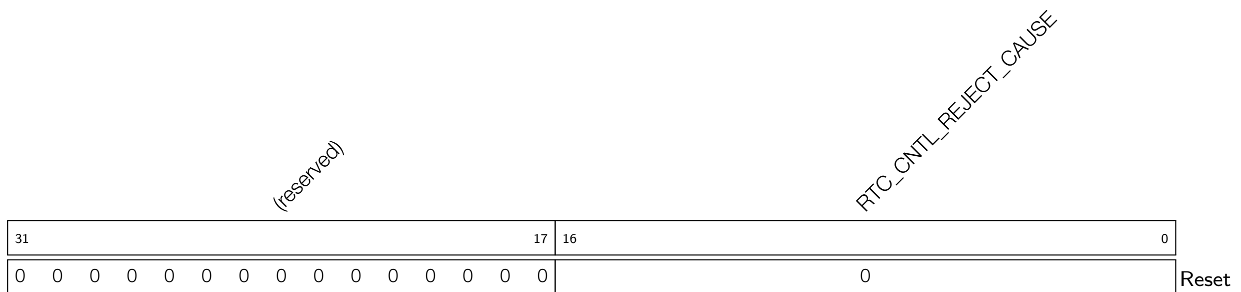
RTC_CNTL_EXT_WAKEUP1_STATUS_CLR Clears the EXT1 wakeup status. (WO)

Register 9.25: RTC_CNTL_EXT_WAKEUP1_STATUS_REG (0x00E0)



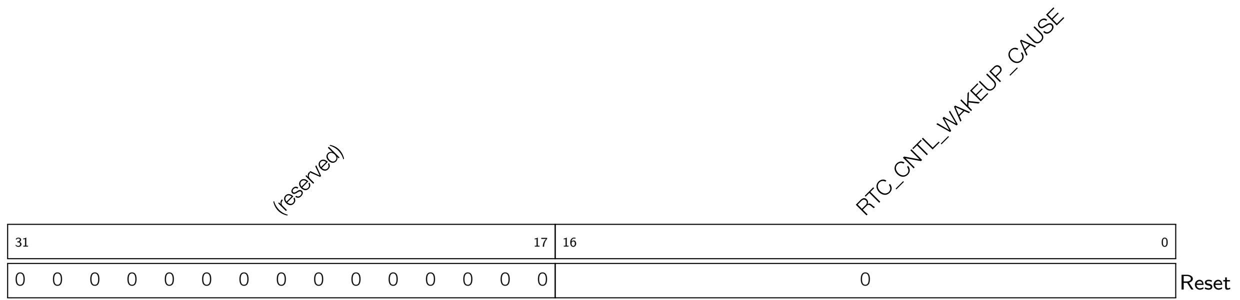
RTC_CNTL_EXT_WAKEUP1_STATUS Indicates the EXT1 wakeup status. (RO)

Register 9.26: RTC_CNTL_SLP_REJECT_CAUSE_REG (0x0124)



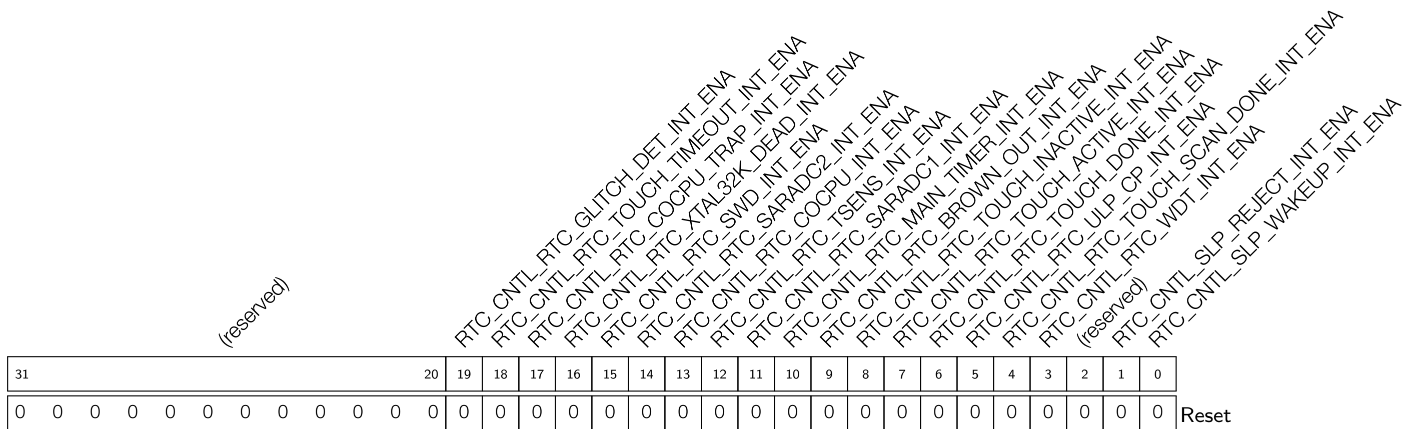
RTC_CNTL_REJECT_CAUSE Stores the reject-to-sleep cause. (RO)

Register 9.27: RTC_CNTL_SLP_WAKEUP_CAUSE_REG (0x012C)



RTC_CNTL_WAKEUP_CAUSE Stores the wakeup cause. (RO)

Register 9.28: RTC_CNTL_INT_ENA_RTC_REG (0x0040)



RTC_CNTL_SLP_WAKEUP_INT_ENA Enables interrupts when the chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA Enables interrupts when the chip rejects to go to sleep. (R/W)

RTC_CNTL_RTC_WDT_INT_ENA Enables the RTC watchdog interrupt. (R/W)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_ENA Enables interrupts upon the completion of a touch scanning. (R/W)

RTC_CNTL_RTC_ULP_CP_INT_ENA Enables the ULP co-processor interrupt. (R/W)

Continued on the next page...

Register 9.28: RTC_CNTL_INT_ENA_RTC_REG (0x0040)

Continued from the previous page...

RTC_CNTL_RTC_TOUCH_DONE_INT_ENA Enables interrupts upon the completion of a single touch. (R/W)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_ENA Enables interrupts when a touch is detected. (R/W)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_ENA Enables interrupts when a touch is released. (R/W)

RTC_CNTL_RTC_BROWN_OUT_INT_ENA Enables the brownout interrupt. (R/W)

RTC_CNTL_RTC_MAIN_TIMER_INT_ENA Enables the RTC main timer interrupt. (R/W)

RTC_CNTL_RTC_SARADC1_INT_ENA Enables the SAR ADC 1 interrupt. (R/W)

RTC_CNTL_RTC_TSENS_INT_ENA Enables the temperature sensor interrupt. (R/W)

RTC_CNTL_RTC_COCPU_INT_ENA Enables the ULP-RISCV interrupt. (R/W)

RTC_CNTL_RTC_SARADC2_INT_ENA Enables the SAR ADC 2 interrupt. (R/W)

RTC_CNTL_RTC_SWD_INT_ENA Enables the super watchdog interrupt. (R/W)

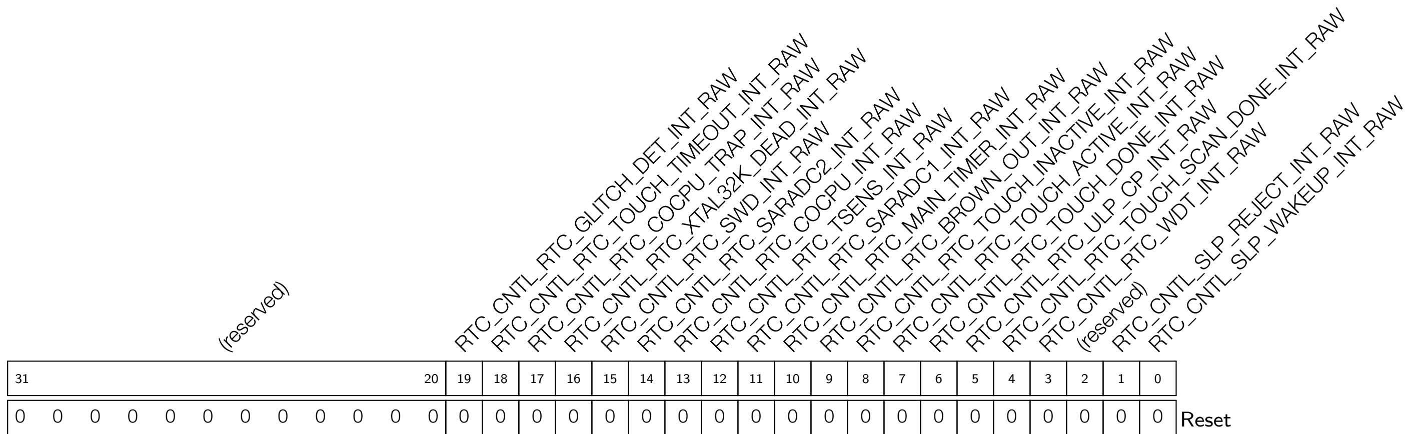
RTC_CNTL_RTC_XTAL32K_DEAD_INT_ENA Enables interrupts when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_RTC_COCPU_TRAP_INT_ENA Enables interrupts when the ULP-RISCV is trapped. (R/W)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_ENA Enables interrupts when touch sensor times out. (R/W)

RTC_CNTL_RTC_GLITCH_DET_INT_ENA Enables interrupts when a glitch is detected. (R/W)

Register 9.29: RTC_CNTL_INT_RAW_RTC_REG (0x0044)



RTC_CNTL_SLP_WAKEUP_INT_RAW Stores the raw interrupt triggered when the chip wakes up from sleep. (RO)

RTC_CNTL_SLP_REJECT_INT_RAW Stores the raw interrupt triggered when the chip rejects to go to sleep. (RO)

RTC_CNTL_RTC_WDT_INT_RAW Stores the raw RTC watchdog interrupt. (RO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_RAW Stores the raw interrupt triggered upon the completion of a touch scanning. (RO)

RTC_CNTL_RTC_ULP_CP_INT_RAW Stores the raw ULP co-processor interrupt. (RO)

RTC_CNTL_RTC_TOUCH_DONE_INT_RAW Stores the raw interrupt triggered upon the completion of a single touch. (RO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_RAW Stores the raw interrupt triggered when a touch is detected. (RO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_RAW Stores the raw interrupt triggered when a touch is released. (RO)

RTC_CNTL_RTC_BROWN_OUT_INT_RAW Stores the raw brownout interrupt. (RO)

RTC_CNTL_RTC_MAIN_TIMER_INT_RAW Stores the raw RTC main timer interrupt. (RO)

RTC_CNTL_RTC_SARADC1_INT_RAW Stores the raw SAR ADC 1 interrupt. (RO)

RTC_CNTL_RTC_TSENS_INT_RAW Stores the raw temperature sensor interrupt. (RO)

RTC_CNTL_RTC_COCPU_INT_RAW Stores the raw ULP-RISCV interrupt. (RO)

RTC_CNTL_RTC_SARADC2_INT_RAW Stores the raw SAR ADC 2 interrupt. (RO)

RTC_CNTL_RTC_SWD_INT_RAW Stores the raw super watchdog interrupt. (RO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_RAW Stores the raw interrupt triggered when the 32 kHz crystal is dead. (RO)

Continued on the next page...

Register 9.29: RTC_CNTL_INT_RAW_RTC_REG (0x0044)

Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_RAW Stores the raw interrupt triggered when the ULP-RISCV is trapped. (RO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_RAW Stores the raw interrupt triggered when touch sensor times out. (RO)

RTC_CNTL_RTC_GLITCH_DET_INT_RAW Stores the raw interrupt triggered when a glitch is detected. (RO)

Register 9.30: RTC_CNTL_INT_ST_RTC_REG (0x0048)

31–20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	RTC_CNTL_RTC_GLITCH_DET_INT_ST	RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_ST	RTC_CNTL_RTC_COCPU_TRAP_INT_ST	RTC_CNTL_RTC_XTAL32K_DEAD_INT_ST	RTC_CNTL_RTC_SWD_INT_ST	RTC_CNTL_RTC_SARADC2_INT_ST	RTC_CNTL_RTC_COCPU_INT_ST	RTC_CNTL_RTC_TSENS_INT_ST	RTC_CNTL_RTC_SARADC1_INT_ST	RTC_CNTL_RTC_MAIN_TIMER_INT_ST	RTC_CNTL_RTC_BROWN_OUT_INT_ST	RTC_CNTL_RTC_TOUCH_INACTIVE_INT_ST	RTC_CNTL_RTC_TOUCH_ACTIVE_INT_ST	RTC_CNTL_RTC_TOUCH_DONE_INT_ST	RTC_CNTL_RTC_ULP_CP_INT_ST	RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_ST	RTC_CNTL_RTC_WDT_INT_ST	(reserved)	RTC_CNTL_SLP_REJECT_INT_ST	RTC_CNTL_SLP_WAKEUP_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_SLP_WAKEUP_INT_ST Stores the status of the interrupt triggered when the chip wakes up from sleep. (RO)

RTC_CNTL_SLP_REJECT_INT_ST Stores the status of the interrupt triggered when the chip rejects to go to sleep. (RO)

RTC_CNTL_RTC_WDT_INT_ST Stores the status of the RTC watchdog interrupt. (RO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_ST Stores the status of the interrupt triggered upon the completion of a touch scanning. (RO)

RTC_CNTL_RTC_ULP_CP_INT_ST Stores the status of the ULP co-processor interrupt. (RO)

RTC_CNTL_RTC_TOUCH_DONE_INT_ST Stores the status of the interrupt triggered upon the completion of a single touch. (RO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_ST Stores the status of the interrupt triggered when a touch is detected. (RO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_ST Stores the status of the interrupt triggered when a touch is released. (RO)

RTC_CNTL_RTC_BROWN_OUT_INT_ST Stores the status of the brownout interrupt. (RO)

RTC_CNTL_RTC_MAIN_TIMER_INT_ST Stores the status of the RTC main timer interrupt. (RO)

RTC_CNTL_RTC_SARADC1_INT_ST Stores the status of the SAR ADC 1 interrupt. (RO)

RTC_CNTL_RTC_TSENS_INT_ST Stores the status of the temperature sensor interrupt. (RO)

RTC_CNTL_RTC_COCPU_INT_ST Stores the status of the ULP-RISCV interrupt. (RO)

RTC_CNTL_RTC_SARADC2_INT_ST Stores the status of the SAR ADC 2 interrupt. (RO)

RTC_CNTL_RTC_SWD_INT_ST Stores the status of the super watchdog interrupt. (RO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_ST Stores the status of the interrupt triggered when the 32 kHz crystal is dead. (RO)

Continued on the next page...

Register 9.30: RTC_CNTL_INT_ST_RTC_REG (0x0048)

Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_ST Stores the status of the interrupt triggered when the ULP-RISCV is trapped. (RO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_ST Stores the status of the interrupt triggered when touch sensor times out. (RO)

RTC_CNTL_RTC_GLITCH_DET_INT_ST Stores the status of the interrupt triggered when a glitch is detected. (RO)

Register 9.31: RTC_CNTL_INT_CLR_RTC_REG (0x004C)

31–20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	RTC_CNTL_RTC_GLITCH_DET_INT_CLR	RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_CLR	RTC_CNTL_RTC_COCPU_TRAP_INT_CLR	RTC_CNTL_RTC_XTAL32K_DEAD_INT_CLR	RTC_CNTL_RTC_SWD_INT_CLR	RTC_CNTL_RTC_SARADC2_INT_CLR	RTC_CNTL_RTC_COCPU_INT_CLR	RTC_CNTL_RTC_TSENS_INT_CLR	RTC_CNTL_RTC_SARADC1_INT_CLR	RTC_CNTL_RTC_MAIN_TIMER_INT_CLR	RTC_CNTL_RTC_BROWN_OUT_INT_CLR	RTC_CNTL_RTC_TOUCH_INACTIVE_INT_CLR	RTC_CNTL_RTC_TOUCH_ACTIVE_INT_CLR	RTC_CNTL_RTC_TOUCH_DONE_INT_CLR	RTC_CNTL_RTC_ULP_CP_INT_CLR	RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_CLR	RTC_CNTL_RTC_WDT_INT_CLR	(reserved)	RTC_CNTL_SLP_REJECT_INT_CLR	RTC_CNTL_SLP_WAKEUP_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_SLP_WAKEUP_INT_CLR Clears the interrupt triggered when the chip wakes up from sleep. (WO)

RTC_CNTL_SLP_REJECT_INT_CLR Clears the interrupt triggered when the chip rejects to go to sleep. (WO)

RTC_CNTL_RTC_WDT_INT_CLR Clears the RTC watchdog interrupt. (WO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_CLR Clears the interrupt triggered upon the completion of a touch scanning. (WO)

RTC_CNTL_RTC_ULP_CP_INT_CLR Clears the ULP co-processor interrupt. (WO)

RTC_CNTL_RTC_TOUCH_DONE_INT_CLR Clears the interrupt triggered upon the completion of a single touch. (WO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_CLR Clears the interrupt triggered when a touch is detected. (WO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_CLR Clears the interrupt triggered when a touch is released. (WO)

RTC_CNTL_RTC_BROWN_OUT_INT_CLR Clears the brownout interrupt. (WO)

RTC_CNTL_RTC_MAIN_TIMER_INT_CLR Clears the RTC main timer interrupt. (WO)

RTC_CNTL_RTC_SARADC1_INT_CLR Clears the SAR ADC 1 interrupt. (WO)

RTC_CNTL_RTC_TSENS_INT_CLR Clears the temperature sensor interrupt. (WO)

RTC_CNTL_RTC_COCPU_INT_CLR Clears the ULP-RISCV interrupt. (WO)

RTC_CNTL_RTC_SARADC2_INT_CLR Clears the SAR ADC 2 interrupt. (WO)

RTC_CNTL_RTC_SWD_INT_CLR Clears the super watchdog interrupt. (WO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_CLR Clears the interrupt triggered when the 32 kHz crystal is dead. (WO)

Continued on the next page...

Register 9.31: RTC_CNTL_INT_CLR_RTC_REG (0x004C)

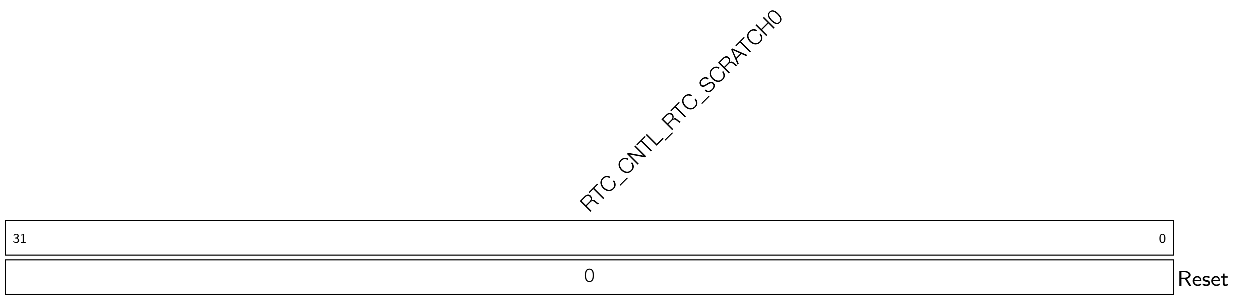
Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_CLR Clears the interrupt triggered when the ULP-RISCV is trapped. (WO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_CLR Clears the interrupt triggered when touch sensor times out. (WO)

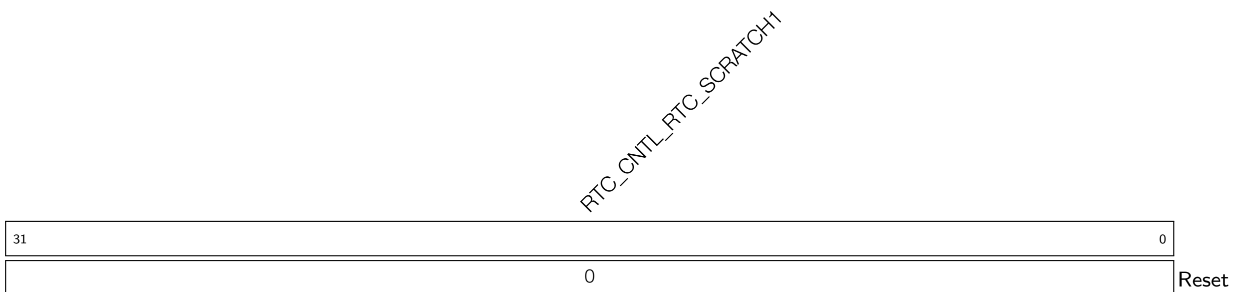
RTC_CNTL_RTC_GLITCH_DET_INT_CLR Clears the interrupt triggered when a glitch is detected. (WO)

Register 9.32: RTC_CNTL_STORE0_REG (0x0050)



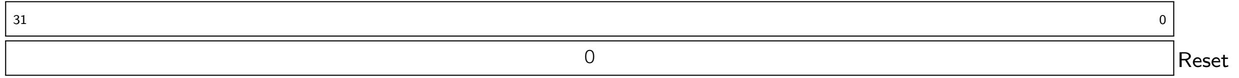
RTC_CNTL_RTC_SCRATCH0 Reservation register 0. (R/W)

Register 9.33: RTC_CNTL_STORE1_REG (0x0054)



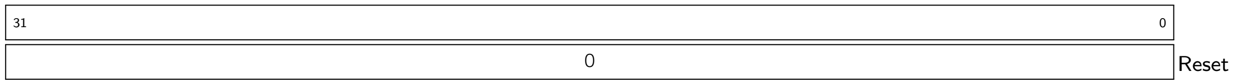
RTC_CNTL_RTC_SCRATCH1 Reservation register 1. (R/W)

Register 9.34: RTC_CNTL_STORE2_REG (0x0058)



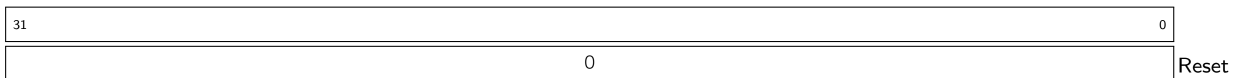
RTC_CNTL_RTC_SCRATCH2 Reservation register 2. (R/W)

Register 9.35: RTC_CNTL_STORE3_REG (0x005C)



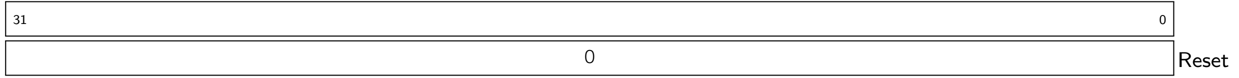
RTC_CNTL_RTC_SCRATCH3 Reservation register 3. (R/W)

Register 9.36: RTC_CNTL_STORE4_REG (0x00BC)



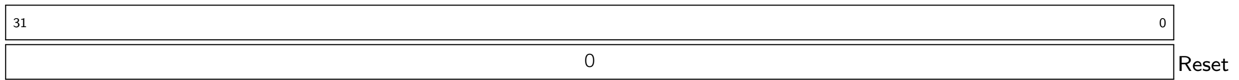
RTC_CNTL_RTC_SCRATCH4 Reservation register 4. (R/W)

Register 9.37: RTC_CNTL_STORE5_REG (0x00C0)



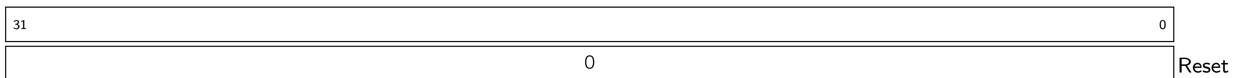
RTC_CNTL_RTC_SCRATCH5 Reservation register 5. (R/W)

Register 9.38: RTC_CNTL_STORE6_REG (0x00C4)



RTC_CNTL_RTC_SCRATCH6 Reservation register 6. (R/W)

Register 9.39: RTC_CNTL_STORE7_REG (0x00C8)



RTC_CNTL_RTC_SCRATCH7 Reservation register 7. (R/W)

Register 9.40: RTC_CNTL_EXT_XTL_CONF_REG (0x0060)

31	30	29–24	23	22–20	19–9	8	7	6	5	4	3	2	1	0	
RTC_CNTL_XTL_EXT_CTR_EN	RTC_CNTL_XTL_EXT_CTR_LV	(reserved)	RTC_CNTL_RTC_XTAL32K_GPIO_SEL	RTC_CNTL_RTC_WDT_STATE	(reserved)	RTC_CNTL_ENCKINIT_XTAL_32K	RTC_CNTL_XTAL32K_XPD_FORCE	RTC_CNTL_XTAL32K_AUTO_RETURN	RTC_CNTL_XTAL32K_AUTO_RESTART	RTC_CNTL_XTAL32K_AUTO_BACKUP	RTC_CNTL_XTAL32K_EXT_CLK_FO	RTC_CNTL_XTAL32K_WDT_RESET	RTC_CNTL_XTAL32K_WDT_CLK_FO	RTC_CNTL_XTAL32K_WDT_EN	
0	0	0 0 0 0 0 0	0	0x0	0 0 0 0 0 0 0 0 0 0 0	0	1	0	0	0	0	0	0	0	Reset

RTC_CNTL_XTAL32K_WDT_EN Set this bit to enable the 32 kHz crystal watchdog. (R/W)

RTC_CNTL_XTAL32K_WDT_CLK_FO Set this bit to FPU the 32 kHz crystal watchdog clock. (R/W)

RTC_CNTL_XTAL32K_WDT_RESET Set this bit to reset the 32 kHz crystal watchdog by SW. (R/W)

RTC_CNTL_XTAL32K_EXT_CLK_FO Set this bit to FPU the external clock of 32 kHz crystal. (R/W)

RTC_CNTL_XTAL32K_AUTO_BACKUP Set this bit to switch to the backup clock when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_XTAL32K_AUTO_RESTART Set this bit to restart the 32 kHz crystal automatically when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_XTAL32K_AUTO_RETURN Set this bit to switch back to 32 kHz crystal when the 32 kHz crystal is restarted. (R/W)

RTC_CNTL_XTAL32K_XPD_FORCE Set this bit to allow the software to FPD the 32 kHz crystal; Reset this bit to allow the FSM to FPD the 32 kHz crystal. (R/W)

RTC_CNTL_ENCKINIT_XTAL_32K Set 1 to apply an internal clock to help the 32 kHz crystal to start. (R/W)

RTC_CNTL_RTC_WDT_STATE Stores the status of the 32 kHz watchdog. (RO)

RTC_CNTL_RTC_XTAL32K_GPIO_SEL Selects the 32 kHz crystal clock. 0: selects the external 32 kHz clock; 1: selects clock from the RTC GPIO X32P_C. (R/W)

RTC_CNTL_XTL_EXT_CTR_LV 0: powers down XTAL at high level; 1: powers down XTAL at low level. (R/W)

RTC_CNTL_XTL_EXT_CTR_EN Enables the GPIO to power down the crystal oscillator. (R/W)

Register 9.41: RTC_CNTL_CLK_CONF_REG (0x0074)

Bits	Field	Reset
31:30	RTC_CNTL_ANA_CLK_RTC_SEL	0
29	RTC_CNTL_FAST_CLK_RTC_SEL	0
28:27	(reserved)	0 0
26	RTC_CNTL_CK8M_FORCE_PU	0
25	RTC_CNTL_CK8M_FORCE_PD	0
24:17	(reserved)	0 0 0 0 0 0 0 0
16	RTC_CNTL_CK8M_FORCE_NOGATING	0
15	RTC_CNTL_XTAL_FORCE_NOGATING	0
14:12	RTC_CNTL_CK8M_DIV_SEL	3
11	(reserved)	0
10	RTC_CNTL_DIG_CLK8M_EN	0
9	RTC_CNTL_DIG_CLK8M_D256_EN	1
8	RTC_CNTL_DIG_XTAL32K_EN	0
7	RTC_CNTL_ENB_CK8M	0
6	RTC_CNTL_ENB_CK8M_DIV	0
5:4	RTC_CNTL_CK8M_DIV	1
3	RTC_CNTL_CK8M_DIV_SEL_VLD	1
2:0	(reserved)	0 0 0

RTC_CNTL_CK8M_DIV_SEL_VLD Synchronizes the reg_ck8m_div_sel. Note that you have to invalidate the bus before modifying the frequency divider, and then validate the new divider clock. (R/W)

RTC_CNTL_CK8M_DIV Set the CK8M_D256_OUT divider. 00: divided by 128, 01: divided by 256, 10: divided by 512, 11: divided by 1024. (R/W)

RTC_CNTL_ENB_CK8M Set this bit to disable CK8M and CK8M_D256_OUT. (R/W)

RTC_CNTL_ENB_CK8M_DIV Selects the CK8M_D256_OUT. 1: CK8M, 0: CK8M divided by 256. (R/W)

RTC_CNTL_DIG_XTAL32K_EN Set this bit to enable CK_XTAL_32K clock for the digital core. (R/W)

RTC_CNTL_DIG_CLK8M_D256_EN Set this bit to enable CK8M_D256_OUT clock for the digital core. (R/W)

RTC_CNTL_DIG_CLK8M_EN Set this bit to enable 8 MHz clock for the digital core. (R/W)

RTC_CNTL_CK8M_DIV_SEL Stores the 8 MHz divider, which is reg_ck8m_div_sel + 1. (R/W)

RTC_CNTL_XTAL_FORCE_NOGATING Set this bit to force no gating to crystal during sleep. (R/W)

RTC_CNTL_CK8M_FORCE_NOGATING Set this bit to disable force gating to 8 MHz crystal during sleep. (R/W)

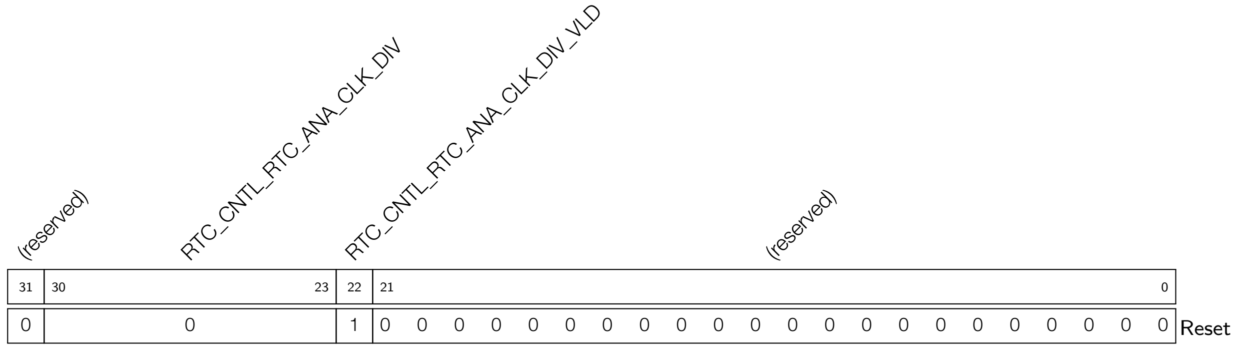
RTC_CNTL_CK8M_FORCE_PD Set this bit to FPD the 8 MHz clock. (R/W)

RTC_CNTL_CK8M_FORCE_PU Set this bit to FPU the 8 MHz clock. (R/W)

RTC_CNTL_FAST_CLK_RTC_SEL Set this bit to select the RTC fast clock. 0: XTAL div 4, 1: CK8M. (R/W)

RTC_CNTL_ANA_CLK_RTC_SEL Set this bit to select the RTC slow clock. 0: 90K rtc_clk, 1: 32k XTAL, 2: 8md256. (R/W)

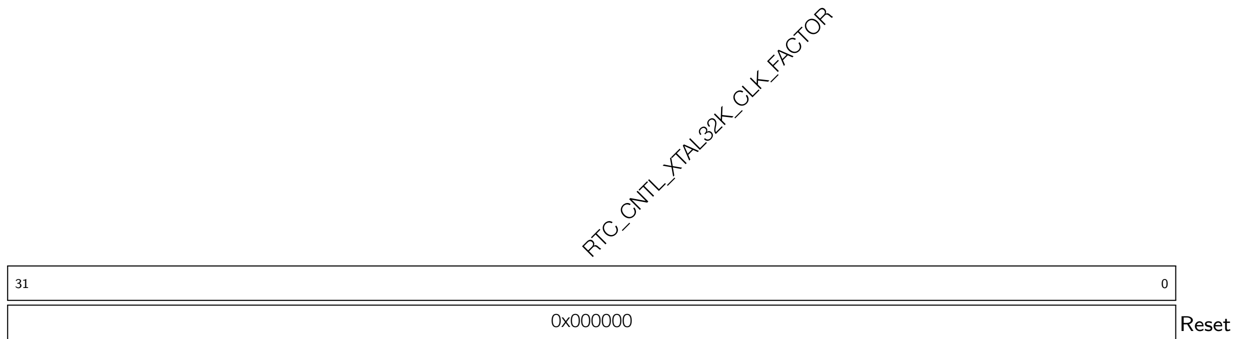
Register 9.42: RTC_CNTL_SLOW_CLK_CONF_REG (0x0078)



RTC_CNTL_RTC_ANA_CLK_DIV_VLD Synchronizes the reg_rtc_ana_clk_div. Note that you have to invalidate the bus before modifying the frequency divider, and then validate the new divider clock. (R/W)

RTC_CNTL_RTC_ANA_CLK_DIV Set the divider for the RTC clock. (R/W)

Register 9.43: RTC_CNTL_XTAL32K_CLK_FACTOR_REG (0x00F0)



RTC_CNTL_XTAL32K_CLK_FACTOR Configures the divider factor for the 32 kHz crystal oscillator. (R/W)

Register 9.44: RTC_CNTL_XTAL32K_CONF_REG (0x00F4)

RTC_CNTL_XTAL32K_STABLE_THRES	RTC_CNTL_XTAL32K_WDT_TIMEOUT	RTC_CNTL_XTAL32K_RESTART_WAIT	RTC_CNTL_XTAL32K_RETURN_WAIT	
31 28	27 20	19 4	3 0	
0x0	0xff	0x00	0x0	Reset

RTC_CNTL_XTAL32K_RETURN_WAIT Defines the waiting cycles before returning to the normal 32 kHz crystal oscillator. (R/W)

RTC_CNTL_XTAL32K_RESTART_WAIT Defines the waiting cycles before restarting the 32 kHz crystal oscillator. (R/W)

RTC_CNTL_XTAL32K_WDT_TIMEOUT Defines the waiting period for clock detection. If no clock is detected after this period, the 32 kHz crystal oscillator can be regarded as dead. (R/W)

RTC_CNTL_XTAL32K_STABLE_THRES Defines the allowed restarting period, within which the 32 kHz crystal oscillator can be regarded as stable. (R/W)

Register 9.45: RTC_CNTL_WDTCONFIG0_REG (0x0094)

Bits	Field	Reset
31	RTC_CNTL_WDT_EN	0
30–28	RTC_CNTL_WDT_STG0	0x0
27–25	RTC_CNTL_WDT_STG1	0x0
24–22	RTC_CNTL_WDT_STG2	0x0
21–19	RTC_CNTL_WDT_STG3	0x0
18–16	RTC_CNTL_WDT_CPU_RESET_LENGTH	0x1
15–13	RTC_CNTL_WDT_SYS_RESET_LENGTH	0x1
12	RTC_CNTL_WDT_FLASHBOOT_MOD_EN	1
11	RTC_CNTL_WDT_PROCPU_RESET_EN	0
10	(reserved)	0
9	RTC_CNTL_WDT_PAUSE_IN_SLP	1
8–0	(reserved)	0 0 0 0 0 0 0 0 0

RTC_CNTL_WDT_PAUSE_IN_SLP Set this bit to pause the watchdog in sleep. (R/W)

RTC_CNTL_WDT_PROCPU_RESET_EN Set this bit to allow the watchdog to be able to reset CPU. (R/W)

RTC_CNTL_WDT_FLASHBOOT_MOD_EN Set this bit to enable watchdog when the chip boots from flash. (R/W)

RTC_CNTL_WDT_SYS_RESET_LENGTH Sets the length of the system reset counter. (R/W)

RTC_CNTL_WDT_CPU_RESET_LENGTH Sets the length of the CPU reset counter. (R/W)

RTC_CNTL_WDT_STG3 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_WDT_STG2 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

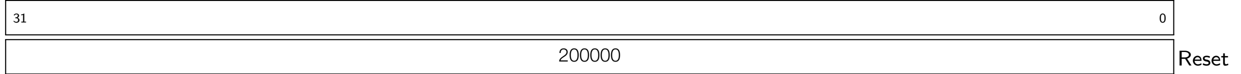
RTC_CNTL_WDT_STG1 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_WDT_STG0 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_WDT_EN Set this bit to enable the RTC watchdog. (R/W)

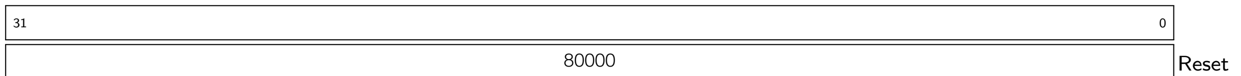
Note:
For details, please refer to Chapter 13 *XTAL32K Watchdog Timer (XTWDT)*.

Register 9.46: RTC_CNTL_WDTCONFIG1_REG (0x0098)



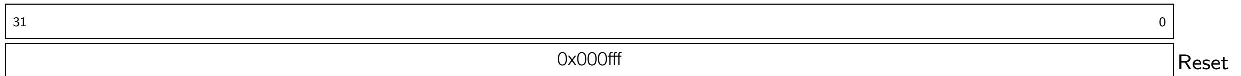
RTC_CNTL_WDT_STG0_HOLD Configures the hold time of RTC watchdog at level 1. (R/W)

Register 9.47: RTC_CNTL_WDTCONFIG2_REG (0x009C)



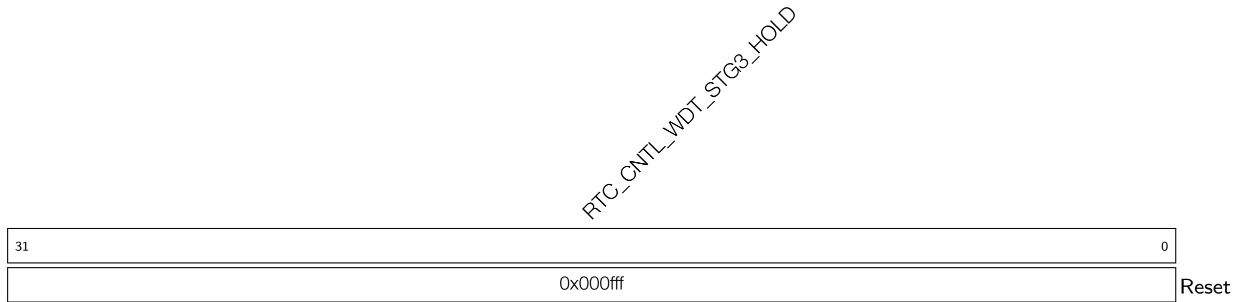
RTC_CNTL_WDT_STG1_HOLD Configures the hold time of RTC watchdog at level 2. (R/W)

Register 9.48: RTC_CNTL_WDTCONFIG3_REG (0x00A0)



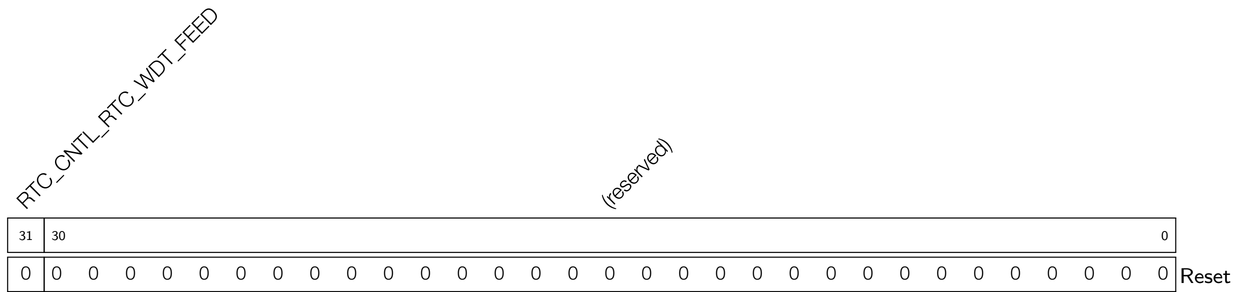
RTC_CNTL_WDT_STG2_HOLD Configures the hold time of RTC watchdog at level 3. (R/W)

Register 9.49: RTC_CNTL_WDTCONFIG4_REG (0x00A4)



RTC_CNTL_WDT_STG3_HOLD Configures the hold time of RTC watchdog at level 4. (R/W)

Register 9.50: RTC_CNTL_WDTFEED_REG (0x00A8)

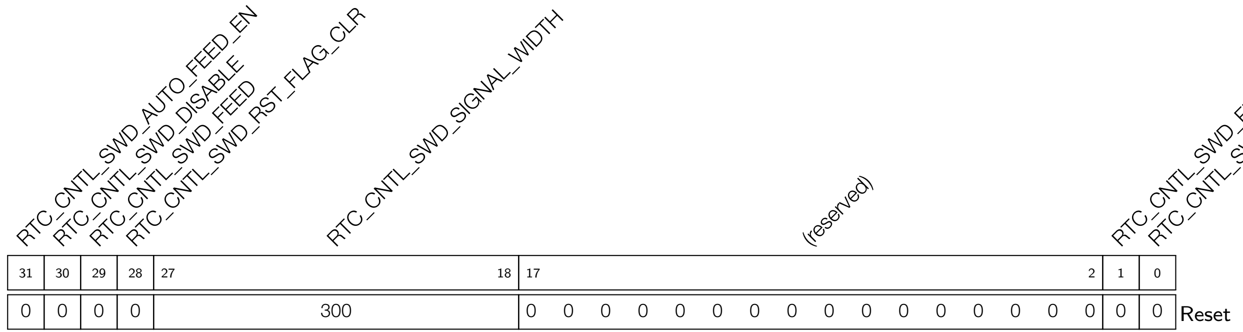


RTC_CNTL_RTC_WDT_FEED Set this bit to feed the RTC watchdog. (WO)

Register 9.51: RTC_CNTL_WDTWPROTECT_REG (0x00AC)

RTC_CNTL_WDT_WKEY Sets the write protection key of the watchdog. (R/W)

Register 9.52: RTC_CNTL_SWD_CONF_REG (0x00B0)



RTC_CNTL_SWD_RESET_FLAG Indicates the super watchdog reset flag. (RO)

RTC_CNTL_SWD_FEED_INT Receiving this interrupt leads to feeding the super watchdog via SW. (RO)

RTC_CNTL_SWD_SIGNAL_WIDTH Adjusts the signal width sent to the super watchdog. (R/W)

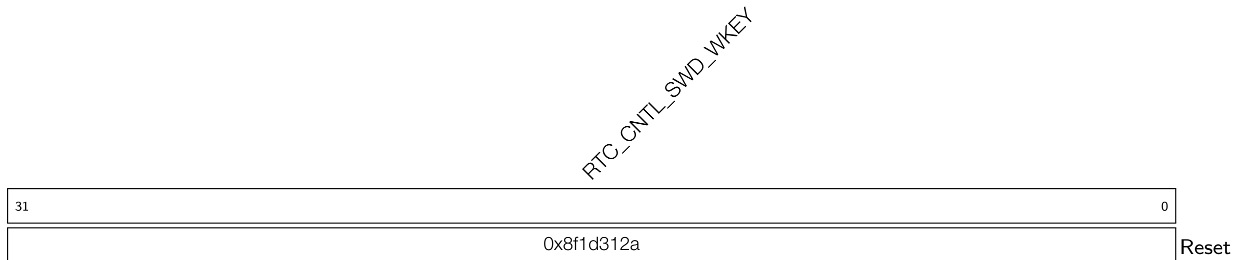
RTC_CNTL_SWD_RST_FLAG_CLR Set to reset the super watchdog reset flag. (WO)

RTC_CNTL_SWD_FEED Set to feed the super watchdog via SW. (WO)

RTC_CNTL_SWD_DISABLE Set this bit to disable super watchdog. (R/W)

RTC_CNTL_SWD_AUTO_FEED_EN Set this bit to enable automatic watchdog feeding upon interrupts. (R/W)

Register 9.53: RTC_CNTL_SWD_WPROTECT_REG (0x00B4)



RTC_CNTL_SWD_WKEY Sets the write protection key of the super watchdog. (R/W)

Register 9.54: RTC_CNTL_SW_CPU_STALL_REG (0x00B8)

RTC_CNTL_SW_STALL_PROCPU_C1	(reserved)	
31 26	25 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_SW_STALL_PROCPU_C1 When RTC_CNTL_SW_STALL_PROCPU_C0 is configured to 0x2, setting this bit to 0x21 stalls the CPU by SW. (R/W)

Register 9.55: RTC_CNTL_PAD_HOLD_REG (0x00D4)

(reserved)	RTC_CNTL_RTC_PAD21_HOLD	RTC_CNTL_RTC_PAD20_HOLD	RTC_CNTL_RTC_PAD19_HOLD	RTC_CNTL_PDAC2_HOLD	RTC_CNTL_PDAC1_HOLD	RTC_CNTL_X32N_HOLD	RTC_CNTL_X32P_HOLD	RTC_CNTL_TOUCH_PAD14_HOLD	RTC_CNTL_TOUCH_PAD13_HOLD	RTC_CNTL_TOUCH_PAD12_HOLD	RTC_CNTL_TOUCH_PAD11_HOLD	RTC_CNTL_TOUCH_PAD10_HOLD	RTC_CNTL_TOUCH_PAD9_HOLD	RTC_CNTL_TOUCH_PAD8_HOLD	RTC_CNTL_TOUCH_PAD7_HOLD	RTC_CNTL_TOUCH_PAD6_HOLD	RTC_CNTL_TOUCH_PAD5_HOLD	RTC_CNTL_TOUCH_PAD4_HOLD	RTC_CNTL_TOUCH_PAD3_HOLD	RTC_CNTL_TOUCH_PAD2_HOLD	RTC_CNTL_TOUCH_PAD1_HOLD	RTC_CNTL_TOUCH_PAD0_HOLD	
31 22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_TOUCH_PAD0_HOLD Sets the touch GPIO 0 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD1_HOLD Sets the touch GPIO 1 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD2_HOLD Sets the touch GPIO 2 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD3_HOLD Sets the touch GPIO 3 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD4_HOLD Sets the touch GPIO 4 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD5_HOLD Sets the touch GPIO 5 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD6_HOLD Sets the touch GPIO 6 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD7_HOLD Sets the touch GPIO 7 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD8_HOLD Sets the touch GPIO 8 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD9_HOLD Sets the touch GPIO 9 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD10_HOLD Sets the touch GPIO 10 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD11_HOLD Sets the touch GPIO 11 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD12_HOLD Sets the touch GPIO 12 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD13_HOLD Sets the touch GPIO 13 to the holding state. (R/W)

RTC_CNTL_TOUCH_PAD14_HOLD Sets the touch GPIO 14 to the holding state. (R/W)

RTC_CNTL_X32P_HOLD Sets the x32p to the holding state. (R/W)

RTC_CNTL_X32N_HOLD Sets the x32n to the holding state. (R/W)

RTC_CNTL_PDAC1_HOLD Sets the pdac1 to the holding state. (R/W)

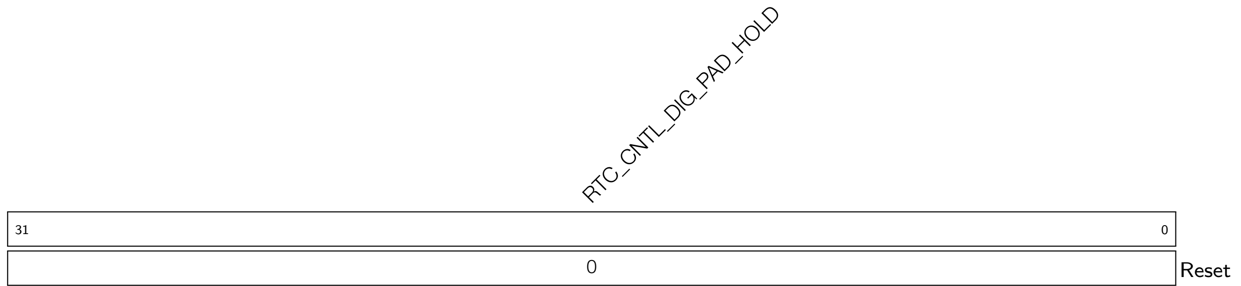
RTC_CNTL_PDAC2_HOLD Sets the pdac2 to the holding state. (R/W)

RTC_CNTL_RTC_PAD19_HOLD Sets the RTG GPIO 19 to the holding state. (R/W)

RTC_CNTL_RTC_PAD20_HOLD Sets the RTG GPIO 20 to the holding state. (R/W)

RTC_CNTL_RTC_PAD21_HOLD Sets the RTG GPIO 21 to the holding state. (R/W)

Register 9.56: RTC_CNTL_DIG_PAD_HOLD_REG (0x00D8)



RTC_CNTL_DIG_PAD_HOLD Set GPIO 21 to GPIO 45 to the holding state. (See bitmap to locate any GPIO). (R/W)

Register 9.57: RTC_CNTL_BROWN_OUT_REG (0x00E4)

RTC_CNTL_RTC_BROWN_OUT_DET	RTC_CNTL_BROWN_OUT_ENA	RTC_CNTL_BROWN_OUT_CNT_CLR	(reserved)	RTC_CNTL_BROWN_OUT_RST_SEL	RTC_CNTL_BROWN_OUT_RST_ENA	RTC_CNTL_BROWN_OUT_RST_WAIT	RTC_CNTL_BROWN_OUT_PD_RF_ENA	RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA	RTC_CNTL_BROWN_OUT_INT_WAIT	(reserved)	RTC_CNTL_BROWN_OUT2_ENA	
31	30	29	28	27	26	25 ... 16	15	14	13 ... 4	3 ... 1	0	
0	0	0	0	0	0	0x3ff	0	0	0x2ff	0 0 0	1	Reset

RTC_CNTL_BROWN_OUT2_ENA Enables the brown_out2 to initiate a chip reset. (R/W)

RTC_CNTL_BROWN_OUT_INT_WAIT Configures the waiting cycles before sending an interrupt. (R/W)

RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA Set this bit to enable PD the flash when a brown-out happens. (R/W)

RTC_CNTL_BROWN_OUT_PD_RF_ENA Set this bit to enable PD the RF circuits when a brown-out happens. (R/W)

RTC_CNTL_BROWN_OUT_RST_WAIT Configures the waiting cycles before the reset after a brown-out. (R/W)

RTC_CNTL_BROWN_OUT_RST_ENA Enables to reset brown-out. (R/W)

RTC_CNTL_BROWN_OUT_RST_SEL Selects the reset type when a brown-out happens. 1: chip reset, 0: system reset. (R/W)

RTC_CNTL_BROWN_OUT_CNT_CLR Clears the brown-out counter. (WO)

RTC_CNTL_BROWN_OUT_ENA Set this bit to enable brown-out detection. (R/W)

RTC_CNTL_RTC_BROWN_OUT_DET Indicates the status of the brown-out signal. (RO)

10. System Timer (SYSTIMER)

10.1 Overview

System timer is a 64-bit timer specially for operating system. It can be used to schedule operating system tasks by generating periodical system ticks or certain time delay interrupts. With the help of RTC timer, system timer can keep updated after Light-sleep or Deep-sleep.

10.2 Main Features

- A 64-bit timer
- Clocked with APB_CLK
- Timer value increment step can be configured for each APB_CLK cycle.
- Support automatic time compensation in case of APB_CLK clock source switching between PLL_CLK and XTAL_CLK, to improve timer accuracy.
- Generate three independent interrupts based on different alarm values or periods (targets).
- Support for 64-bit alarm values and 30-bit periods.
- Load back sleep time recorded by RTC timer after Deep-sleep or Light-sleep by software.
- Keep stalled if CPU is stalled or CPU is in on-chip-debugging mode.

10.3 Clock Source Selection

The System Timer is driven using XTAL_CLK or PLL_CLK clock. Selection between the two clock sources is described in Table CPU_CLK Source in Chapter 6 *Reset and Clock*. For the specific clock frequency used for the System Timer, please refer to Table 49 *APB_CLK Source*. On each clock period the timer will be increased by a step value configured in either SYSTIMER_TIMER_XTAL_STEP or SYSTIMER_TIMER_PLL_STEP, depending on which clock source is used.

10.4 Functional Description

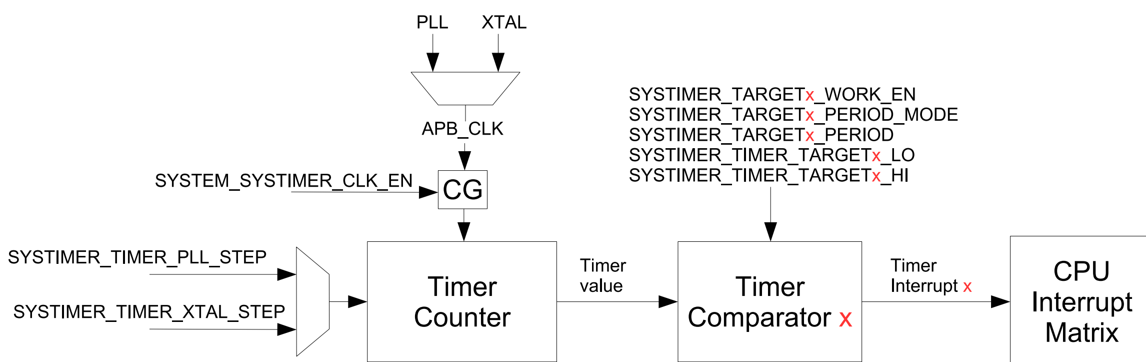


Figure 10-1. System Timer Structure

Figure 10-1 shows the structure of system timer. The system timer can be enabled by setting the bit SYSTEM_SYSTIMER_CLK_EN in register SYSTEM_PERIP_CLK_EN0_REG and be reset via software by setting the bit SYSTEM_SYSTIMER_RST in register SYSTEM_PERIP_RST_EN0_REG. For more information, please refer

to Table 87 *Peripheral Clock Gating and Reset Bits* in Chapter 15 *System Registers (SYSTEM)*.

10.4.1 Read System Timer Value

1. Set SYSTIMER_TIMER_UPDATE to update the timer value into registers.
2. Wait till SYSTIMER_TIMER_VALUE_VALID is set, which means users now can read the timer values from registers.
3. Read the high 32 bits of the timer value from SYSTIMER_TIMER_VALUE_HI, and the low 32 bits from SYSTIMER_TIMER_VALUE_LO.

10.4.2 Configure a Time-Delay Alarm

1. Read the current value of system timer, see Section 10.4.1. This value will be used to calculate the target in Step 3.
2. Clear SYSTIMER_TARGET*x*_PERIOD_MODE to set the timer into a time-delay alarm mode.
3. Write the high 32 bits of the target (alarm value) to SYSTIMER_TIMER_TARGET*x*_HI, and the low 32 bits to SYSTIMER_TIMER_TARGET*x*_LO.
4. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected work mode.
5. Set SYSTIMER_INT*x*_ENA to enable timer interrupt. When the timer counts to the alarm value, an interrupt will be triggered.

10.4.3 Configure Periodic Alarms

1. Set SYSTIMER_TARGET*x*_PERIOD_MODE to configure the timer into periodic alarms mode.
2. Write the target (alarm period) to SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TARGET*x*_WORK_EN to enable periodical alarms mode.
4. Set SYSTIMER_INT*x*_ENA to enable timer interrupt. An interrupt will be triggered when the timer counts to the target value set in Step 2.

10.4.4 Update after Deep-sleep and Light-sleep

1. Configure RTC timer before the chip goes to Deep-sleep or Light-sleep, to record the exact sleep time.
2. Read the sleep time from RTC timer when the chip is woken up from Deep-sleep or Light-sleep.
3. Read current value of the system timer. See Section 10.4.1 for how to read the timer value.
4. Add current value of the system timer to the time that RTC timer records in Deep-sleep mode or in Light-sleep mode.
5. Write the result into SYSTIMER_TIMER_LOAD_HI (high 32 bits), and into SYSTIMER_TIMER_LOAD_LO (low 32 bits).
6. Set SYSTIMER_TIMER_LOAD to load the new value stored in SYSTIMER_TIMER_LOAD_HI and SYSTIMER_TIMER_LOAD_LO into the system timer. By such way, the system timer is updated.

10.5 Base Address

Users can access system timer registers with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and*

Memory.

Table 67: System Timer Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F423000
PeriBUS2	0x60023000

10.6 Register Summary

The addresses in the following table are relative to system timer base addresses provided in Section 10.5.

Name	Description	Address	Access
System Timer Registers			
SYSTIMER_CONF_REG	Configure system timer clock	0x0000	R/W
SYSTIMER_LOAD_REG	Load value to system timer	0x0004	WO
SYSTIMER_LOAD_HI_REG	High 32 bits to be loaded to system timer	0x0008	R/W
SYSTIMER_LOAD_LO_REG	Low 32 bits to be loaded to system timer	0x000C	R/W
SYSTIMER_STEP_REG	System timer accumulation step	0x0010	R/W
SYSTIMER_TARGET0_HI_REG	System timer target 0, high 32 bits	0x0014	R/W
SYSTIMER_TARGET0_LO_REG	System timer target 0, low 32 bits	0x0018	R/W
SYSTIMER_TARGET1_HI_REG	System timer target 1, high 32 bits	0x001C	R/W
SYSTIMER_TARGET1_LO_REG	System timer target 1, low 32 bits	0x0020	R/W
SYSTIMER_TARGET2_HI_REG	System timer target 2, high 32 bits	0x0024	R/W
SYSTIMER_TARGET2_LO_REG	System timer target 2, low 32 bits	0x0028	R/W
SYSTIMER_TARGET0_CONF_REG	Configure work mode for system timer target 0	0x002C	R/W
SYSTIMER_TARGET1_CONF_REG	Configure work mode for system timer target 1	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	Configure work mode for system timer target 2	0x0034	R/W
SYSTIMER_UPDATE_REG	Read out system timer value	0x0038	varies
SYSTIMER_VALUE_HI_REG	System timer value, high 32 bits	0x003C	RO
SYSTIMER_VALUE_LO_REG	System timer value, low 32 bits	0x0040	RO
SYSTIMER_INT_ENA_REG	System timer interrupt enable	0x0044	R/W
SYSTIMER_INT_RAW_REG	System timer interrupt raw	0x0048	RO
SYSTIMER_INT_CLR_REG	System timer interrupt clear	0x004C	WO
Version Register			
SYSTIMER_DATE_REG	Version control register	0x00FC	R/W

10.7 Registers

Register 10.1: SYSTIMER_CONF_REG (0x0000)

SYSTIMER_CLK_EN	(reserved)	SYSTIMER_CLK_FO	
31	30 ... 1	0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SYSTIMER_CLK_FO System timer clock force enable. (R/W)

SYSTIMER_CLK_EN Register clock enable. (R/W)

Register 10.2: SYSTIMER_LOAD_REG (0x0004)

SYSTIMER_TIMER_LOAD	(reserved)	
31	30 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

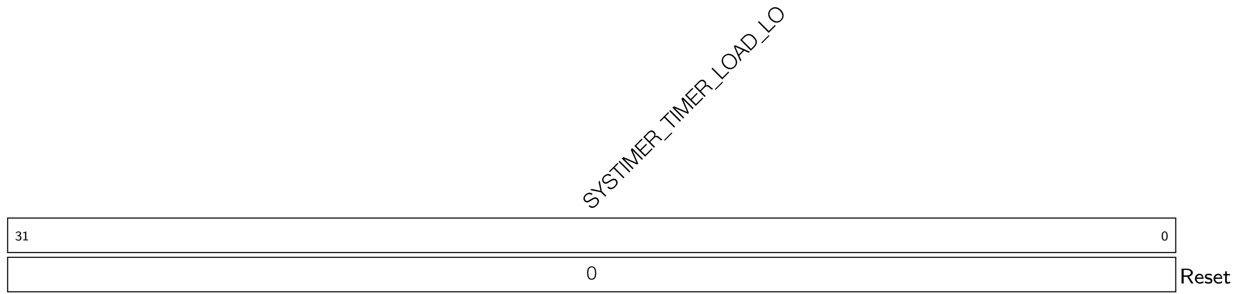
SYSTIMER_TIMER_LOAD Set this bit to 1, the value stored in SYSTIMER_TIMER_LOAD_HI and in SYSTIMER_TIMER_LOAD_LO will be loaded to system timer. (WO)

Register 10.3: SYSTIMER_LOAD_HI_REG (0x0008)

SYSTIMER_TIMER_LOAD_HI	
31 ... 0	
0	Reset

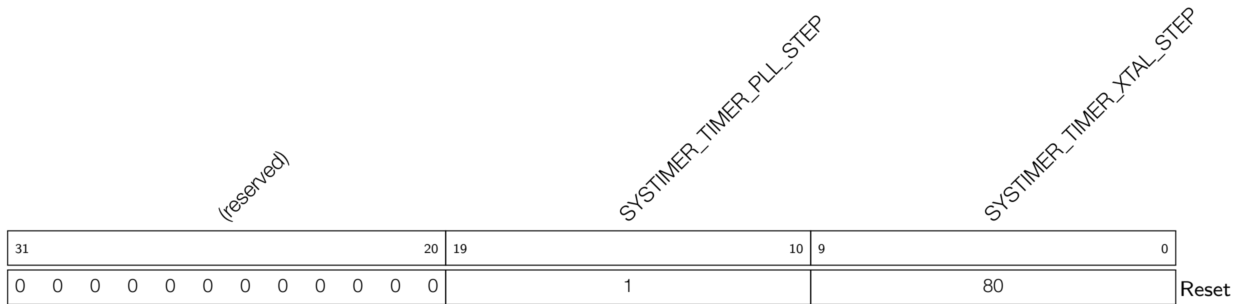
SYSTIMER_TIMER_LOAD_HI The value to be loaded into system timer, high 32 bits. (R/W)

Register 10.4: SYSTIMER_LOAD_LO_REG (0x000C)



SYSTIMER_TIMER_LOAD_LO The value to be loaded into system timer, low 32 bits. (R/W)

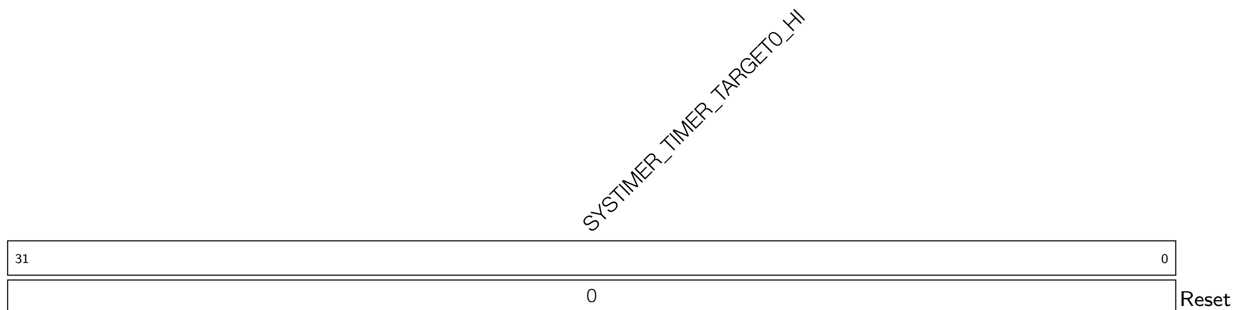
Register 10.5: SYSTIMER_STEP_REG (0x0010)



SYSTIMER_TIMER_XTAL_STEP Set system timer increment step when using XTAL_CLK. (R/W)

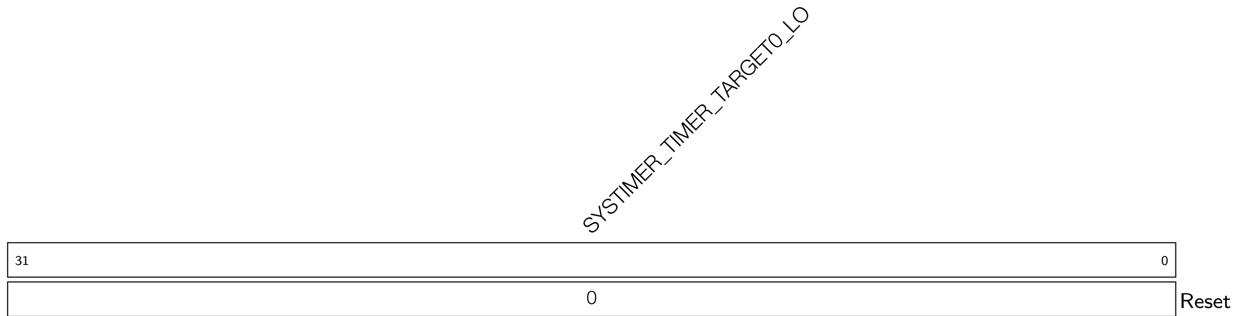
SYSTIMER_TIMER_PLL_STEP Set system timer increment step when using PLL_CLK. (R/W)

Register 10.6: SYSTIMER_TARGET0_HI_REG (0x0014)



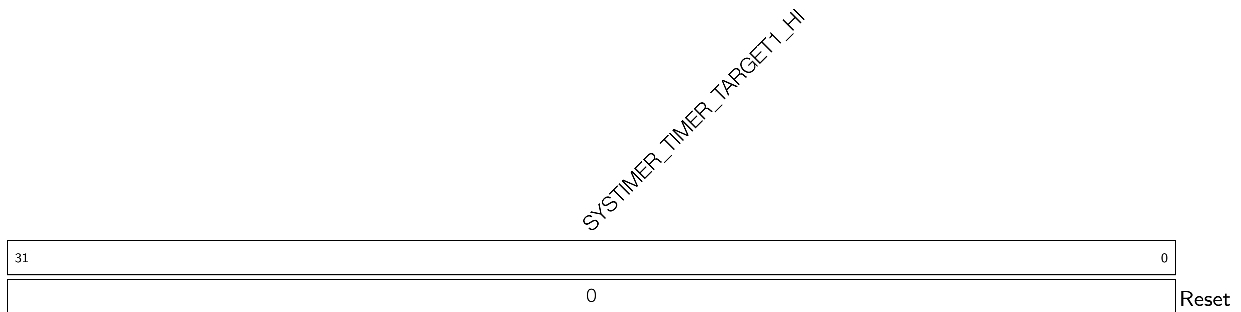
SYSTIMER_TIMER_TARGET0_HI System timer target 0, high 32 bits. (R/W)

Register 10.7: SYSTIMER_TARGET0_LO_REG (0x0018)



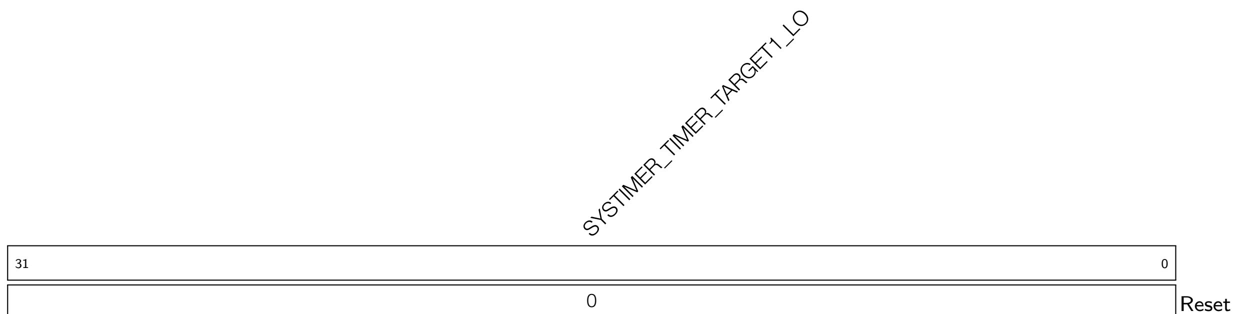
SYSTIMER_TIMER_TARGET0_LO System timer target 0, low 32 bits. (R/W)

Register 10.8: SYSTIMER_TARGET1_HI_REG (0x001C)



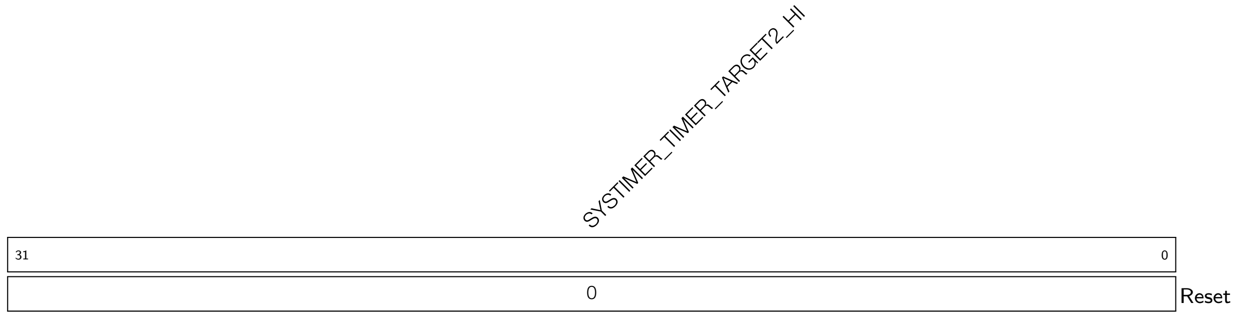
SYSTIMER_TIMER_TARGET1_HI System timer target 1, high 32 bits. (R/W)

Register 10.9: SYSTIMER_TARGET1_LO_REG (0x0020)



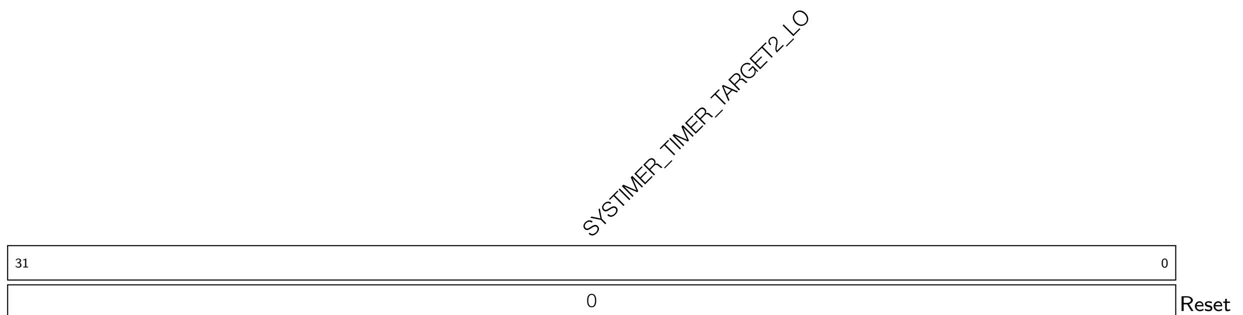
SYSTIMER_TIMER_TARGET1_LO System timer target 1, low 32 bits. (R/W)

Register 10.10: SYSTIMER_TARGET2_HI_REG (0x0024)



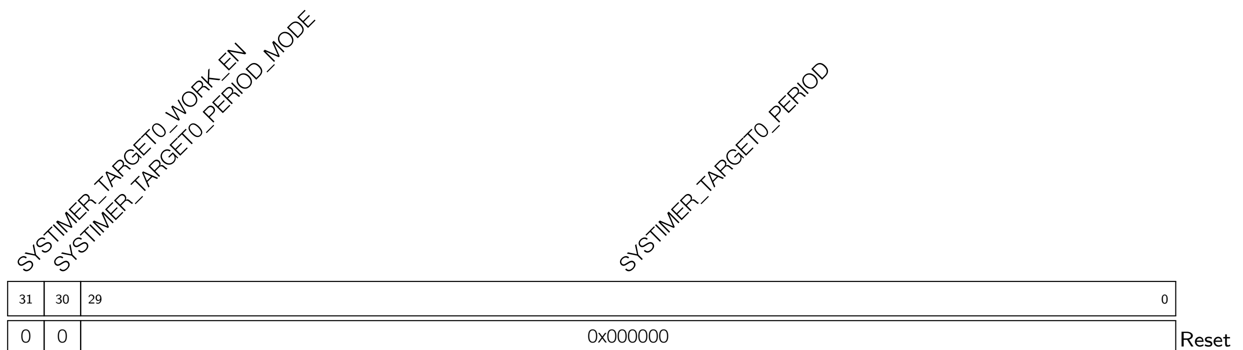
SYSTIMER_TIMER_TARGET2_HI System timer target 2, high 32 bits. (R/W)

Register 10.11: SYSTIMER_TARGET2_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET2_LO System timer target 2, low 32 bits. (R/W)

Register 10.12: SYSTIMER_TARGET0_CONF_REG (0x002C)

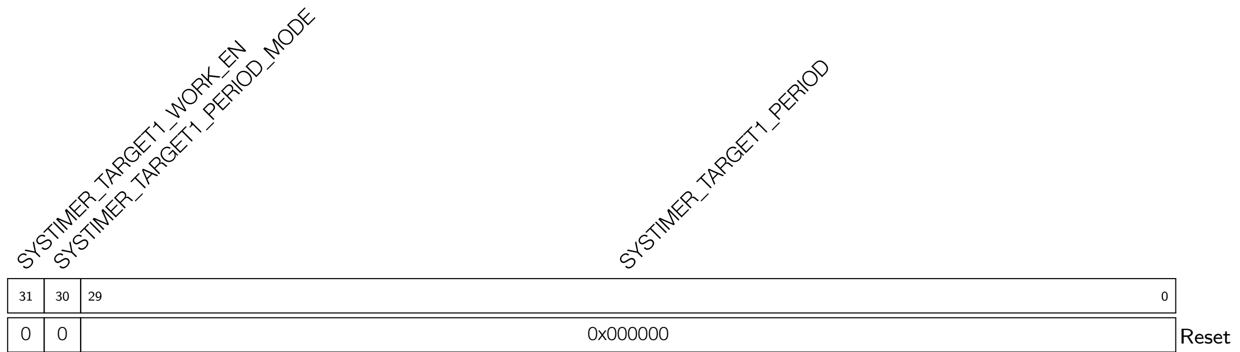


SYSTIMER_TARGET0_PERIOD Set alarm period for system timer target 0, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Set work mode for system timer target 0. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET0_WORK_EN System timer target 0 work enable. (R/W)

Register 10.13: SYSTIMER_TARGET1_CONF_REG (0x0030)

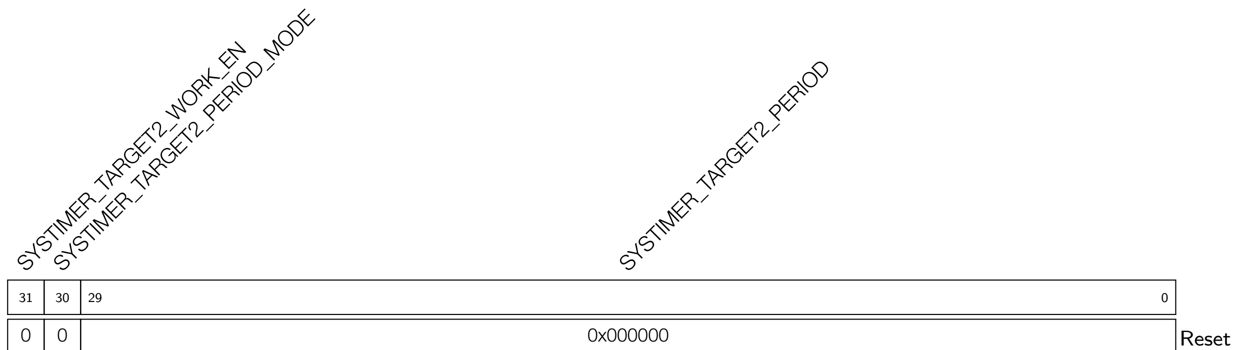


SYSTIMER_TARGET1_PERIOD Set alarm period for system timer target 1, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Set work mode for system timer target 1. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET1_WORK_EN System timer target 1 work enable. (R/W)

Register 10.14: SYSTIMER_TARGET2_CONF_REG (0x0034)



SYSTIMER_TARGET2_PERIOD Set alarm period for sytem timer target 2, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Set work mode for system timer target 2. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET2_WORK_EN System timer target 2 work enable. (R/W)

Register 10.15: SYSTIMER_UPDATE_REG (0x0038)

SYSTIMER_TIMER_UPDATE	SYSTIMER_TIMER_VALUE_VALID	(reserved)	
31	30	29 ... 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SYSTIMER_TIMER_VALUE_VALID Check if it is valid to read out timer value from registers. 0: Not ready to read timer value from registers; 1: Ready to read timer value from registers. (RO)

SYSTIMER_TIMER_UPDATE Update system timer value to registers. (WO)

Register 10.16: SYSTIMER_VALUE_HI_REG (0x003C)

SYSTIMER_TIMER_VALUE_HI	
31 ... 0	
0	Reset

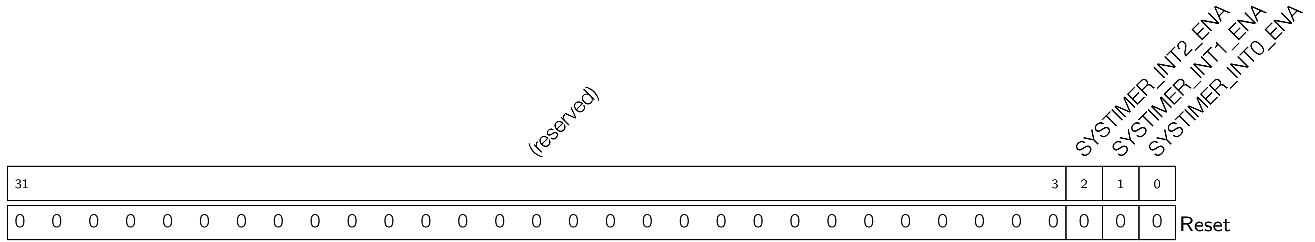
SYSTIMER_TIMER_VALUE_HI System timer value, high 32 bits. (RO)

Register 10.17: SYSTIMER_VALUE_LO_REG (0x0040)

SYSTIMER_TIMER_VALUE_LO	
31 ... 0	
0	Reset

SYSTIMER_TIMER_VALUE_LO System timer value, low 32 bits. (RO)

Register 10.18: SYSTIMER_INT_ENA_REG (0x0044)

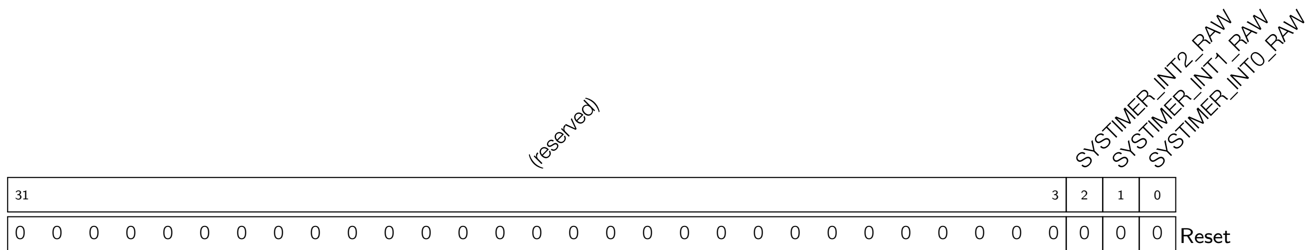


SYSTIMER_INT0_ENA Interrupt enable bit of system timer target 0. (R/W)

SYSTIMER_INT1_ENA Interrupt enable bit of system timer target 1. (R/W)

SYSTIMER_INT2_ENA Interrupt enable bit of system timer target 2. (R/W)

Register 10.19: SYSTIMER_INT_RAW_REG (0x0048)

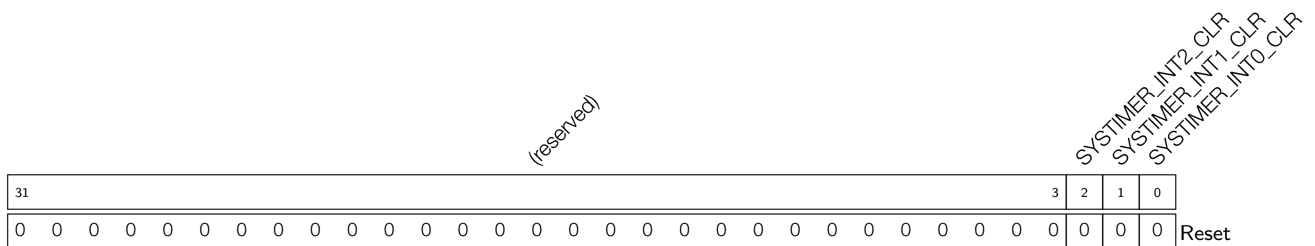


SYSTIMER_INT0_RAW Interrupt raw bit of system timer target 0. (RO)

SYSTIMER_INT1_RAW Interrupt raw bit of system timer target 1. (RO)

SYSTIMER_INT2_RAW Interrupt raw bit of system timer target 2. (RO)

Register 10.20: SYSTIMER_INT_CLR_REG (0x004C)

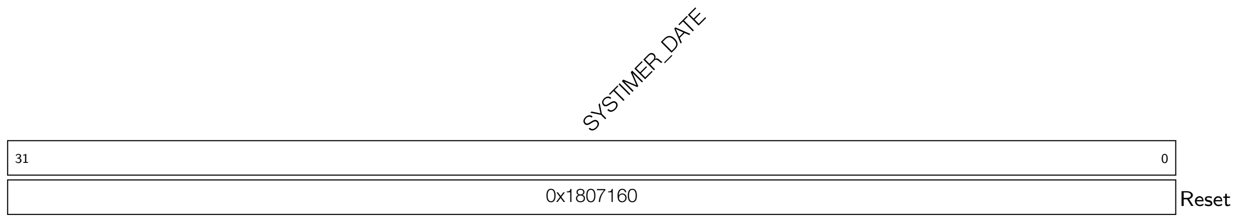


SYSTIMER_INT0_CLR Interrupt clear bit of system timer target 0. (WO)

SYSTIMER_INT1_CLR Interrupt clear bit of system timer target 1. (WO)

SYSTIMER_INT2_CLR Interrupt clear bit of system timer target 2. (WO)

Register 10.21: SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

11. Timer Group (TIMG)

11.1 Overview

General purpose timers can be used to precisely time an interval, trigger an interrupt after a particular interval (periodically and aperiodically), or act as a hardware clock. As shown in Figure 11-1, the ESP32-S2 chip contains two timer groups, namely timer group 0 and timer group 1. Each timer group consists of two general purpose timers referred to as T*x* (where *x* is 0 or 1) and one Main System Watchdog Timer. All general purpose timers are based on 16-bit prescalers and 64-bit auto-reload-capable up/down counters.

Figure 11-1. Timer Units within Groups

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 12: *Watchdog Timers*. Therefore, the term 'timers' within this chapter refers to the general purpose timers.

The timers' features are summarized as follows:

- A 16-bit clock prescaler, from 1 to 65536
- A 64-bit time-base counter programmable to be incrementing or decrementing
- Able to read real-time value of the time-base counter
- Halting and resuming the time-base counter
- Programmable alarm generation
- Timer value reload (Auto-reload at alarm or software-controlled instant reload)
- Level and edge interrupt generation

11.2 Functional Description

11.2.1 16-bit Prescaler and Clock Selection

Each timer can select between the APB clock (APB_CLK) or external clock (XTAL_CLK) as its clock source by setting the TIMG_Tx_USE_XTAL field of the TIMG_TxCONFIG_REG register. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB_CLK) used by the time-base counter. The 16-bit prescaler is configured by the TIMG_Tx_DIVIDER field and can take any value from 1 to 65536. Note that programming a value of 0 in TIMG_Tx_DIVIDER will result in the divisor being 65536.

The timer must be disabled (i.e. TIMG_Tx_EN should be cleared) before modifying the 16-bit prescaler. Modifying the 16-bit prescaler whilst the timer is enabled can lead to unpredictable results.

11.2.2 64-bit Time-based Counter

The 64-bit time-base counters are based on TB_CLK and can the configured to increment or decrement via the TIMG_Tx_INCREASE field. The time-base counter can be enabled/disabled by setting/clearing the TIMG_Tx_EN field. Whilst enabled, the time-base counter will increment/decrement on each cycle of TB_CLK. When disabled, the time-base counter is essentially frozen. Note that the TIMG_Tx_INCREASE field can be changed whilst TIMG_Tx_EN is set and will cause the time-base counter to change direction instantly.

To read the 64-bit current timer value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_TxUPDATE_REG, the current value of the 64-bit timer is instantly latched into the TIMG_TxLO_REG and TIMG_TxHI_REG registers containing the lower and upper 32-bits respectively. TIMG_TxLO_REG and TIMG_TxHI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_TxUPDATE_REG is written to again.

11.2.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 11.2.4). The 64-bit alarm value is configured in the TIMG_TxALARMLO_REG and TIMG_TxALARMHI_REG representing the lower and upper 32-bits of the alarm value respectively. However, the configured alarm value is ineffective until the alarm is enabled by setting the TIMG_Tx_ALARM_EN field. In order to simply the scenario where the alarm is enabled 'too late' (i.e. the timer value has already passed the alarm value when the alarm is enabled), the alarm value will also trigger immediately if the current timer value is larger/smaller than the alarm value for an up-counting/down-counting timer.

When an alarm occurs, the TIMG_Tx_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_Tx_ALARM_EN is set.

11.2.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_Tx_LOAD_LO and TIMG_Tx_LOAD_HI registers that correspond to the lower and upper 32-bits of the timer's new value respectively. However, writing a reload value to TIMG_Tx_LOAD_LO and TIMG_Tx_LOAD_HI will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to TIMG_TxLOAD_REG causing the timer's current value to be instantly reloaded. If TIMG_Tx_EN is set, the timer will continue incrementing/decrementing from the new value. If TIMG_Tx_EN is cleared, the timer will remain frozen at the new value until counting is

re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs thus allowing the timer to continue incrementing/decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the TIMG_Tx_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment/decrement past the alarm value after an alarm.

11.2.5 Interrupts

Each timer has its own pair of interrupt lines (for edge and level interrupts) that can be routed to the CPU. Thus, there are a total of six interrupt lines per timer group and they are named as follows:

- TIMG_WDT_LEVEL_INT: Level interrupt line for the watchdog timer in the group, generated when a watchdog timer interrupt stage times out.
- TIMG_WDT_EDGE_INT: Edge interrupt line for the watchdog timer in the group, generated when a watchdog timer interrupt stage times out.
- TIMG_Tx_LEVEL_INT: Level interrupt for one of the general purpose timers, generated when an alarm event happens.
- TIMG_Tx_EDGE_INT: Edge interrupt for one of the general purpose timer, generated when an alarm event happens.

Interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupt lines will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. Conversely, edge interrupts will generate a short pulse after an alarm (or stage timeout) occurs. To enable a timer's level or edge interrupt lines, the TIMG_Tx_LEVEL_INT_EN or TIMG_Tx_EDGE_INT_EN bits should be set respectively.

The interrupts of each timer group are governed by a set of registers. Each timer within the group will have a corresponding bit in each of these registers:

- TIMG_Tx_INT_RAW : An alarm event sets it to 1. The bit will remain set until writing to the timer's corresponding bit in TIMG_Tx_INT_CLR.
- TIMG_WDT_INT_RAW : A stage time out will set the timer's bit to 1. The bit will remain set until writing to the timer's corresponding bit in TIMG_WDT_INT_CLR.
- TIMG_Tx_INT_ST : Reflects the status of each timer's interrupt and is generated by masking the bits of TIMG_Tx_INT_RAW with TIMG_Tx_INT_ENA. For level interrupts, these bits reflect the level on the watchdog timer's level interrupt line.
- TIMG_WDT_INT_ST : Reflects the status of each watchdog timer's interrupt and is generated by masking the bits of TIMG_WDT_INT_RAW with TIMG_WDT_INT_ENA. For level interrupts, these bits reflect the level on the watchdog timer's level interrupt line.
- TIMG_Tx_INT_ENA : Used to enable or mask the interrupt status bits of timers within the group.
- TIMG_WDT_INT_ENA : Used to enable or mask the interrupt status bits of watchdog timer within the group.
- TIMG_Tx_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The timer's corresponding bit in TIMG_Tx_INT_RAW and TIMG_Tx_INT_ST will be cleared as a result. Note that a timer's interrupt must be cleared before the next interrupt occurs when using level interrupts.
- TIMG_WDT_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The watchdog timer's corresponding bit in TIMG_WDT_INT_RAW and TIMG_WDT_INT_ST will be cleared as a result.

Note that a watchdog timer's interrupt must be cleared before the next interrupt occurs when using level interrupts.

11.3 Configuration and Usage

11.3.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting TIMG_Tx_USE_XTAL field.
   - Configure the 16-bit prescaler by setting TIMG_Tx_DIVIDER.
   - Configure the timer direction by setting/clearing TIMG_Tx_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_Tx_LOAD_LO and TIMG_Tx_LOAD_HI, then reloading it into the timer by writing any value to TIMG_TxLOAD_REG.
2. Start the timer by setting TIMG_Tx_EN.
3. Get the timer's current value.
   - Write any value to TIMG_TxUPDATE_REG to latch the timer's current value.
   - Read the latched timer value from TIMG_TxLO_REG and TIMG_TxHI_REG.

11.3.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 11.3.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_TxALARMLO_REG and TIMG_TxALARMHI_REG.
   - Enable interrupt by setting TIMG_Tx_LEVEL_INT_EN or TIMG_Tx_EDGE_INT_EN for level or edge interrupts respectively.
3. Disable auto reload by clearing TIMG_Tx_AUTORELOAD.
4. Start the timer by setting TIMG_Tx_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_Tx_INT_CLR.
   - Disable the timer by clearing TIMG_Tx_EN.

11.3.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 of Section 11.3.1.
2. Configure the alarm following step 2 of Section 11.3.2.
3. Enable auto reload by setting TIMG_Tx_AUTORELOAD and setting the reload value in TIMG_Tx_LOAD_LO and TIMG_Tx_LOAD_HI.
4. Start the timer by setting TIMG_Tx_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_Tx_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e. different alarm interval per iteration), then TIMG_TxALARMLO_REG, TIMG_TxALARMHI_REG, TIMG_Tx_LOAD_LO, and TIMG_Tx_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_Tx_ALARM_EN.
6. Stop the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_Tx_INT_CLR.
   - Disable the timer by clearing TIMG_Tx_EN.

11.4 Base Address

Users can access the 64-bit Timer with four base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 69: 64-bit Timers Base Address

Module	Bus to Access Peripheral	Base Address
TIMG0	PeriBUS1	0x3F41F000
	PeriBUS2	0x6001F000
TIMG1	PeriBUS1	0x3F420000
	PeriBUS2	0x60020000

11.5 Register Summary

The addresses in the following table are relative to the 64-bit Timer base addresses provided in Section 11.4.

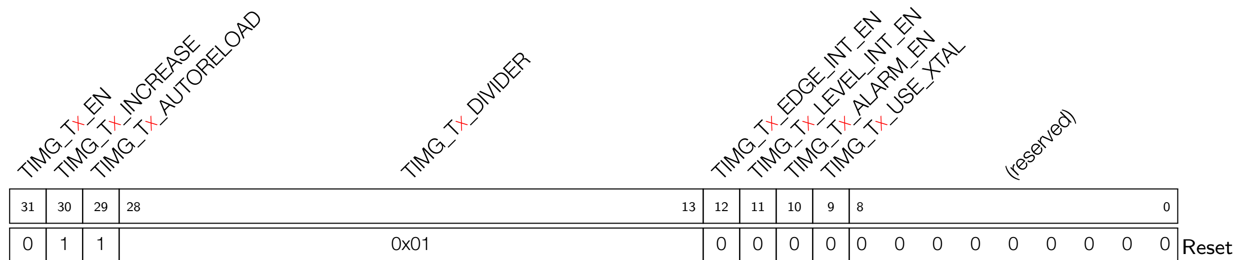
Name	Description	Address	Access
Timer 0 Configuration and Control Register			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	R/W
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 32 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMG_T0_(LO/HI)_REG	0x000C	R/W
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x0014	R/W

Name	Description	Address	Access
TIMG_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_T0LOADHI_REG	Timer 0 reload value, high 32 bits	0x001C	R/W
TIMG_T0LOAD_REG	Write to reload timer from TIMG_T0_(LOADLO/LOADHI)_REG	0x0020	WO
Timer 1 Configuration and Control Register			
TIMG_T1CONFIG_REG	Timer 1 configuration register	0x0024	R/W
TIMG_T1LO_REG	Timer 1 current value, low 32 bits	0x0028	RO
TIMG_T1HI_REG	Timer 1 current value, high 32 bits	0x002C	RO
TIMG_T1UPDATE_REG	Write to copy current timer value to TIMG_T1_(LO/HI)_REG	0x0030	R/W
TIMG_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x0034	R/W
TIMG_T1ALARMHI_REG	Timer 1 alarm value, high bits	0x0038	R/W
TIMG_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x003C	R/W
TIMG_T1LOADHI_REG	Timer 1 reload value, high 32 bits	0x0040	R/W
TIMG_T1LOAD_REG	Write to reload timer from TIMG_T0_(LOADLO/LOADHI)_REG	0x0044	WO
Configuration and Control Register for WDT			
TIMG_WDTCONFIG0_REG	Watchdog timer configuration register	0x0048	R/W
TIMG_WDTCONFIG1_REG	Watchdog timer prescaler register	0x004C	R/W
TIMG_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x0050	R/W
TIMG_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x0054	R/W
TIMG_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x0058	R/W
TIMG_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x005C	R/W
TIMG_WDTFEED_REG	Write to feed the watchdog timer	0x0060	WO
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
Configuration and Control Register for RTC CALI			
TIMG_RTCCALICFG_REG	RTC calibration configuration register	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC calibration configuration register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	Timer group calibration register	0x00A8	varies
Configuration and Control Register for LACT			
TIMG_LACTCONFIG_REG	LACT configuration register	0x0070	R/W
TIMG_LACTRTC_REG	LACT RTC register	0x0074	R/W
TIMG_LACTLO_REG	LACT low register	0x0078	RO
TIMG_LACTHI_REG	LACT high register	0x007C	RO
TIMG_LACTUPDATE_REG	LACT update register	0x0080	WO
TIMG_LACTALARMLO_REG	LACT alarm low register	0x0084	R/W
TIMG_LACTALARMHI_REG	LACT alarm high register	0x0088	R/W
TIMG_LACTLOADLO_REG	LACT load low register	0x008C	R/W
TIMG_LACTLOADHI_REG	Timer LACT load high register	0x0090	R/W
TIMG_LACTLOAD_REG	Timer LACT load register	0x0094	WO
Interrupt Register			
TIMG_INT_ENA_TIMERS_REG	Interrupt enable bits	0x0098	R/W
TIMG_INT_RAW_TIMERS_REG	Raw interrupt status	0x009C	RO

Name	Description	Address	Access
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x00A0	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x00A4	WO
Version Register			
TIMG_TIMERS_DATE_REG	Version control register	0x00F8	R/W
Configuration Register			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

11.6 Registers

Register 11.1: TIMG_TxCONFIG_REG (x: 0-1) (0x0000+0x24*x)



TIMG_Tx_USE_XTAL 1: Use XTAL_CLK as the source clock of timer group. 0: Use APB_CLK as the source clock of timer group. (R/W)

TIMG_Tx_ALARM_EN When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W)

TIMG_Tx_LEVEL_INT_EN When set, an alarm will generate a level type interrupt. (R/W)

TIMG_Tx_EDGE_INT_EN When set, an alarm will generate an edge type interrupt. (R/W)

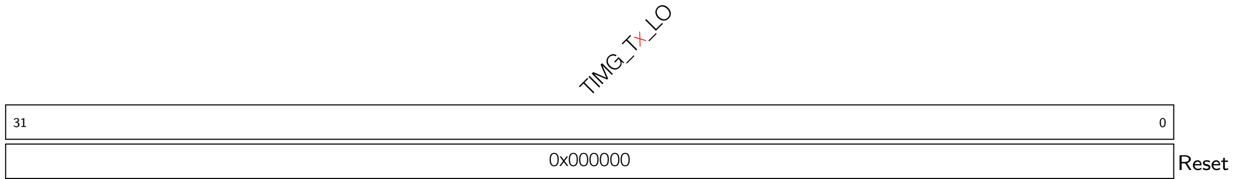
TIMG_Tx_DIVIDER Timer x clock (Tx_clk) prescaler value. (R/W)

TIMG_Tx_AUTORELOAD When set, timer x auto-reload at alarm is enabled. (R/W)

TIMG_Tx_INCREASE When set, the timer x time-base counter will increment every clock tick. When cleared, the timer x time-base counter will decrement. (R/W)

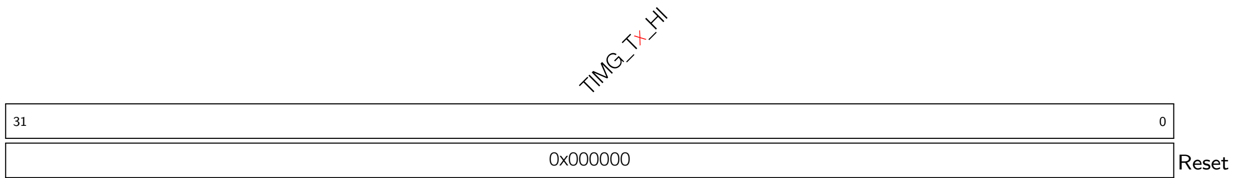
TIMG_Tx_EN When set, the timer x time-base counter is enabled. (R/W)

Register 11.2: TIMG_TxLO_REG (x: 0-1) (0x0004+0x24*x)



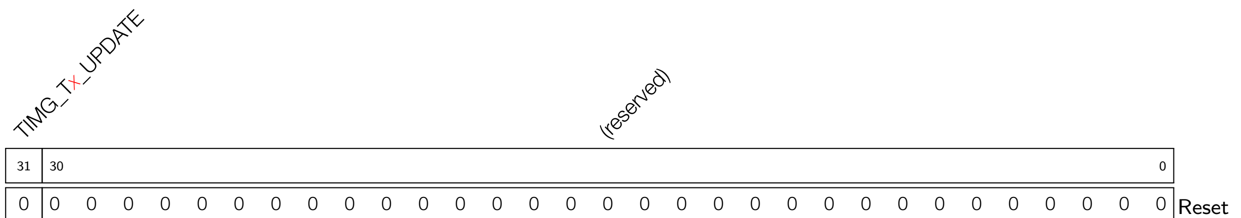
TIMG_Tx_LO After writing to TIMG_TxUPDATE_REG, the low 32 bits of the time-base counter of timer x can be read here. (RO)

Register 11.3: TIMG_TxHI_REG (x: 0-1) (0x0008+0x24*x)



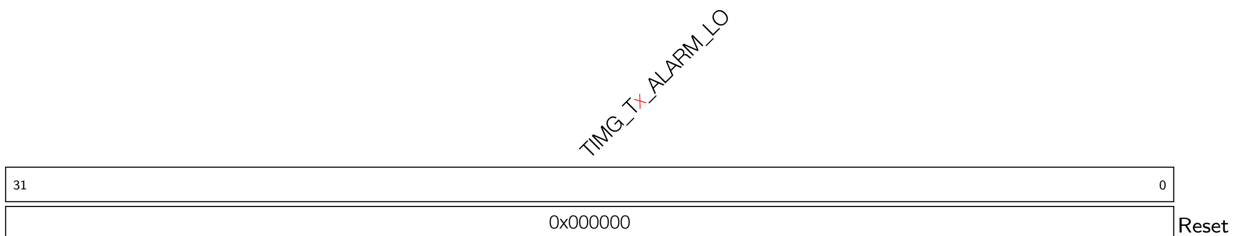
TIMG_Tx_HI After writing to TIMG_TxUPDATE_REG, the high 32 bits of the time-base counter of timer x can be read here. (RO)

Register 11.4: TIMG_TxUPDATE_REG (x: 0-1) (0x000C+0x24*x)



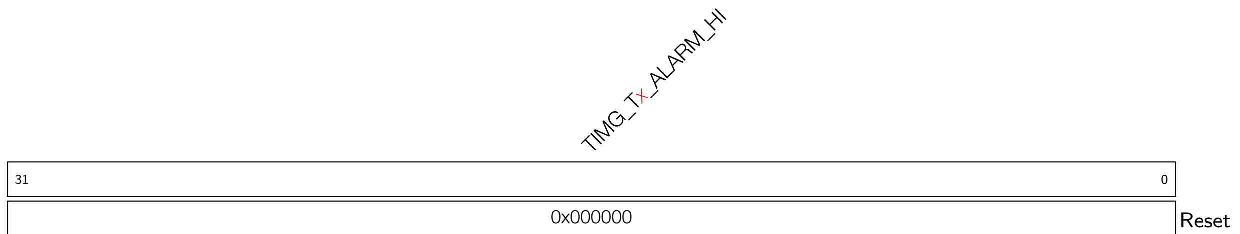
TIMG_Tx_UPDATE After writing 0 or 1 to TIMG_TxUPDATE_REG, the counter value is latched. (R/W)

Register 11.5: TIMG_TxALARMLO_REG (x: 0-1) (0x0010+0x24*x)



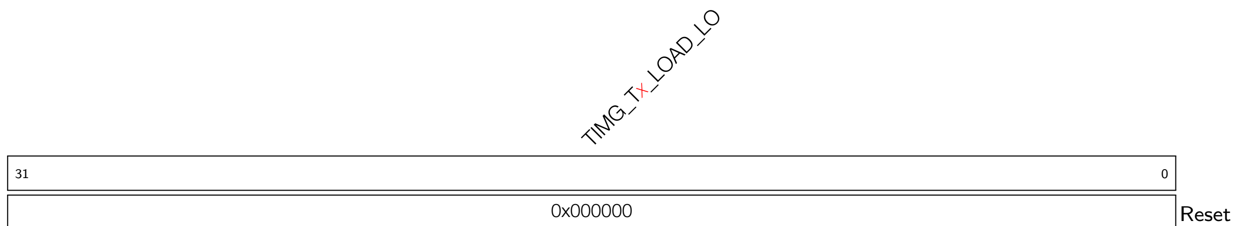
TIMG_Tx_ALARM_LO Timer x alarm trigger time-base counter value, low 32 bits. (R/W)

Register 11.6: TIMG_TxALARMHI_REG (x: 0-1) (0x0014+0x24*x)



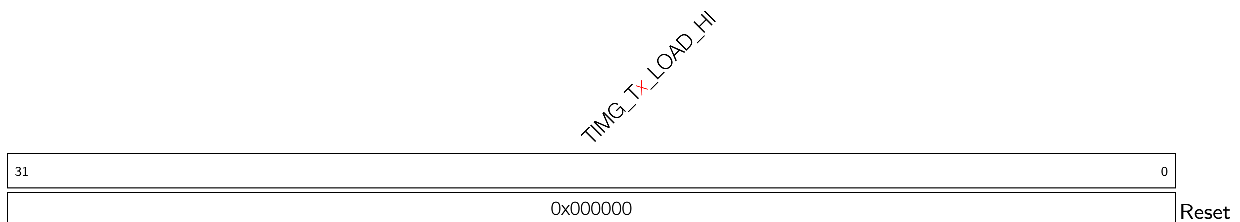
TIMG_Tx_ALARM_HI Timer x alarm trigger time-base counter value, high 32 bits. (R/W)

Register 11.7: TIMG_TxLOADLO_REG (x: 0-1) (0x0018+0x24*x)



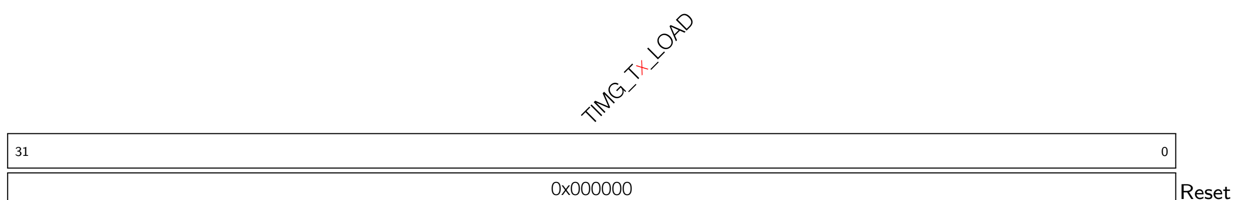
TIMG_Tx_LOAD_LO Low 32 bits of the value that a reload will load onto timer x time-base counter. (R/W)

Register 11.8: TIMG_TxLOADHI_REG (x: 0-1) (0x001C+0x24*x)



TIMG_Tx_LOAD_HI High 32 bits of the value that a reload will load onto timer x time-base counter. (R/W)

Register 11.9: TIMG_TxLOAD_REG (x: 0-1) (0x0020+0x24*x)



TIMG_Tx_LOAD Write any value to trigger a timer x time-base counter reload. (WO)

Register 11.10: TIMG_WDTCONFIG0_REG (0x0048)

TIMG_WDT_EN	TIMG_WDT_STG0	TIMG_WDT_STG1	TIMG_WDT_STG2	TIMG_WDT_STG3	TIMG_WDT_EDGE_INT_EN	TIMG_WDT_LEVEL_INT_EN	TIMG_WDT_CPU_RESET_LENGTH	TIMG_WDT_SYS_RESET_LENGTH	TIMG_WDT_FLASHBOOT_MOD_EN	TIMG_WDT_PROCPU_RESET_EN	TIMG_WDT_APPCPU_RESET_EN	(reserved)	
31	30 29	28 27	26 25	24 23	22	21	20 18	17 15	14	13	12	11 0	
0	0	0	0	0	0	0	0x1	0x1	1	0	0	0 0 0 0 0 0 0 0 0 0 0 0	Reset

TIMG_WDT_APPCPU_RESET_EN Reserved. (R/W)

TIMG_WDT_PROCPU_RESET_EN WDT reset CPU enable. (R/W)

TIMG_WDT_FLASHBOOT_MOD_EN When set, Flash boot protection is enabled. (R/W)

TIMG_WDT_SYS_RESET_LENGTH System reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 us, 7: 3.2 us. (R/W)

TIMG_WDT_CPU_RESET_LENGTH CPU reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 us, 7: 3.2 us. (R/W)

TIMG_WDT_LEVEL_INT_EN When set, a level type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG_WDT_EDGE_INT_EN When set, an edge type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG_WDT_STG3 Stage 3 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

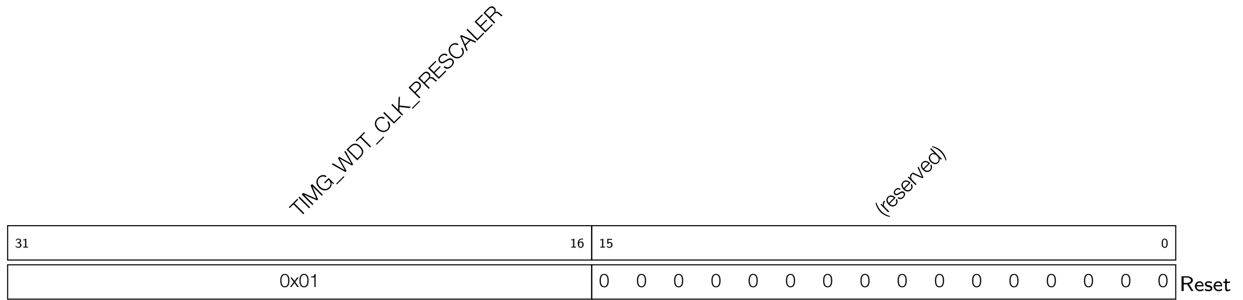
TIMG_WDT_STG2 Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG_WDT_STG1 Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG_WDT_STG0 Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

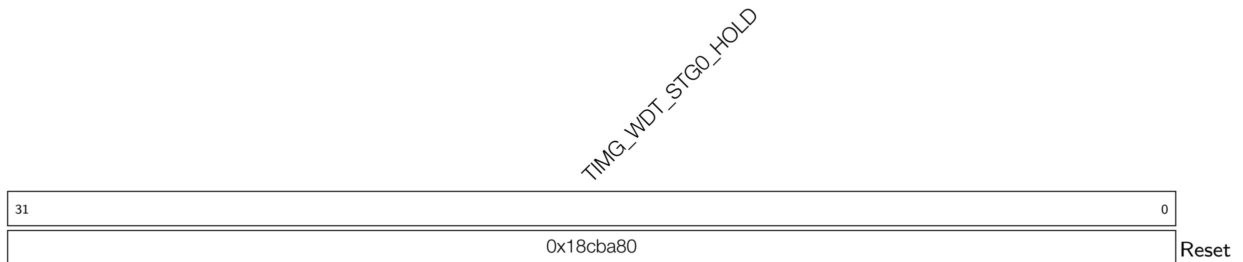
TIMG_WDT_EN When set, MWDT is enabled. (R/W)

Register 11.11: TIMG_WDTCONFIG1_REG (0x004C)



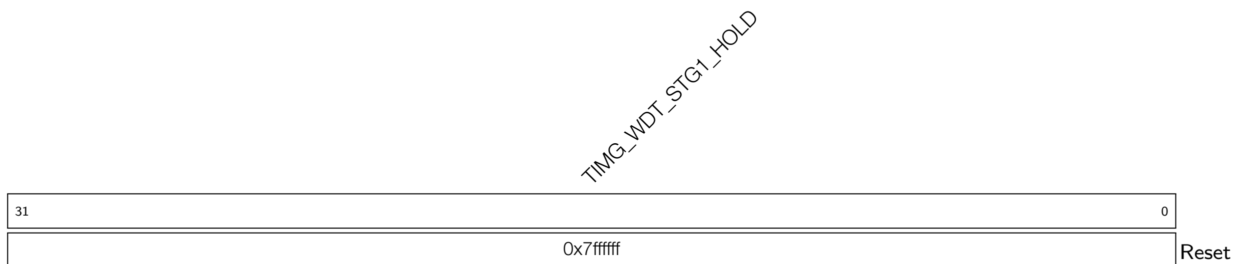
TIMG_WDT_CLK_PRESCALER MWDT clock prescaler value. MWDT clock period = 12.5 ns * TIMG_WDT_CLK_PRESCALE. (R/W)

Register 11.12: TIMG_WDTCONFIG2_REG (0x0050)



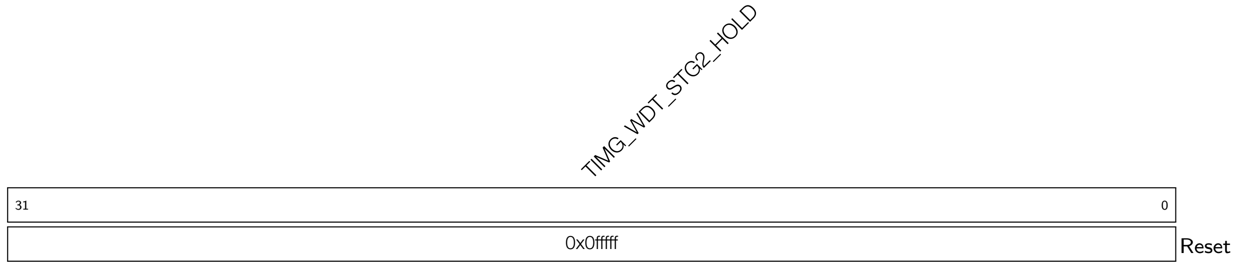
TIMG_WDT_STG0_HOLD Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 11.13: TIMG_WDTCONFIG3_REG (0x0054)



TIMG_WDT_STG1_HOLD Stage 1 timeout value, in MWDT clock cycles. (R/W)

Register 11.14: TIMG_WDTCONFIG4_REG (0x0058)

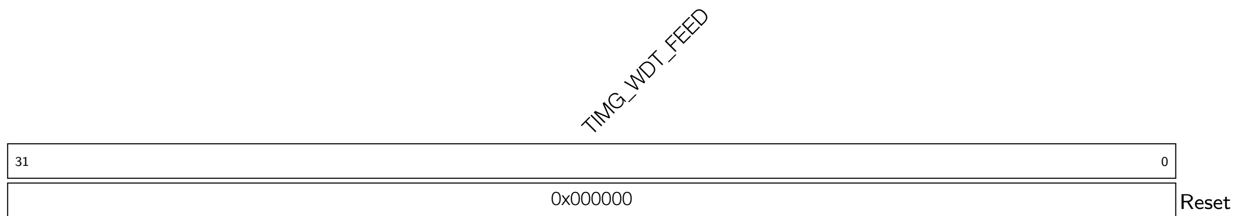


TIMG_WDT_STG2_HOLD Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 11.15: TIMG_WDTCONFIG5_REG (0x005C)

TIMG_WDT_STG3_HOLD Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 11.16: TIMG_WDTFEED_REG (0x0060)



TIMG_WDT_FEED Write any value to feed the MWDT. (WO)

Register 11.17: TIMG_WDTWPROTECT_REG (0x0064)

TIMG_WDT_WKEY If the register contains a different value than its reset value, write protection is enabled. (R/W)

Register 11.18: TIMG_RTCCALICFG_REG (0x0068)

31	30 … 16	15	14 … 13	12	11 … 0	
TIMG_RTC_CALI_START	TIMG_RTC_CALI_MAX	TIMG_RTC_CALI_RDY	TIMG_RTC_CALI_CLK_SEL	TIMG_RTC_CALI_START_CYCLING	(reserved)	
0	0x01	0	0x1	1	0 0 0 0 0 0 0 0 0 0 0 0	Reset

TIMG_RTC_CALI_START_CYCLING When set, periodic calibration is enabled. (R/W)

TIMG_RTC_CALI_CLK_SEL Used to select the clock to be calibrated. 0: RTC_CLK. 1: RTC20M_D256_CLK. 2: XTAL32K_CLK. (R/W)

TIMG_RTC_CALI_RDY Set this bit to mark the completion of calibration. (RO)

TIMG_RTC_CALI_MAX Calibration time, in cycles of the clock to be calibrated. (R/W)

TIMG_RTC_CALI_START Set this bit to starts calibration. (R/W)

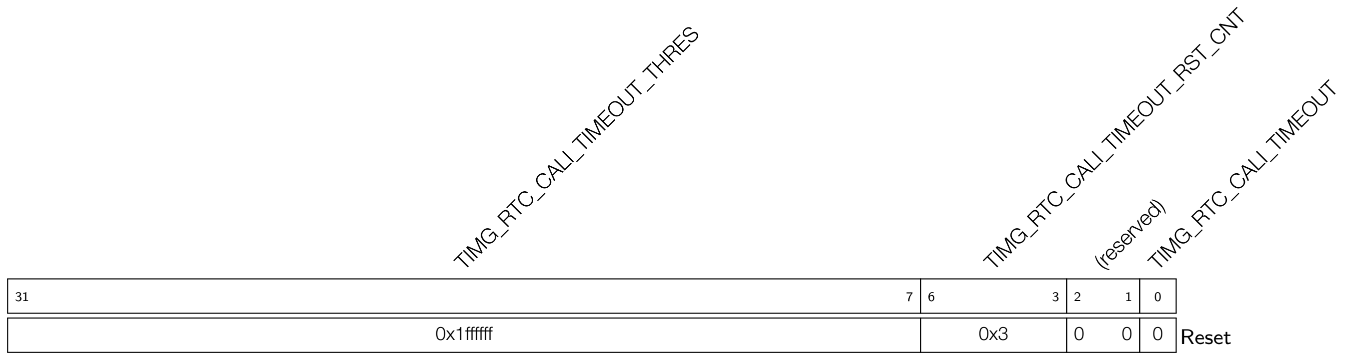
Register 11.19: TIMG_RTCCALICFG1_REG (0x006C)

31 … 7	6 … 1	0	
TIMG_RTC_CALI_VALUE	(reserved)	TIMG_RTC_CALI_CYCLING_DATA_VLD	
0x00000	0 0 0 0 0 0	0	Reset

TIMG_RTC_CALI_CYCLING_DATA_VLD Periodic calibration valid signal. (RO)

TIMG_RTC_CALI_VALUE Calibration value when cycles of clock to be calibrated reach TIMG_RTC_CALI_MAX, in unit of XTAL_CLK clock cycles. (RO)

Register 11.20: TIMG_RTCCALICFG2_REG (0x00A8)

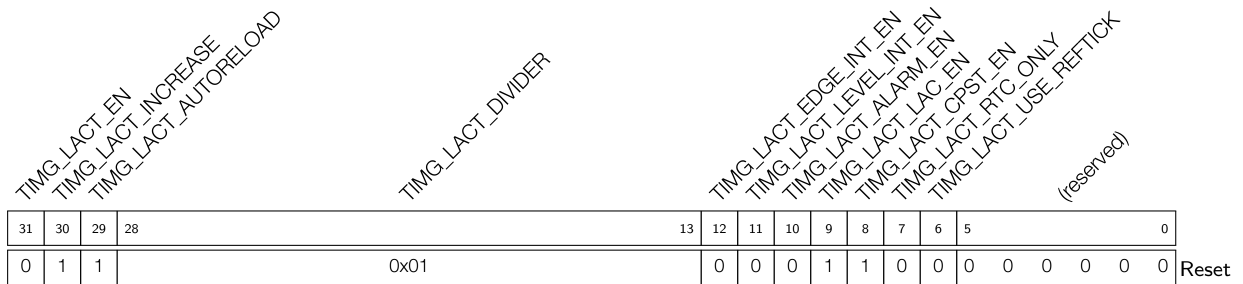


TIMG_RTC_CALI_TIMEOUT RTC calibration timeout indicator. (RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT Cycles that release calibration timeout reset. (R/W)

TIMG_RTC_CALI_TIMEOUT_THRES Threshold value for the RTC calibration timer. If the calibration timer's value exceeds this threshold, a timeout is triggered. (R/W)

Register 11.21: TIMG_LACTCONFIG_REG (0x0070)



TIMG_LACT_USE_REFTICK Reserved. (R/W)

TIMG_LACT_RTC_ONLY Reserved. (R/W)

TIMG_LACT_CPST_EN Reserved. (R/W)

TIMG_LACT_LAC_EN Reserved. (R/W)

TIMG_LACT_ALARM_EN Reserved. (R/W)

TIMG_LACT_LEVEL_INT_EN Reserved. (R/W)

TIMG_LACT_EDGE_INT_EN Reserved. (R/W)

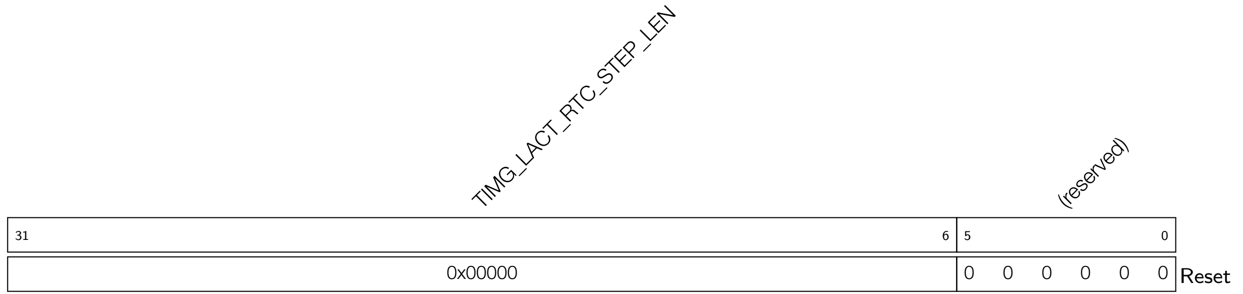
TIMG_LACT_DIVIDER Reserved. (R/W)

TIMG_LACT_AUTORELOAD Reserved. (R/W)

TIMG_LACT_INCREASE Reserved. (R/W)

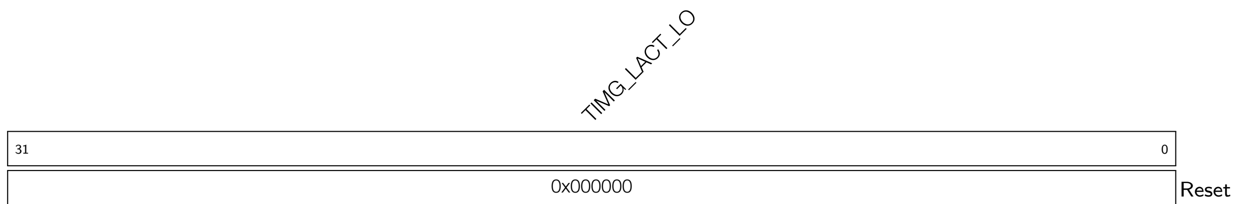
TIMG_LACT_EN Reserved. (R/W)

Register 11.22: TIMG_LACTRTC_REG (0x0074)



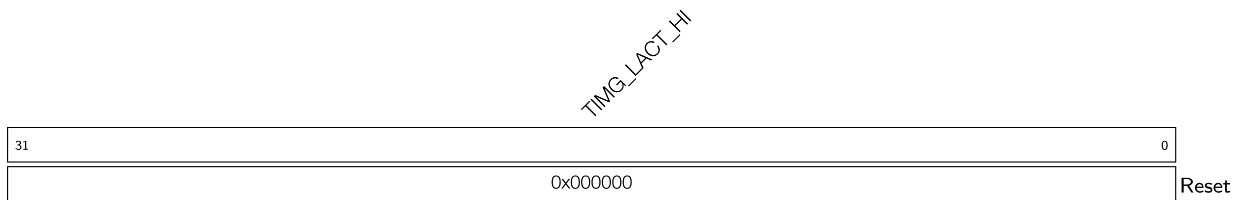
TIMG_LACT_RTC_STEP_LEN Reserved. (R/W)

Register 11.23: TIMG_LACTLO_REG (0x0078)



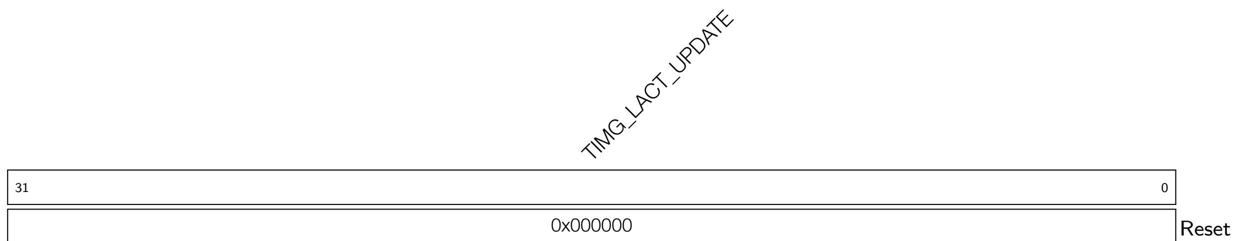
TIMG_LACT_LO Reserved. (RO)

Register 11.24: TIMG_LACTHI_REG (0x007C)



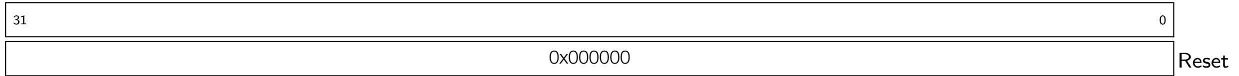
TIMG_LACT_HI Reserved. (RO)

Register 11.25: TIMG_LACTUPDATE_REG (0x0080)



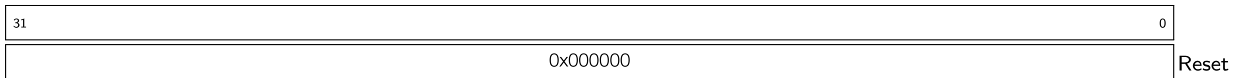
TIMG_LACT_UPDATE Reserved. (WO)

Register 11.26: TIMG_LACTALARMLO_REG (0x0084)



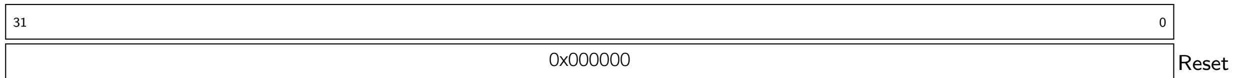
TIMG_LACT_ALARM_LO Reserved. (R/W)

Register 11.27: TIMG_LACTALARMHI_REG (0x0088)



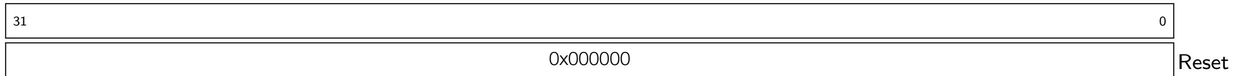
TIMG_LACT_ALARM_HI Reserved. (R/W)

Register 11.28: TIMG_LACTLOADLO_REG (0x008C)



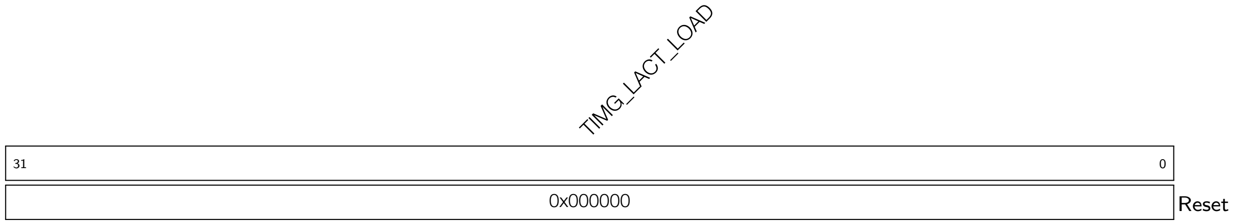
TIMG_LACT_LOAD_LO Reserved. (R/W)

Register 11.29: TIMG_LACTLOADHI_REG (0x0090)



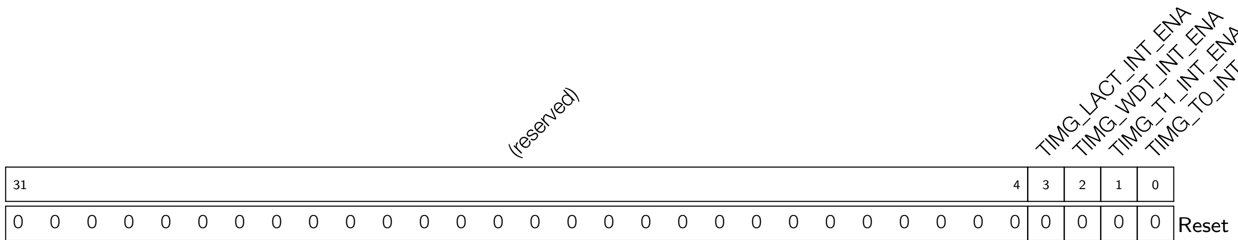
TIMG_LACT_LOAD_HI Reserved. (R/W)

Register 11.30: TIMG_LACTLOAD_REG (0x0094)



TIMG_LACT_LOAD Reserved. (WO)

Register 11.31: TIMG_INT_ENA_TIMERS_REG (0x0098)

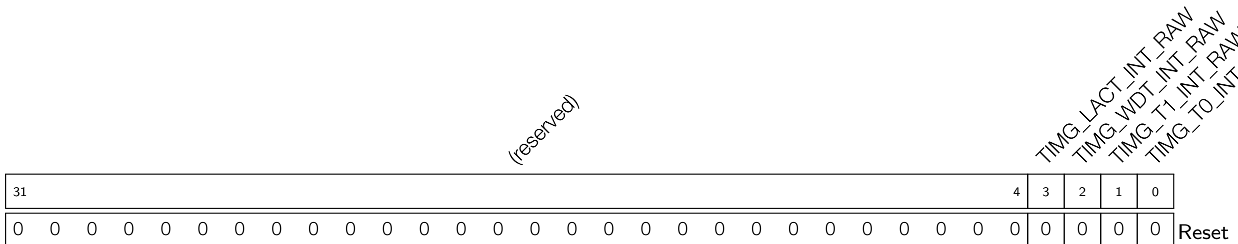


TIMG_Tx_INT_ENA The interrupt enable bit for the TIMG_Tx_INT interrupt. (R/W)

TIMG_WDT_INT_ENA The interrupt enable bit for the TIMG_WDT_INT interrupt. (R/W)

TIMG_LACT_INT_ENA The interrupt enable bit for the TIMG_LACT_INT interrupt. (R/W)

Register 11.32: TIMG_INT_RAW_TIMERS_REG (0x009C)



TIMG_Tx_INT_RAW The raw interrupt status bit for the TIMG_Tx_INT interrupt. (RO)

TIMG_WDT_INT_RAW The raw interrupt status bit for the TIMG_WDT_INT interrupt. (RO)

TIMG_LACT_INT_RAW The raw interrupt status bit for the TIMG_LACT_INT interrupt. (RO)

Register 11.33: TIMG_INT_ST_TIMERS_REG (0x00A0)

(reserved)	TIMG_LACT_INT_ST	TIMG_WDT_INT_ST	TIMG_T1_INT_ST	TIMG_T0_INT_ST	
31 ... 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

TIMG_Tx_INT_ST The masked interrupt status bit for the TIMG_Tx_INT interrupt. (RO)

TIMG_WDT_INT_ST The masked interrupt status bit for the TIMG_WDT_INT interrupt. (RO)

TIMG_LACT_INT_ST The masked interrupt status bit for the TIMG_LACT_INT interrupt. (RO)

Register 11.34: TIMG_INT_CLR_TIMERS_REG (0x00A4)

(reserved)	TIMG_LACT_INT_CLR	TIMG_WDT_INT_CLR	TIMG_T1_INT_CLR	TIMG_T0_INT_CLR	
31 ... 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

TIMG_Tx_INT_CLR Set this bit to clear the TIMG_Tx_INT interrupt. (WO)

TIMG_WDT_INT_CLR Set this bit to clear the TIMG_WDT_INT interrupt. (WO)

TIMG_LACT_INT_CLR Set this bit to clear the TIMG_LACT_INT interrupt. (WO)

Register 11.35: TIMG_TIMERS_DATE_REG (0x00F8)

(reserved)	TIMG_TIMERS_DATE	
31 ... 28	27 ... 0	
0 0 0 0	0x1907261	Reset

TIMG_TIMERS_DATE Version control register. (R/W)

Register 11.36: TIMG_REGCLK_REG (0x00FC)

31	30 ... 0	
TIMG_CLK_EN	(reserved)	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

TIMG_CLK_EN Register clock gate signal. 1: Registers can be read and written to by software. 0: Registers can not be read or written to by software. (R/W)

12. Watchdog Timers (WDT)

12.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g. is stuck in a software loop or in overdue events) will fail to feed the watchdog thus trigger a watchdog time out. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

The ESP32-S2 contains three watchdog timers: one in each of the two timer groups (called Main System Watchdog Timers, or MWDT) and one in the RTC Module (called the RTC Watchdog Timer, or RWDT). Each watchdog timer allows for four separately configurable stages and each stage can be programmed to take one of three (or four for RWDT) actions upon expiry, unless the watchdog is fed or disabled. The actions upon expiry are: interrupt, CPU reset, core reset and system reset. Only RWDT can trigger a system reset that will reset the entire digital circuits, which is the main system including the RTC itself. A timeout value can be set for each stage individually.

In flash boot mode, RWDT and the first MWDT are enabled by default in order to detect and recover from booting errors.

Note that while this chapter provides the functional descriptions of the watchdog timer's, their register descriptions are provided in Chapter 11: *Timer Group (TIMG)*.

12.2 Features

Watchdog timers have the following features:

- Four stages, each with a programmable timeout value. Each stage can be configured and enabled/disabled separately
- One of three/four (for MWDTs/ RWDT) possible actions (interrupt, CPU reset, core reset and system reset) available upon expiry of each stage
- 32-bit expiry counter
- Write protection, to prevent RWDT and MWDT configuration from being altered inadvertently
- Flash boot protection
  If the boot process from an SPI flash does not complete within a predetermined period of time, the watchdog will reboot the entire main system.

12.3 Functional Description

12.3.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter. The clock source of MWDTs is derived from the APB clock via a pre-MWDT 16-bit configurable prescaler. Conversely, the clock source of RWDT is derived directly from a RTC slow clock (without a prescaler) which is usually running at 32 kHz. The 16-bit prescaler for MWDTs is configured via the TIMG_WDT_CLK_PRESCALER field of TIMG_WDTCONFIG1_REG.

MWDTs and RWDT are enabled by setting the TIMG_WDT_EN and RTC_CNTL_WDT_EN fields respectively. When enabled, the 32-bit counters of each watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e. expiry of the current stage). When this occurs, the current counter value

is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to TIMG_WDTFEED_REG for MDWTs and RTC_CNTL_RTC_WDT_FEED for RWDT.

12.3.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding expiry action. When one stage expires, the expiry action is triggered, the counter value is reset to zero, and the next stage becomes active. MWDTs/ RWDT provide four stages (called stages 0 to 3). The watchdog timers will progress through each stage in a loop (i.e. from stage 0 to 3, then back to stage 0).

Timeout values of each stage for MWDTs are configured in TIMG_WDTCONFIG*i*_REG (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured using RTC_CNTL_WDT_STG*j*_HOLD field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT (T_{hold0}) is determined together by EFUSE_WDT_DELAY_SEL field of an eFuse register and RTC_CNTL_WDT_STG0_HOLD. The relationship is as follows:

T_{hold0} = RTC_CNTL_WDT_STG0_HOLD « EFUSE_WDT_DELAY_SEL + 1

Upon the expiry of each stage, one of the following expiry actions will be executed:

- Trigger an interrupt
  When the stage expires, an interrupt is triggered.
- CPU reset
  When the stage expires, the CPU core will be reset.
- Core reset
  When the stage expires, the main system (which includes MWDTs, CPU, and all peripherals) will be reset. The RTC will not be reset.
- System reset
  When the stage expires the main system and the RTC will both be reset. This action is only available in RWDT.
- Disabled
  This stage will have no effects on the system.

For MWDTs, the expiry action of all stages is configured in TIMG_WDTCONFIG0_REG. Likewise for RWDT, the expiry action is configured in RTC_CNTL_WDTCONFIG0_REG.

12.3.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g. due to a misplaced register write). Therefore, MWDTs and RWDT incorporate a write protection mechanism that prevent the watchdogs from being disabled or tampered with due to an accidental write. The write protection mechanism is implemented using a write-key register for each timer (TIMG_WDT_WKEY for MWDT, RTC_CNTL_WDT_WKEY for RWDT). The value 0x50D83AA1 must be written to the watchdog timer's write-key register before any other register of the same watchdog timer can be changed. Any attempts to write to a watchdog timer's registers (other than the write-key register itself) whilst the write-key register's value is not 0x50D83AA1 will be ignored. The recommended procedure for accessing a watchdog timer is as follows:

1. Disable the write protection by writing the value 0x50D83AA1 to the timer's write-key register.
2. Make the required modification of the watchdog such as feeding or changing its configuration.
3. Re-enable write protection by writing any value other than 0x50D83AA1 to the timer's write-key register.

12.3.4 Flash Boot Protection

In flash boot mode, MWDT in timer group 0 (TIMG0), as well as RWDT, are enabled by default. Stage 0 for the enabled MWDT is automatically configured to reset the system upon expiry. Likewise, stage 0 for RWDT is configured to reset the main system and RTC when it expires. After booting, TIMG_WDT_FLASHBOOT_MOD_EN and RTC_CNTL_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for both MWDT and RWDT respectively. After this, MWDT and RWDT can be configured by software.

12.4 Super Watchdog

Super watchdog (SWD) is an ultra-low-power circuit that helps to prevent the system from operating in a sub-optimal state and resets the system if required. SWD contains a watchdog circuit that needs to be fed for nearly every 1 s. About 100 ms before watchdog timeout, it will also send out a WD_INTR signal as a request to remind the system to feed the watchdog.

If the system doesn't respond to SWD feed request and watchdog finally times out, SWD will generate a system level signal SWD_RSTB to reset the whole digital circuits on chip.

12.4.1 Features

SWD has the following features:

- Ultra-low power and small in area
- Interrupt to remind the system to feed SWD
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

12.4.2 Super Watchdog Controller

12.4.2.1 Structure

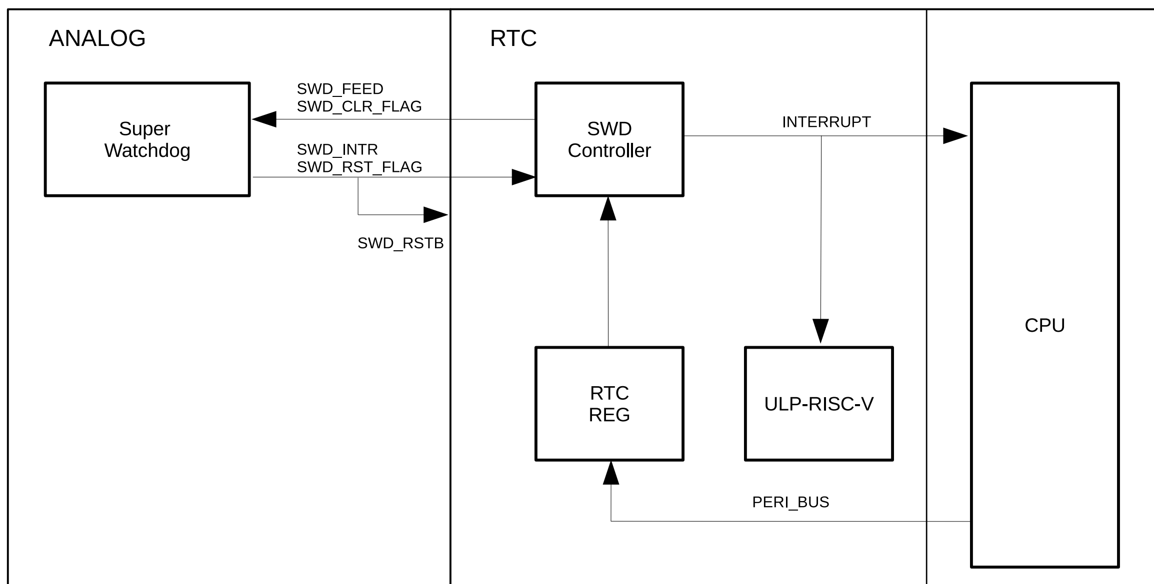


Figure 12-1. Super Watchdog Controller Structure

12.4.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU or ULP-RISC-V.
- Main CPU can decide whether to feed SWD directly by setting RTC_CNTL_SWD_FEED, or send an interrupt to ULP-RISC-V and ask ULP-RISC-V to feed SWD by setting RTC_CNTL_SWD_FEED.
- When trying to feed SWD, CPU or the co-processor needs to disable SWD controller's write protection by writing 0x8F1D312A to RTC_CNTL_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_CNTL_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU or ULP-RISC-V.

After reset:

- Check RTC_CNTL_RESET_CAUSE_PROCPU[5:0] for the cause of CPU reset.
  If RTC_CNTL_RESET_CAUSE_PROCPU[5:0] == 0x12, it indicates that the cause is SWD reset.
- Set RTC_CNTL_SWD_RST_FLAG_CLR to clear the SWD reset flag.

12.5 Registers

MWDT registers are part of the timer submodule and are described in the Chapter 11: *Timer Group (TIMG)* Timer Registers section. RWDT and SWD registers are part of the RTC submodule and are described in Chapter 9: *Low-Power Management (RTC_CNTL)* RTC Registers section.

13. XTAL32K Watchdog Timer (XTWDT)

13.1 Overview

The XTAL32K watchdog timer on ESP32-S2 is used to monitor the status of external crystal XTAL32K_CLK. This Watchdog can detect the oscillation failure of XTAL32K_CLK, change the clock source of RTC, etc. When XTAL32K_CLK works as the clock source of RTC SLOW_CLK and stops vibrating, the XTAL32K watchdog timer first switches to BACKUP32K_CLK derived from RTC_CLK and generates an interrupt (if the chip is in Deep-sleep mode, the CPU will be woken up), and then switches back to XTAL32K_CLK after it is restarted by software.

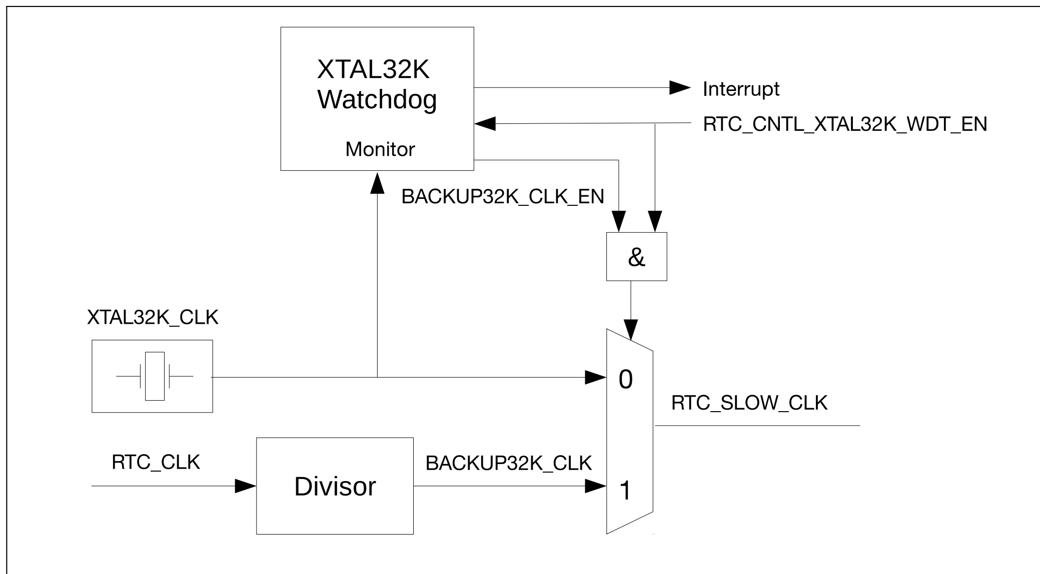


Figure 13-1. XTAL32K Watchdog Timer

13.2 Features

13.2.1 XTAL32K Watchdog Timer Interrupts and Wake-up

When the XTAL32K watchdog timer detects the oscillation failure of XTAL32K_CLK, an oscillation failure interrupt is generated. At this point the CPU will be woken up if in hibernation.

13.2.2 BACKUP32K_CLK

Once the XTAL32K watchdog timer detects the oscillation failure of XTAL32K_CLK, it replaces XTAL32K_CLK with BACKUP32K_CLK (with a frequency of 32 kHz) derived from RTC_CLK as RTC's SLOW_CLK, so as to ensure proper functioning of the system.

13.3 Functional Description

13.3.1 Workflow

1. The XTAL32K watchdog timer starts counting when RTC_CNTL_XTAL32K_WDT_EN is enabled. The counter keeps counting until it detects the positive edge of XTAL_32K and is then cleared. When the counter reaches RTC_CNTL_XTAL32K_WDT_TIMEOUT, it generates an interrupt or a wake-up signal and is then reset.

2. The XTAL32K watchdog timer automatically enables BACKUP32K_CLK as the alternative clock source of

RTC SLOW_CLK, to ensure the system's proper functioning and the accuracy of timers running on RTC SLOW_CLK (e.g. RTC_TIMER). For information about clock frequency configuration, please refer to Section 13.3.2 *Configuring the Divisor of BACKUP32K_CLK*.

3. To restore the XTAL32K watchdog timer, software restarts XTAL32K_CLK by turning its XPD signal on and on again via RTC_CNTL_XPD_XTAL_32K bit. Then, the XTAL32K watchdog timer switches back to XTAL32K_CLK by setting RTC_CNTL_XTAL32K_WDT_EN bit to 0. If the chip is in Deep-sleep mode, the XTAL32K watchdog timer will wake up the CPU to finish the above steps.

13.3.2 Configuring the Divisor of BACKUP32K_CLK

Chips have different RTC_CLK frequencies due to production process variations. To ensure the accuracy of RTC_TIMER and other timers running on SLOW_CLK when BACKUP32K_CLK is at work, the divisor of BACKUP32K_CLK should be configured according to the actual frequency of RTC_CLK. For details about the actual frequency of RTC_CLK, please refer to 9 *Low-Power Management (RTC_CNTL)*.

Define the frequency of RTC_CLK as f_rtc_clk (unit: kHz), and the eight divisor components as $x_0, x_1, x_2, x_3, x_4, x_5, x_6$, and x_7, respectively. $S = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$.

The following conditions should be fulfilled:

$$S = f_rtc_clk \times (4/32)$$
$$M + 1 \geq x_n \geq M (0 \leq n \leq 7)$$
$$M = f_rtc_clk/32/2$$

x_n should be an integer. M and S are rounded up or down. Each divisor component (x_0 ~x_7) is 4-bit long, and corresponds to the value of RTC_CNTL_XTAL32K_CLK_FACTOR (32-bit) in order.

For example, if the frequency of RTC_CLK is 163 kHz, then $f_rtc_clk = 163, S = 20, M = 2$, and $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} = \{2, 3, 2, 3, 2, 3, 2, 3\}$. As a result, the frequency of BACKUP32K_CLK is 32.6 kHz.

14. Permission Control (PMS)

14.1 Overview

In ESP32-S2, different type of buses and modules can have different permissions to access the memory. The permission control module is used to manage all the access permissions through corresponding permission control registers.

14.2 Features

- Permission controls for the CPU buses to access the internal memory
  - SRAM partially adopts unified permission control and partially adopts split permission control.
  - RTC FAST SRAM adopts split permission control.
  - RTC SLOW SRAM adopts split permission control.
- Access to the external memory is managed by both the memory management unit (MMU) and the permission control module.

14.3 Functional Description

14.3.1 Internal Memory Permission Controls

The internal memory resources in ESP32-S2 include ROM, SRAM, RTC FAST Memory, and RTC SLOW Memory. The breakdown of each memory is as follows:

- The total ROM size is 128 KB and consists of two 64 KB blocks.
- The total SRAM size is 320 KB and consists of four 8 KB blocks and eighteen 16 KB blocks, numbered Block 0 ~ 21. The offset address range of each block is shown in Table 71. For the base addresses accessed by different buses please refer to Section 14.3.1.1, 14.3.1.2, and 14.3.1.3. Each of the four 8 KB blocks (Block 0 ~ 3) is equipped with an independent permission control register. The eighteen 16 KB blocks (Block 4 ~ 21) are regarded as a large storage space as a whole, which is managed in split parts (i.e., the storage space is split into high and low areas with separate permission controls). The split address cannot fall within the address range of Block 4 ~ 5.
- RTC FAST Memory adopts split permission control. The high and low areas have independent permission control registers.
- RTC SLOW Memory also adopts split permission control. The high and low areas have independent permission control registers.

The following sections describe the buses and modules in ESP32-S2 that can access the above-mentioned internal memory resources.

Table 71: Offset Address Range of Each SRAM Block

SRAM Block	Offset Start Address	Offset End Address
Block 0	0x0000	0x1FFF
Block 1	0x2000	0x3FFF
Block 2	0x4000	0x5FFF
Block 3	0x6000	0x7FFF

SRAM Block	Offset Start Address	Offset End Address
Block 4	0x8000	0xBFFF
Block 5	0xC000	0xFFFF
Block 6	0x10000	0x13FFF
Block 7	0x14000	0x17FFF
Block 8	0x18000	0x1BFFF
Block 9	0x1C000	0x1FFFF
Block 10	0x20000	0x23FFF
Block 11	0x24000	0x27FFF
Block 12	0x28000	0x2BFFF
Block 13	0x2C000	0x2FFFF
Block 14	0x30000	0x33FFF
Block 15	0x34000	0x37FFF
Block 16	0x38000	0x3BFFF
Block 17	0x3C000	0x3FFFF
Block 18	0x40000	0x43FFF
Block 19	0x44000	0x47FFF
Block 20	0x48000	0x4BFFF
Block 21	0x4C000	0x4FFFF

14.3.1.1 Permission Control for the Instruction Bus (IBUS)

The address range on IBUS that requires permission control is 0x4002_0000 ~ 0x4007_1FFF, where SRAM memory blocks and RTC FAST Memory are located. Access to different memory blocks is controlled by independent permission control registers which are configured by software.

The permission control registers related to IBUS are also controlled by the PMS_PRO_IRAM0_LOCK signal. When this signal is set to 1, the values configured in the permission control registers will be locked and cannot be changed. At the same time, the value of the PMS_PRO_IRAM0_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_IRAM0_LOCK signal is reset to 0 only when the CPU is reset.

SRAM Memory Blocks

Through IBUS, the CPU can Read (data) from SRAM, Write (data) to SRAM, and eXecute (an instruction), indicated as R/W/X, respectively. The base address of SRAM accessed by the CPU is 0x4002_0000. Software can configure valid access types to each SRAM block in advance. The configuration of the permission control registers is shown in Table 72.

Table 72: Permission Control for IBUS to Access SRAM

Register	Bit Positions	Description
PMS_PRO_IRAM0_1_REG	[11:9]	Configure the permission for IBUS to access SRAM Block 3 (from high to low as W/R/X)
	[8:6]	Configure the permission for IBUS to access SRAM Block 2 (from high to low as W/R/X)
	[5:3]	Configure the permission for IBUS to access SRAM Block 1 (from high to low as W/R/X)

Register	Bit Positions	Description
	[2:0]	Configure the permission for IBUS to access SRAM Block 0 (from high to low as W/R/X)
PMS_PRO_IRAM0_2_REG	[22:20]	Configure the permission for IBUS to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R/X)
	[19:17]	Configure the permission for IBUS to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R/X)
	[16:0]*	Configure the split address of IBUS in the SRAM Block 4 ~ 21 address range. The base address is 0x4000_0000. See below for instructions on how to configure split address.

Note:
* **Configuring split address:**

Split address is 32 bits in width and equal to base address plus the configured field value multiplied by 4. For example, if 0x10000 is written to PMS_PRO_IRAM0_2_REG [16:0], the split address of IBUS in the SRAM Block 4 ~ 21 address range is equal to 0x4004_0000. Note that the split address on IBUS and DBUS0 cannot fall within the address range of Block 0 ~ 5.

If the type of access initiated by the CPU to SRAM through IBUS does not match the configured access type, the permission control module will directly deny this access. For example, if software allows the IBUS bus to only read and write SRAM Block5, but the CPU initiates an instruction fetch, the permission control module will deny this fetch access.

An access denial does not block the access initiated by the CPU, which means:

- The write operation will not take effect and will not change the data in the memory block.
- Read and fetch operations will get invalid data.

If the interrupt that indicates an illegitimate access to SRAM via IBUS is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

RTC FAST Memory

The CPU can perform R/W/X access to the RTC FAST Memory through IBUS. As RTC FAST Memory adopts split permission control, software needs to configure access permissions for the high and low address ranges, respectively. The configuration of the permission control registers is shown in Table 73.

Table 73: Permission Control for IBUS to Access RTC FAST Memory

Register	Bit Positions	Description
PMS_PRO_IRAM0_3_REG	[16:14]	Configure the permission for IBUS to access the high address range in RTC FAST Memory (from high to low as W/R/X)
	[13:11]	Configure the permission for IBUS to access the low address range in RTC FAST Memory (from high to low as W/R/X)

Register	Bit Positions	Description
	[10:0]	Configure the user-configurable split address of IBUS in the RTC FAST address range. The base address is 0x4007_0000. For how to configure the user-configurable split address, please refer to the instructions above.

If the type of access initiated by the CPU to RTC Fast Memory through IBUS does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to the RTC Fast Memory via IBUS is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

14.3.1.2 Permission Control for the Data Bus (DBUS0)

The CPU can access two address ranges through DBUS0: 0x3FFB_0000 ~ 0x3FFF_FFFF and 0x3FF9_E000 ~ 0x3FF9_FFFF, where SRAM memory blocks and RTC FAST Memory are located. Access to different memory blocks is controlled by independent permission control registers which are configured by software.

The permission control registers related to DBUS0 are also controlled by the PMS_PRO_DRAM0_LOCK signal. When this signal is set to 1, the configured values of the permission control registers will be locked and cannot be changed. The value of the PMS_PRO_DRAM0_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_DRAM0_LOCK signal is reset to 0 only when the CPU is reset.

SRAM Memory Blocks

The CPU can perform R/W access to SRAM via DBUS0, with the based address 0x3FFB_0000. Software can configure allowed access types initiated by DBUS0 to each SRAM block in advance. The configuration of the permission control registers is shown in Table 74.

Table 74: Permission Control for DBUS0 to Access SRAM

Register	Bit Positions	Description
PMS_PRO_DRAM0_1_REG	[28:27]	Configure the permission for DBUS0 to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[26:25]	Configure the permission for DBUS0 to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[24:8]	Configure the user-configurable split address of DBUS0 in the SRAM Block 4 ~ 21 address range. The base address is 0x3FFB_0000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.
	[7:6]	Configure the permission for DBUS0 to access SRAM Block3 (from high to low as W/R)
	[5:4]	Configure the permission for DBUS0 to access SRAM Block2 (from high to low as W/R)
	[3:2]	Configure the permission for DBUS0 to access SRAM Block1 (from high to low as W/R)

Register	Bit Positions	Description
	[1:0]	Configure the permission for DBUS0 to access SRAM Block0 (from high to low as W/R)

If the type of access initiated by the CPU to SRAM through DBUS0 does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to SRAM via DBUS0 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

RTC FAST Memory

The CPU can perform R/W access to RTC FAST Memory through DBUS0. As RTC FAST Memory adopts split permission control, software needs to configure access permissions for the high and low address ranges, respectively. The configuration of the permission control registers is shown in Table 75.

Table 75: Permission Control for DBUS0 to Access RTC FAST Memory

Register	Bit Positions	Description
PMS_PRO_DRAM0_2_REG	[14:13]	Configure the permission for DBUS0 to access the high address range in RTC FAST Memory (from high to low as W/R)
	[12:11]	Configure the permission for DBUS0 to access the low address range in RTC FAST Memory (from high to low as W/R)
	[10:0]	Configure the user-configurable split address of DBUS0 within the RTC FAST Memory address range. The base address is 0x3FF9_E000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.

If the type of access initiated by the CPU to RTC FAST Memory through DBUS0 does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to RTC FAST Memory via DBUS0 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

14.3.1.3 Permission Control for On-chip DMA

The on-chip DMA can access the internal memory through Internal DMA, Copy DMA RX (receiving) channel, or Copy DMA TX (transmitting) channel, using the address range 0x3FFB_0000 ~ 0x3FFF_FFFF (0x3FFB_0000 is the base address). Software can configure the type of DMA access to each SRAM block in advance. The configuration of the permission control registers is shown in Table 76, where XX can be APB, TX, or RX, corresponding to the configuration registers of Internal DMA, TX Copy DMA, and RX Copy DMA, respectively. More details about DMA can be found in Chapter 2 *DMA Controller (DMA)*.

The permission control registers related to DMA are also controlled by the PMS_DMA_XX_I_LOCK signal. When this signal is set to 1, the configured values of the permission control registers will be locked and cannot be changed. The value of the PMS_DMA_XX_I_LOCK signal will also remain at 1 and cannot be changed. The PMS_DMA_XX_I_LOCK signal value is reset to 0 only when the CPU is reset.

Table 76: Permission Control for On-chip DMA to Access SRAM

Register	Bit Positions	Description
PMS_DMA_XX_I_1_REG	[28:27]	Configure the permission for DMA to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[26:25]	Configure the permission for DMA to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[24:8]	Configure the user-configurable split address of DMA in SRAM Block 4 ~ 21 address range. The base address is 0x3FFB_0000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.
	[7:6]	Configure the permission for DMA to access SRAM Block3 (from high to low as W/R)
	[5:4]	Configure the permission for DMA to access SRAM Block2 (from high to low as W/R)
	[3:2]	Configure the permission for DMA to access SRAM Block1 (from high to low as W/R)
	[1:0]	Configure the permission for DMA to access SRAM Block0 (from high to low as W/R)

If the type of access initiated by the on-chip DMA to SRAM does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to SRAM via Internal DMA, TX Copy DMA, or RX Copy DMA is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

14.3.1.4 Permission Control for PeriBus1

The CPU can access the address range 0x3F40_0000 ~ 0x3F4F_FFFF through PeriBus1, where RTC SLOW SRAM is located. PeriBus1 can perform R/W operations. The read/write permission is controlled by software.

Software can also configure whether PeriBus1 is allowed to access peripherals that are within the address range 0x3F40_0000 ~ 0x3F4B_FFFF.

As the CPU can initiate predictive read operations through PeriBus1, it may cause peripheral FIFO read errors. To avoid FIFO read errors, the FIFO addresses of the corresponding peripherals have already been written in hardware and can no longer be changed, as Table 77 shows. In addition, four software-configurable address registers PMS_PRO_DPORT_2 ~ 5_REG are reserved for users to write into addresses read-protected from PeriBus1 reading. The configuration of the permission control registers is shown in Table 78.

Table 77: Peripherals and FIFO Address

Peripherals	FIFO Address
ADDR_RTCSLOW	0x6002_1000
ADDR_FIFO_UART0	0x6000_0000
ADDR_FIFO_UART1	0x6001_0000
ADDR_FIFO_UART2	0x6002_E000
ADDR_FIFO_I2S0	0x6000_F004
ADDR_FIFO_I2S1	0x6002_D004
ADDR_FIFO_RMT_CH0	0x6001_6000
ADDR_FIFO_RMT_CH1	0x6001_6004
ADDR_FIFO_RMT_CH2	0x6001_6008
ADDR_FIFO_RMT_CH3	0x6001_600C
ADDR_FIFO_I2C_EXT0	0x6001_301C
ADDR_FIFO_I2C_EXT1	0x6002_701C
ADDR_FIFO_USB_0	0x6008_0020
ADDR_FIFO_USB_1_L	0x6008_1000
ADDR_FIFO_USB_1_H	0x6009_0FFF

The permission control registers related to PeriBus1 are also controlled by the PMS_PRO_DPORT_LOCK signal. When this signal is set to 1, the configured values of the permission registers will be locked and cannot be changed. The value of the PMS_PRO_DPORT_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_DPORT_LOCK signal is reset to 0 only when the CPU is reset.

Table 78: Permission Control for PeriBus1

Register	Bit Positions	Description
PMS_PRO_DPORT_1_REG	[19:16]	The value of each bit determines whether to enable the corresponding reserved FIFO address.
	[15:14]	Configure the permission for PeriBus1 to access the high address range in RTC SLOW Memory (from high to low as W/R)
	[13:12]	Configure the permission for PeriBus1 to access the low address range in RTC SLOW Memory (from high to low as W/R)
	[11:1]	Configure the user-configurable split address of PeriBus1 in the RTC SLOW Memory address range. The base address is 0x3F42_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.
	[0]	Configure whether PeriBus1 can access the peripherals within the address range 0x3F40_0000 ~ 0x3F4B_FFFF
PMS_PRO_DPORT_2_REG	[17:0]	The zeroth address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [16]
PMS_PRO_DPORT_3_REG	[17:0]	The first address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [17]
PMS_PRO_DPORT_4_REG	[17:0]	The second address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [18]
PMS_PRO_DPORT_5_REG	[17:0]	The third address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [19]

If PeriBus1 initiates a read access to unreadable address, or if the type of the access to RTC SLOW Memory does not match the configured type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to RTC SLOW Memory by PeriBus1 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

14.3.1.5 Permission Control for PeriBus2

The two address ranges on PeriBus2 that require permission control are 0x5000_0000 ~ 0x5000_1FFF and 0x6000_0000 ~ 0x600B_FFFF where RTC SLOW Memory and peripheral modules are located.

PeriBus2 supports R/W/X access to RTC SLOW Memory through two address ranges, namely RTCSlow_0 and RTCSlow_1. Software can configure the allowed access types. The configuration of the permission control registers is shown in Table 79.

PeriBus2 does not support fetch operations on peripherals. If a fetch operation is initiated on a peripheral, invalid data will be obtained.

The permission control registers related to PeriBus2 are also controlled by the PMS_PRO_AHB_LOCK signal. When this signal is configured to 1, the values of the permission control registers are locked and cannot be changed. The value of the PMS_PRO_AHB_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_AHB_LOCK signal is reset to 0 only when the CPU is reset.

Table 79: Permission Control for PeriBus2 to Access RTC SLOW Memory

Register	Bit Positions	Description
PMS_PRO_AHB_1_REG	[16:14]	Configure the permission for PeriBus2 to access the high address range in RTCSlow_0 (from high to low as W/R/X)
	[13:11]	Configure the permission for PeriBus2 to access the low address range in RTCSlow_0 (from high to low as W/R/X)
	[10:0]	Configure the user-configurable split address of PeriBus2 in the RTCSlow_0 address range. The base address is 0x5000_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.
PMS_PRO_AHB_2_REG	[16:14]	Configure the permission for PeriBus2 to access the high address range in RTCSlow_1 (from high to low as W/R/X)
	[13:11]	Configure the permission for PeriBus2 to access the low address range in RTCSlow_1 (from high to low as W/R/X)
	[10:0]	Configure the user-configurable split address of PeriBus2 in the RTCSlow_1 address range. The base address is 0x6002_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 14.3.1.1.

If the type of the access to RTCSlow_0 or RTCSlow_1 made by the CPU through PeriBus2 does not match the configured type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access by PeriBus2 is enabled, the permission control module will record the current access address

and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

14.3.1.6 Permission Control for Cache

Among the SRAM blocks, only Block 0 ~ 3 can be assigned to Icache and Dcache. Icache or Dcache can access up to 16 KB internal memory, which means, they can access up to two blocks. The 16 KB address range accessed by Icache or Dcache is divided into two ranges: high address range (indicated by "_H" and low address range (indicated by "_L").

The user can allocate Block 0 ~ 3 to Icache_H, Icache_L, Dcache_H, or Dcache_L by configuring the field PMS_PRO_CACHE_CONNECT in register PMS_PRO_CACHE_1_REG. The detailed configuration is shown in Table 80, where *FIELD* indicates field PMS_PRO_CACHE_CONNECT.

Table 80: Configuration of Register PMS_PRO_CACHE_1_REG

SRAM Block 0-3	Dcache_H	Dcache_L	Icache_H	Icache_L
Block 0	*FIELD*[3]	*FIELD*[2]	*FIELD*[1]	*FIELD*[0]
Block 1	*FIELD*[7]	*FIELD*[6]	*FIELD*[5]	*FIELD*[4]
Block 2	*FIELD*[11]	*FIELD*[10]	*FIELD*[9]	*FIELD*[8]
Block 3	*FIELD*[15]	*FIELD*[14]	*FIELD*[13]	*FIELD*[12]

Notes:

- One block can be assigned to only one of Dcache_H, Dcache_L, Icache_H, and Icache_L, which means that one bit at most can be set to 1 in the same row in Table 80.
- One of Dcache_H, Dcache_L, Icache_H, and Icache_L can occupy only one block, which means that one bit at most can be set to 1 in the same column in Table 80.
- SRAM memory occupied by the cache cannot be accessed by the CPU. Non-occupied SRAM memory can still be accessed by the CPU.

14.3.1.7 Permission Control of Other Types of Internal Memory

In ESP32-S2, software can read Trace memory to get information about the runtime status of the CPU. Additionally, it can also be used as a debug channel to communicate with software running on the CPU. Software can enable the Trace memory function by configuring register PMS_PRO_TRACE_1. This register is also controlled by the PMS_PRO_TRACE_LOCK signal. When this signal is set to 1, the value configured in the register will be locked and cannot be changed. At the same time, the value of the PMS_PRO_TRACE_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_TRACE_LOCK signal is reset to 0 only when the CPU is reset.

After the Trace memory function is enabled, register PMS_OCCUPY_3 needs to be configured to select one block from SRAM Block 4 ~ 21 as Trace memory. Only one block can be used as Trace memory, in which case, it can no longer be accessed by the CPU. Register PMS_OCCUPY_3 is controlled by the PMS_OCCUPY_LOCK signal. When this signal is configured as 1, the value configured in the register will be locked and cannot be changed. At the same time, the value of the PMS_OCCUPY_LOCK signal will also remain at 1 and cannot be changed. The PMS_OCCUPY_LOCK signal value is reset to 0 the CPU is reset.

When an SRAM block is selected as Trace Memory, neither IBUS, DBUS, nor DMA can access this block.

14.3.2 External Memory Permission Control

The CPU can access the external flash and SRAM by SPI1, EDMA or cache. The former two support direct access, and the cache supports indirect access.

14.3.2.1 Cache MMU

The Cache MMU is used to control the permission for cache and EDMA to access the external memory. Its primary function is to convert virtual addresses to physical addresses. The MMU needs to be configured before enabling cache and EDMA. When a cache miss occurs or the data is written back to the external memory, the cache controller will automatically access the MMU and generate a physical address to access the external memory. When EDMA reads and writes the external SRAM, the EDMA controller also automatically accesses the MMU and generates a physical address to access the external SRAM.

Table 81: MMU Entries

MMU Entries	Bit Positions	Description
SRAM	[16]	The external memory attribute indicates whether cache/EDMA accesses the external flash or SRAM. If bit 15 is set, cache/EDMA accesses flash. If bit 16 is set, cache/EDMA accesses SRAM. The two bits cannot be set at the same time.
Flash	[15]	
Invalid	[14]	Cleared if MMU entry is valid.
Page number	[13:0]	For MMU purposes, external memory is divided in memory blocks called 'pages'. The page size is fixed at 64 KB. The page number of the physical address space indicates which page is accessed by cache/EDMA.

When cache or EDMA accesses an invalid MMU page or the external memory attribute is not specified in the MMU, an MMU error interrupt will be triggered.

14.3.2.2 External Memory Permission Controls

The hardware divides the physical address of the external memory (flash + SRAM) into eight (4 + 4) areas. Each area can be individually configured for W/R/X access. Software needs to configure the size of each area and its access type in advance.

The eight areas correspond to eight sets of registers, and the flash and SRAM each has four sets. Each set of registers contains three parts as follows:

1. Attribute list: APB_CTRL_X_ACE_n_ATTR_REG has 3 bits in total, which are W/R/X bits from high to low;
2. Area start address: APB_CTRL_X_ACE_n_ADDR_REG, which represents the physical address;
3. Area length: APB_CTRL_X_ACE_n_SIZE_REG in multiples of 64 KB;

Wherein "X" represents flash or SRAM, "n" represents 0 ~ 3. Note that the total size of the four areas corresponding to flash must be 1 GB, so is the total size of the four areas corresponding to SRAM.

When the CPU directly accesses external memory, the permission control module only monitors the CPU's write requests, not the read requests. The control module checks whether the access type matches the configured type according to the accessed physical address. If the area does not allow write access, the permission module directly rejects write access, records the information about the current access, and triggers an access error interrupt.

Cache miss, cache write back, and cache prefetch operations will trigger W/R/X requests from the cache to the external memory. The permission control module will query for the access attributes of the area according to the accessed physical address, and compare the the cache request against the pre-configured attributes. Only when the two match, the permission control module will pass the request to the external memory. If the cache initiates an R/F request, the permission control module will send the access attributes of the area to the cache for storage. When the access request is inconsistent with the current access attributes, the hardware records the information about the current access and triggers an access error interrupt.

When the CPU accesses the cache, the permission control module must check the attributes of the CPU access by comparing it with the local access attribute list. Only if the two match, the CPU access is allowed. When the access request is inconsistent with the current access attributes, the hardware records the information about the current access and triggers an access error interrupt.

If the access attribute check fails repeatedly, the hardware only records the first error message, and the subsequent error messages will be discarded directly. The access error interrupt is a level signal. When software clears the access error interrupt flag, the error record will be cleared at the same time.

14.3.3 Non-Aligned Access Permission Control

ESP32-S2 supports monitoring of non-word-aligned access. When PeriBus1 or PeriBus2 initiates a non-word address alignment or non-word sized access to a peripheral device, an interrupt or system exception may be triggered. The following table lists all possible access types and their corresponding results, where *IN* means interrupt, *EX* means exceptions, and $\surd$ means no action is triggered.

Table 82: Non-Aligned Access to Peripherals

Accessed by	Access Address	Access Unit	Read	Write
PeriBus2	0xXXXX_XXX0	byte	IN	IN
		halfword	IN	IN
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	√	IN
		word	√	IN
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	√	IN
Address range 0x3F40_0000 - 0x3F4B_FFFF	0xXXXX_XXX0	byte	IN	IN
		halfword	IN	IN
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	EX	EX
		word	EX	EX
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	EX	EX
Address range 0x3F4C_0000 - 0x3F4F_FFFF	0xXXXX_XXX0	byte	√	√
		halfword	√	√
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	EX	EX
		word	EX	EX
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	EX	EX

14.4 Base Address

Users can access the permission control module with the base address as shown in Table 83. For more information about accessing peripherals please see Chapter 3: *System and Memory*.

Table 83: Permission Control Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C1000

14.5 Register Summary

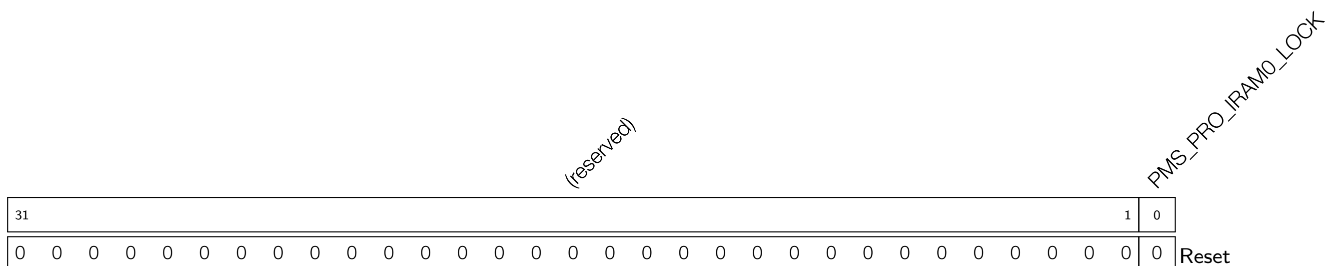
The address in the following table represents the address offset (relative address) with the respect to the peripheral base address, not the absolute address. For detailed information about the permission control base address, please refer to Section 14.4.

Name	Description	Address	Access
Control Registers			
PMS_PRO_IRAM0_0_REG	IBUS permission control register 0.	0x0010	R/W
PMS_PRO_DRAM0_0_REG	DBUS permission control register 0.	0x0028	R/W
PMS_PRO_DPORT_0_REG	PeriBus1 permission control register 0.	0x003C	R/W
PMS_PRO_AHB_0_REG	PeriBus2 permission control register 0.	0x005C	R/W
PMS_PRO_TRACE_0_REG	Trace memory permission control register 0.	0x0070	R/W
PMS_PRO_CACHE_0_REG	Cache permission control register 0.	0x0078	R/W
PMS_DMA_APB_I_0_REG	Internal DMA permission control register 0.	0x008C	R/W
PMS_DMA_RX_I_0_REG	RX Copy DMA permission control register 0.	0x009C	R/W
PMS_DMA_TX_I_0_REG	TX Copy DMA permission control register 0.	0x00AC	R/W
PMS_CACHE_SOURCE_0_REG	Cache access permission control register 0.	0x00C4	R/W
PMS_APB_PERIPHERAL_0_REG	Peripheral access permission control register 0.	0x00CC	R/W
PMS_OCCUPY_0_REG	Occupy permission control register 0.	0x00D4	R/W
PMS_CACHE_TAG_ACCESS_0_REG	Cache tag permission control register 0.	0x00E4	R/W
PMS_CACHE_MMU_ACCESS_0_REG	Cache MMU permission control register 0.	0x00EC	R/W
PMS_CLOCK_GATE_REG_REG	Clock gate register of permission control.	0x0104	R/W
Configuration Registers			
PMS_PRO_IRAM0_1_REG	IBUS permission control register 1.	0x0014	R/W
PMS_PRO_IRAM0_2_REG	IBUS permission control register 2.	0x0018	R/W
PMS_PRO_IRAM0_3_REG	IBUS permission control register 3.	0x001C	R/W
PMS_PRO_DRAM0_1_REG	DBUS permission control register 1.	0x002C	R/W
PMS_PRO_DRAM0_2_REG	DBUS permission control register 2.	0x0030	R/W
PMS_PRO_DPORT_1_REG	PeriBus1 permission control register 1.	0x0040	R/W
PMS_PRO_DPORT_2_REG	PeriBus1 permission control register 2.	0x0044	R/W
PMS_PRO_DPORT_3_REG	PeriBus1 permission control register 3.	0x0048	R/W
PMS_PRO_DPORT_4_REG	PeriBus1 permission control register 4.	0x004C	R/W
PMS_PRO_DPORT_5_REG	PeriBus1 permission control register 5.	0x0050	R/W
PMS_PRO_AHB_1_REG	PeriBus2 permission control register 1.	0x0060	R/W
PMS_PRO_AHB_2_REG	PeriBus2 permission control register 2.	0x0064	R/W
PMS_PRO_TRACE_1_REG	Trace memory permission control register 1.	0x0074	R/W
PMS_PRO_CACHE_1_REG	Cache permission control register 1.	0x007C	R/W
PMS_DMA_APB_I_1_REG	Internal DMA permission control register 1.	0x0090	R/W
PMS_DMA_RX_I_1_REG	RX Copy DMA permission control register 1.	0x00A0	R/W
PMS_DMA_TX_I_1_REG	TX Copy DMA permission control register 1.	0x00B0	R/W
PMS_APB_PERIPHERAL_1_REG	Peripheral access permission control register 1.	0x00D0	R/W
PMS_OCCUPY_1_REG	Occupy permission control register 1.	0x00D8	R/W
PMS_OCCUPY_3_REG	Occupy permission control register 3.	0x00E0	R/W
PMS_CACHE_TAG_ACCESS_1_REG	Cache tag permission control register 1.	0x00E8	R/W
PMS_CACHE_MMU_ACCESS_1_REG	Cache MMU permission control register 1.	0x00F0	R/W
Interrupt Registers			
PMS_PRO_IRAM0_4_REG	IBUS permission control register 4.	0x0020	varies
PMS_PRO_IRAM0_5_REG	IBUS status register.	0x0024	RO
PMS_PRO_DRAM0_3_REG	DBUS permission control register 3.	0x0034	varies

Name	Description	Address	Access
PMS_PRO_DRAM0_4_REG	DBUS status register.	0x0038	RO
PMS_PRO_DPORT_6_REG	PeriBus1 permission control register 6.	0x0054	varies
PMS_PRO_DPORT_7_REG	PeriBus1 status register.	0x0058	RO
PMS_PRO_AHB_3_REG	PeriBus2 permission control register 3.	0x0068	varies
PMS_PRO_AHB_4_REG	PeriBus2 status register.	0x006C	RO
PMS_PRO_CACHE_2_REG	Cache permission control register 2.	0x0080	varies
PMS_PRO_CACHE_3_REG	Icache status register.	0x0084	RO
PMS_PRO_CACHE_4_REG	Dcache status register.	0x0088	RO
PMS_DMA_APB_I_2_REG	Internal DMA permission control register 2.	0x0094	varies
PMS_DMA_APB_I_3_REG	Internal DMA status register.	0x0098	RO
PMS_DMA_RX_I_2_REG	RX Copy DMA permission control register 2.	0x00A4	varies
PMS_DMA_RX_I_3_REG	RX Copy DMA status register.	0x00A8	RO
PMS_DMA_TX_I_2_REG	TX Copy DMA permission control register 2.	0x00B4	varies
PMS_DMA_TX_I_3_REG	TX Copy DMA status register.	0x00B8	RO
PMS_APB_PERIPHERAL_INTR_REG	PeriBus2 permission control register.	0x00F4	varies
PMS_APB_PERIPHERAL_STATUS_REG	PeriBus2 peripheral access status register.	0x00F8	RO
PMS_CPU_PERIPHERAL_INTR_REG	PeriBus1 permission control register.	0x00FC	varies
PMS_CPU_PERIPHERAL_STATUS_REG	PeriBus1 peripheral access status register.	0x0100	RO
Version Control Register			
PMS_DATE	Version control register.	0x0FFC	R/W

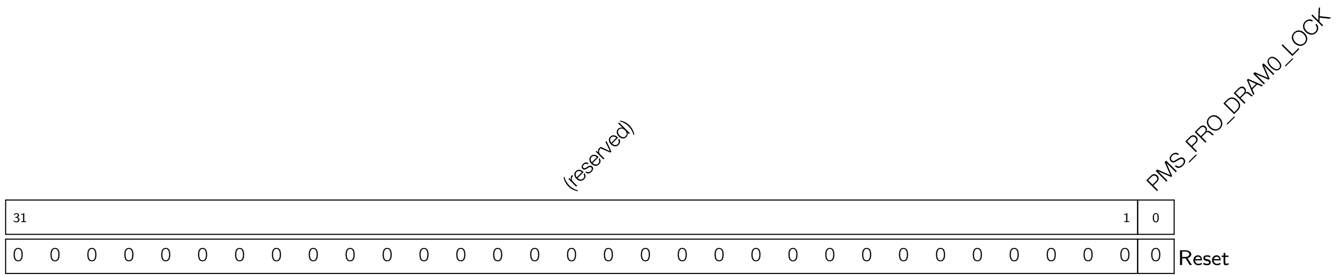
14.6 Registers

Register 14.1: PMS_PRO_IRAM0_0_REG (0x0010)



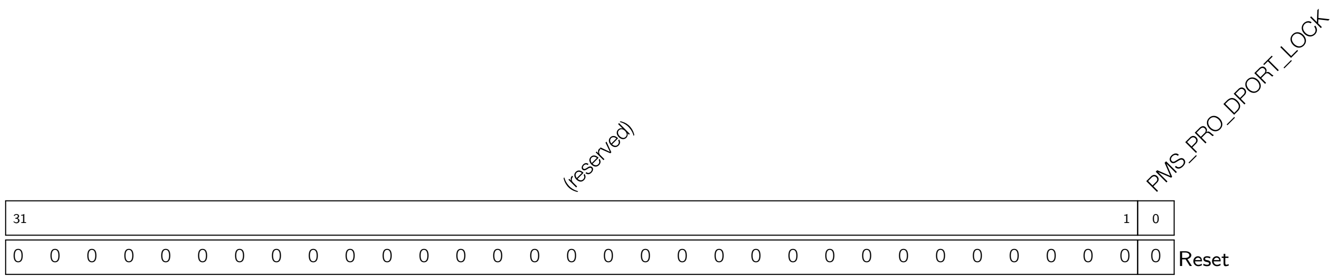
PMS_PRO_IRAM0_LOCK Lock register. Setting to 1 locks IBUS permission control registers. (R/W)

Register 14.2: PMS_PRO_DRAM0_0_REG (0x0028)



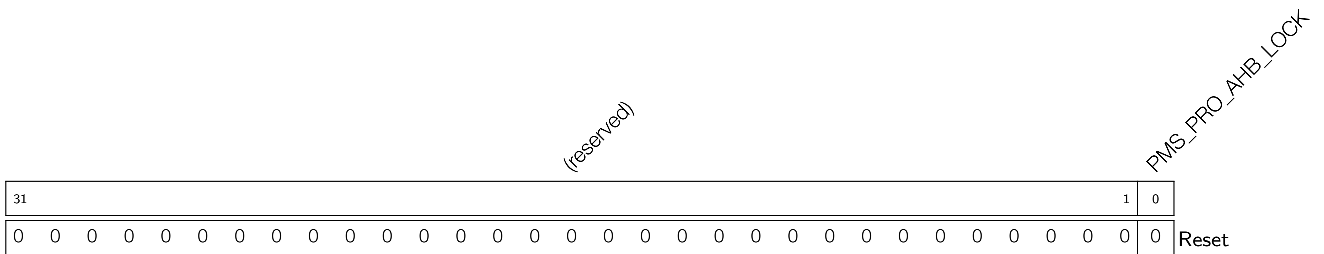
PMS_PRO_DRAM0_LOCK Lock register. Setting to 1 locks DBUS0 permission control registers. (R/W)

Register 14.3: PMS_PRO_DPORT_0_REG (0x003C)



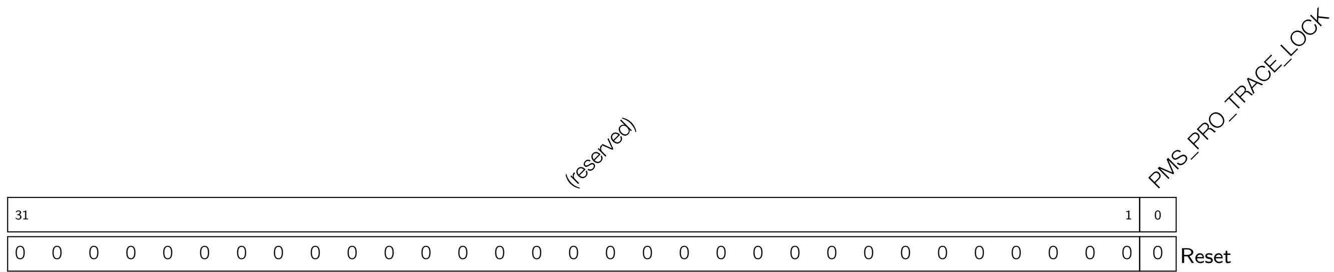
PMS_PRO_DPORT_LOCK Lock register. Setting to 1 locks PeriBus1 permission control registers. (R/W)

Register 14.4: PMS_PRO_AHB_0_REG (0x005C)



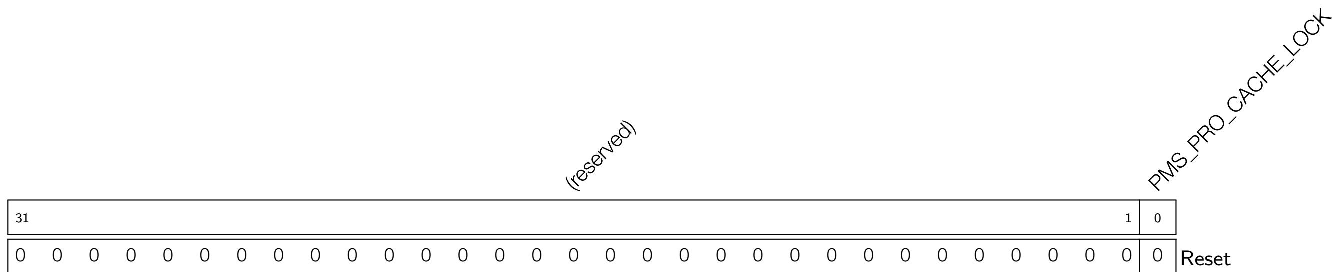
PMS_PRO_AHB_LOCK Lock register. Setting to 1 locks PeriBus2 permission control registers. (R/W)

Register 14.5: PMS_PRO_TRACE_0_REG (0x0070)



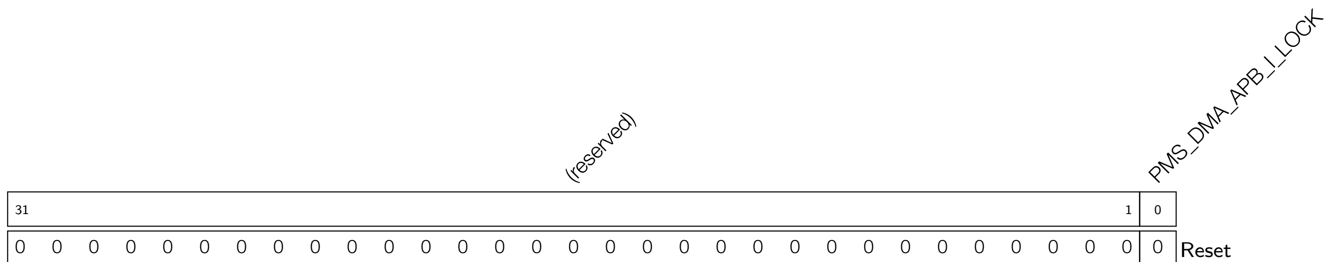
PMS_PRO_TRACE_LOCK Lock register. Setting to 1 locks trace function permission control registers. (R/W)

Register 14.6: PMS_PRO_CACHE_0_REG (0x0078)



PMS_PRO_CACHE_LOCK Lock register. Setting to 1 locks cache permission control registers. (R/W)

Register 14.7: PMS_DMA_APB_I_0_REG (0x008C)



PMS_DMA_APB_I_LOCK Lock register. Setting to 1 locks internal DMA permission control registers. (R/W)

Register 14.8: PMS_DMA_RX_I_0_REG (0x009C)

(reserved)	PMS_DMA_RX_I_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_DMA_RX_I_LOCK Lock register. Setting to 1 locks RX Copy DMA permission control registers. (R/W)

Register 14.9: PMS_DMA_TX_I_0_REG (0x00AC)

(reserved)	PMS_DMA_TX_I_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_DMA_TX_I_LOCK Lock register. Setting to 1 locks TX Copy DMA permission control registers. (R/W)

Register 14.10: PMS_CACHE_SOURCE_0_REG (0x00C4)

(reserved)	PMS_CACHE_SOURCE_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_CACHE_SOURCE_LOCK Lock register. Setting to 1 locks cache access permission control registers. (R/W)

Register 14.11: PMS_APB_PERIPHERAL_0_REG (0x00CC)

(reserved)	PMS_APB_PERIPHERAL_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_APB_PERIPHERAL_LOCK Lock register. Setting to 1 locks TX Copy DMA permission control registers. (R/W)

Register 14.12: PMS_OCCUPY_0_REG (0x00D4)

(reserved)	PMS_OCCUPY_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_OCCUPY_LOCK Lock register. Setting to 1 locks occupy permission control registers. (R/W)

Register 14.13: PMS_CACHE_TAG_ACCESS_0_REG (0x00E4)

(reserved)	PMS_CACHE_TAG_ACCESS_LOCK	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_CACHE_TAG_ACCESS_LOCK Lock register. Setting to 1 locks cache tag permission control registers. (R/W)

Register 14.14: PMS_CACHE_MMU_ACCESS_0_REG (0x00EC)

31–1 (reserved)	0 PMS_CACHE_MMU_ACCESS_LOCK	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_CACHE_MMU_ACCESS_LOCK Lock register. Setting to 1 locks cache MMU permission control registers. (R/W)

Register 14.15: PMS_CLOCK_GATE_REG_REG (0x0104)

31–1 (reserved)	0 PMS_CLK_EN	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

PMS_CLK_EN Enable the clock of permission control module when set to 1. (R/W)

Register 14.16: PMS_PRO_IRAM0_1_REG (0x0014)

(reserved)	PMS_PRO_IRAM0_SRAM_3_W	PMS_PRO_IRAM0_SRAM_3_R	PMS_PRO_IRAM0_SRAM_3_F	PMS_PRO_IRAM0_SRAM_2_W	PMS_PRO_IRAM0_SRAM_2_R	PMS_PRO_IRAM0_SRAM_2_F	PMS_PRO_IRAM0_SRAM_1_W	PMS_PRO_IRAM0_SRAM_1_R	PMS_PRO_IRAM0_SRAM_1_F	PMS_PRO_IRAM0_SRAM_0_W	PMS_PRO_IRAM0_SRAM_0_R	PMS_PRO_IRAM0_SRAM_0_F	
31 … 12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_PRO_IRAM0_SRAM_0_F Setting to 1 grants IBUS permission to fetch SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_0_R Setting to 1 grants IBUS permission to read SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_0_W Setting to 1 grants IBUS permission to write SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_1_F Setting to 1 grants IBUS permission to fetch SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_1_R Setting to 1 grants IBUS permission to read SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_1_W Setting to 1 grants IBUS permission to write SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_2_F Setting to 1 grants IBUS permission to fetch SRAM Block 2. (R/W)

PMS_PRO_IRAM0_SRAM_2_R Setting to 1 grants IBUS permission to read SRAM Block 2. (R/W)

PMS_PRO_IRAM0_SRAM_2_W Setting to 1 grants IBUS permission to write SRAM Block 2. (R/W)

PMS_PRO_IRAM0_SRAM_3_F Setting to 1 grants IBUS permission to fetch SRAM Block 3. (R/W)

PMS_PRO_IRAM0_SRAM_3_R Setting to 1 grants IBUS permission to read SRAM Block 3. (R/W)

PMS_PRO_IRAM0_SRAM_3_W Setting to 1 grants IBUS permission to write SRAM Block 3. (R/W)

Register 14.17: PMS_PRO_IRAM0_2_REG (0x0018)

31–23	22	21	20	19	18	17	16–0	
(reserved)	PMS_PRO_IRAM0_SRAM_4_H_W	PMS_PRO_IRAM0_SRAM_4_H_R	PMS_PRO_IRAM0_SRAM_4_H_F	PMS_PRO_IRAM0_SRAM_4_L_W	PMS_PRO_IRAM0_SRAM_4_L_R	PMS_PRO_IRAM0_SRAM_4_L_F	PMS_PRO_IRAM0_SRAM_4_SPLTADDR	
0 0 0 0 0 0 0 0 0	1	1	1	1	1	1	0	Reset

PMS_PRO_IRAM0_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for IBUS access. (R/W)

PMS_PRO_IRAM0_SRAM_4_L_F Setting to 1 grants IBUS permission to fetch SRAM Block 4-21 low address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_L_R Setting to 1 grants IBUS permission to read SRAM Block 4-21 low address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_L_W Setting to 1 grants IBUS permission to write SRAM Block 4-21 low address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_F Setting to 1 grants IBUS permission to fetch SRAM Block 4-21 high address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_R Setting to 1 grants IBUS permission to read SRAM Block 4-21 high address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_W Setting to 1 grants IBUS permission to write SRAM Block 4-21 high address region. (R/W)

Register 14.18: PMS_PRO_IRAM0_3_REG (0x001C)

(reserved)	PMS_PRO_IRAM0_RTCFAST_H_W	PMS_PRO_IRAM0_RTCFAST_H_R	PMS_PRO_IRAM0_RTCFAST_H_F	PMS_PRO_IRAM0_RTCFAST_L_W	PMS_PRO_IRAM0_RTCFAST_L_R	PMS_PRO_IRAM0_RTCFAST_L_F	PMS_PRO_IRAM0_RTCFAST_SPLTADDR	
31 ... 17	16	15	14	13	12	11	10 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	1	1	0	Reset

PMS_PRO_IRAM0_RTCFAST_SPLTADDR Configure the split address of RTC FAST for IBUS access. (R/W)

PMS_PRO_IRAM0_RTCFAST_L_F Setting to 1 grants IBUS permission to fetch RTC FAST low address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_L_R Setting to 1 grants IBUS permission to read RTC FAST low address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_L_W Setting to 1 grants IBUS permission to write RTC FAST low address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_F Setting to 1 grants IBUS permission to fetch RTC FAST high address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_R Setting to 1 grants IBUS permission to read RTC FAST high address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_W Setting to 1 grants IBUS permission to write RTC FAST high address region. (R/W)

Register 14.19: PMS_PRO_DRAM0_1_REG (0x002C)

31..29	28	27	26	25	24..8	7	6	5	4	3	2	1	0	
(reserved)	PMS_PRO_DRAM0_SRAM_4_H_W	PMS_PRO_DRAM0_SRAM_4_H_R	PMS_PRO_DRAM0_SRAM_4_L_W	PMS_PRO_DRAM0_SRAM_4_L_R	PMS_PRO_DRAM0_SRAM_4_SPLTADDR	PMS_PRO_DRAM0_SRAM_3_W	PMS_PRO_DRAM0_SRAM_3_R	PMS_PRO_DRAM0_SRAM_2_W	PMS_PRO_DRAM0_SRAM_2_R	PMS_PRO_DRAM0_SRAM_1_W	PMS_PRO_DRAM0_SRAM_1_R	PMS_PRO_DRAM0_SRAM_0_W	PMS_PRO_DRAM0_SRAM_0_R	
0 0 0	1	1	1	1	0	1	1	1	1	1	1	1	1	Reset

PMS_PRO_DRAM0_SRAM_0_R Setting to 1 grants DBUS0 permission to read SRAM Block 0. (R/W)

PMS_PRO_DRAM0_SRAM_0_W Setting to 1 grants DBUS0 permission to write SRAM Block 0. (R/W)

PMS_PRO_DRAM0_SRAM_1_R Setting to 1 grants DBUS0 permission to read SRAM Block 1. (R/W)

PMS_PRO_DRAM0_SRAM_1_W Setting to 1 grants DBUS0 permission to write SRAM Block 1. (R/W)

PMS_PRO_DRAM0_SRAM_2_R Setting to 1 grants DBUS0 permission to read SRAM Block 2. (R/W)

PMS_PRO_DRAM0_SRAM_2_W Setting to 1 grants DBUS0 permission to write SRAM Block 2. (R/W)

PMS_PRO_DRAM0_SRAM_3_R Setting to 1 grants DBUS0 permission to read SRAM Block 3. (R/W)

PMS_PRO_DRAM0_SRAM_3_W Setting to 1 grants DBUS0 permission to write SRAM Block 3. (R/W)

PMS_PRO_DRAM0_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for DBUS0 access. (R/W)

PMS_PRO_DRAM0_SRAM_4_L_R Setting to 1 grants DBUS0 permission to read SRAM Block 4-21 low address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_L_W Setting to 1 grants DBUS0 permission to write SRAM Block 4-21 low address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_H_R Setting to 1 grants DBUS0 permission to read SRAM Block 4-21 high address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_H_W Setting to 1 grants DBUS0 permission to write SRAM Block 4-21 high address region. (R/W)

Register 14.20: PMS_PRO_DRAM0_2_REG (0x0030)

31–15	14	13	12	11	10–0	
(reserved)	PMS_PRO_DRAM0_RTCFAST_H_W	PMS_PRO_DRAM0_RTCFAST_H_R	PMS_PRO_DRAM0_RTCFAST_L_W	PMS_PRO_DRAM0_RTCFAST_L_R	PMS_PRO_DRAM0_RTCFAST_SPLTADDR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	0	Reset

PMS_PRO_DRAM0_RTCFAST_SPLTADDR Configure the split address of RTC FAST for DBUS0 access. (R/W)

PMS_PRO_DRAM0_RTCFAST_L_R Setting to 1 grants DBUS0 permission to read RTC FAST low address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_L_W Setting to 1 grants DBUS0 permission to write RTC FAST low address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_H_R Setting to 1 grants DBUS0 permission to read RTC FAST high address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_H_W Setting to 1 grants DBUS0 permission to write RTC FAST high address region. (R/W)

Register 14.21: PMS_PRO_DPORT_1_REG (0x0040)

Bits	31–20	19–16	15	14	13	12	11–1	0
Field	(reserved)	PMS_PRO_DPORT_RESERVE_FIFO_VALID	PMS_PRO_DPORT_RTCSLOW_H_W	PMS_PRO_DPORT_RTCSLOW_H_R	PMS_PRO_DPORT_RTCSLOW_L_W	PMS_PRO_DPORT_RTCSLOW_L_R	PMS_PRO_DPORT_RTCSLOW_SPLTADDR	PMS_PRO_DPORT_APB_PERIPHERAL
Reset	0 0 0 0 0 0 0 0 0 0 0 0	0	1	1	1	1	0	0

PMS_PRO_DPORT_APB_PERIPHERAL_FORBID Setting to 1 denies PeriBus1 bus's access to APB peripheral. (R/W)

PMS_PRO_DPORT_RTCSLOW_SPLTADDR Configure the split address of RTC FAST for PeriBus1 access. (R/W)

PMS_PRO_DPORT_RTCSLOW_L_R Setting to 1 grants PeriBus1 permission to read RTC FAST low address region. (R/W)

PMS_PRO_DPORT_RTCSLOW_L_W Setting to 1 grants PeriBus1 permission to write RTC FAST low address region. (R/W)

PMS_PRO_DPORT_RTCSLOW_H_R Setting to 1 grants PeriBus1 permission to read RTC FAST high address region. (R/W)

PMS_PRO_DPORT_RTCSLOW_H_W Setting to 1 grants PeriBus1 permission to write RTC FAST high address region. (R/W)

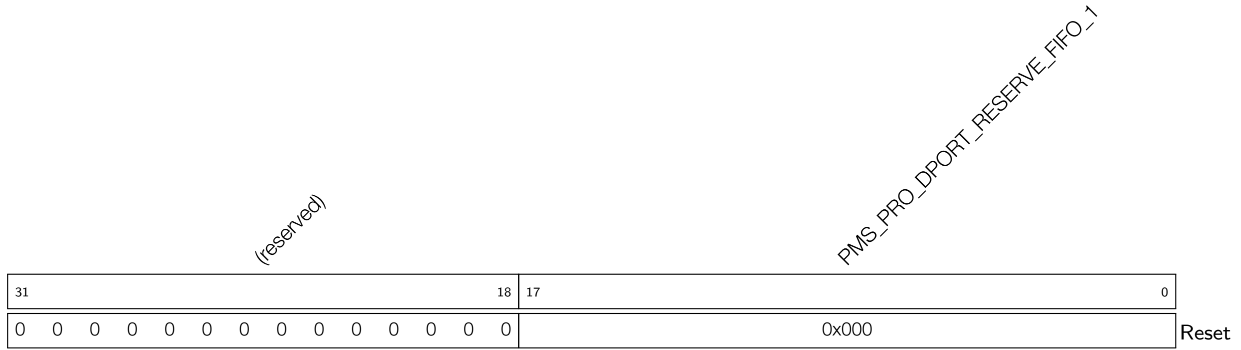
PMS_PRO_DPORT_RESERVE_FIFO_VALID Configure whether to enable read protection for user-configured FIFO address. (R/W)

Register 14.22: PMS_PRO_DPORT_2_REG (0x0044)

Bits	31–18	17–0
Field	(reserved)	PMS_PRO_DPORT_RESERVE_FIFO_0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x000

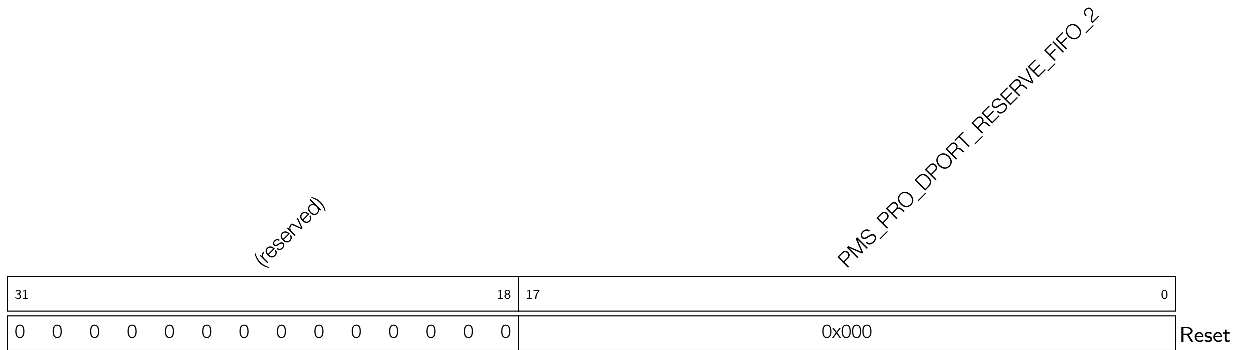
PMS_PRO_DPORT_RESERVE_FIFO_0 Configure read-protection address 0. (R/W)

Register 14.23: PMS_PRO_DPORT_3_REG (0x0048)



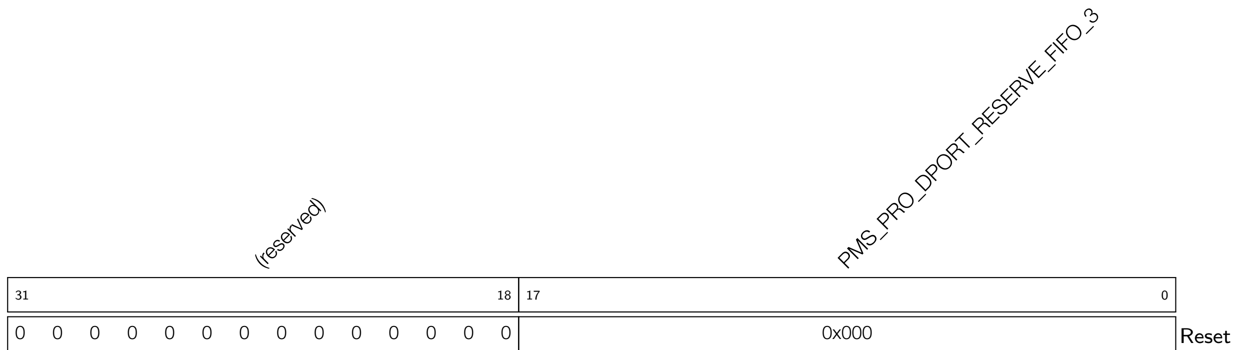
PMS_PRO_DPORT_RESERVE_FIFO_1 Configure read-protection address 1. (R/W)

Register 14.24: PMS_PRO_DPORT_4_REG (0x004C)



PMS_PRO_DPORT_RESERVE_FIFO_2 Configure read-protection address 2. (R/W)

Register 14.25: PMS_PRO_DPORT_5_REG (0x0050)



PMS_PRO_DPORT_RESERVE_FIFO_3 Configure read-protection address 3. (R/W)

Register 14.26: PMS_PRO_AHB_1_REG (0x0060)

(reserved)	PMS_PRO_AHB_RTCSLOW_0_H_W	PMS_PRO_AHB_RTCSLOW_0_H_R	PMS_PRO_AHB_RTCSLOW_0_H_F	PMS_PRO_AHB_RTCSLOW_0_L_W	PMS_PRO_AHB_RTCSLOW_0_L_R	PMS_PRO_AHB_RTCSLOW_0_L_F	PMS_PRO_AHB_RTCSLOW_0_SPLTADDR	
31 ... 17	16	15	14	13	12	11	10 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	1	1	0	Reset

PMS_PRO_AHB_RTCSLOW_0_SPLTADDR Configure the split address of RTCSlow_0 for PeriBus2 access. (R/W)

PMS_PRO_AHB_RTCSLOW_0_L_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_0 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_L_R Setting to 1 grants PeriBus2 permission to read RTCSlow_0 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_L_W Setting to 1 grants PeriBus2 permission to write RTCSlow_0 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_0 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_R Setting to 1 grants PeriBus2 permission to read RTCSlow_0 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_W Setting to 1 grants PeriBus2 permission to write RTCSlow_0 high address region. (R/W)

Register 14.27: PMS_PRO_AHB_2_REG (0x0064)

(reserved)	PMS_PRO_AHB_RTCSLOW_1_H_W	PMS_PRO_AHB_RTCSLOW_1_H_R	PMS_PRO_AHB_RTCSLOW_1_H_F	PMS_PRO_AHB_RTCSLOW_1_L_W	PMS_PRO_AHB_RTCSLOW_1_L_R	PMS_PRO_AHB_RTCSLOW_1_L_F	PMS_PRO_AHB_RTCSLOW_1_SPLTADDR	
31 ... 17	16	15	14	13	12	11	10 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	1	1	0	Reset

PMS_PRO_AHB_RTCSLOW_1_SPLTADDR Configure the split address of RTCSlow_1 for PeriBus2 access. (R/W)

PMS_PRO_AHB_RTCSLOW_1_L_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_1 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_L_R Setting to 1 grants PeriBus2 permission to read RTCSlow_1 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_L_W Setting to 1 grants PeriBus2 permission to write RTCSlow_1 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_H_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_1 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_H_R Setting to 1 grants PeriBus2 permission to read RTCSlow_1 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_H_W Setting to 1 grants PeriBus2 permission to write RTCSlow_1 high address region. (R/W)

Register 14.28: PMS_PRO_TRACE_1_REG (0x0074)

(reserved)	PMS_PRO_TRACE_DISABLE	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_PRO_TRACE_DISABLE Setting to 1 disables the trace memory function. (R/W)

Register 14.29: PMS_PRO_CACHE_1_REG (0x007C)

31 ... 16 (reserved)	15 ... 0 PMS_PRO_CACHE_CONNECT	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_PRO_CACHE_CONNECT Configure which SRAM Block will be occupied by Icache or Dcache. (R/W)

Register 14.30: PMS_DMA_APB_I_1_REG (0x0090)

31–29	28	27	26	25	24–8	7	6	5	4	3	2	1	0	
(reserved)	PMS_DMA_APB_I_SRAM_4_H_W	PMS_DMA_APB_I_SRAM_4_H_R	PMS_DMA_APB_I_SRAM_4_L_W	PMS_DMA_APB_I_SRAM_4_L_R	PMS_DMA_APB_I_SRAM_4_SPLTADDR	PMS_DMA_APB_I_SRAM_3_W	PMS_DMA_APB_I_SRAM_3_R	PMS_DMA_APB_I_SRAM_2_W	PMS_DMA_APB_I_SRAM_2_R	PMS_DMA_APB_I_SRAM_1_W	PMS_DMA_APB_I_SRAM_1_R	PMS_DMA_APB_I_SRAM_0_W	PMS_DMA_APB_I_SRAM_0_R	
0 0 0	1	1	1	1	0	1	1	1	1	1	1	1	1	Reset

PMS_DMA_APB_I_SRAM_0_R Setting to 1 grants internal DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_APB_I_SRAM_0_W Setting to 1 grants internal DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_APB_I_SRAM_1_R Setting to 1 grants internal DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_APB_I_SRAM_1_W Setting to 1 grants internal DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_APB_I_SRAM_2_R Setting to 1 grants internal DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_APB_I_SRAM_2_W Setting to 1 grants internal DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_APB_I_SRAM_3_R Setting to 1 grants internal DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_APB_I_SRAM_3_W Setting to 1 grants internal DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_APB_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for internal DMA access. (R/W)

PMS_DMA_APB_I_SRAM_4_L_R Setting to 1 grants internal DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_APB_I_SRAM_4_L_W Setting to 1 grants internal DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_APB_I_SRAM_4_H_R Setting to 1 grants internal DMA permission to read SRAM Block 4-21 high address region. (R/W)

PMS_DMA_APB_I_SRAM_4_H_W Setting to 1 grants internal DMA permission to write SRAM Block 4-21 high address region. (R/W)

Register 14.31: PMS_DMA_RX_I_1_REG (0x00A0)

31–29	28	27	26	25	24–8	7	6	5	4	3	2	1	0	
(reserved)	PMS_DMA_RX_I_SRAM_4_H_W	PMS_DMA_RX_I_SRAM_4_H_R	PMS_DMA_RX_I_SRAM_4_L_W	PMS_DMA_RX_I_SRAM_4_L_R	PMS_DMA_RX_I_SRAM_4_SPLTADDR	PMS_DMA_RX_I_SRAM_3_W	PMS_DMA_RX_I_SRAM_3_R	PMS_DMA_RX_I_SRAM_2_W	PMS_DMA_RX_I_SRAM_2_R	PMS_DMA_RX_I_SRAM_1_W	PMS_DMA_RX_I_SRAM_1_R	PMS_DMA_RX_I_SRAM_0_W	PMS_DMA_RX_I_SRAM_0_R	
0 0 0	1	1	1	1	0	1	1	1	1	1	1	1	1	Reset

PMS_DMA_RX_I_SRAM_0_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_RX_I_SRAM_0_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_RX_I_SRAM_1_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_RX_I_SRAM_1_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_RX_I_SRAM_2_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_RX_I_SRAM_2_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_RX_I_SRAM_3_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_RX_I_SRAM_3_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_RX_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for RX Copy DMA access. (R/W)

PMS_DMA_RX_I_SRAM_4_L_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_RX_I_SRAM_4_L_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_RX_I_SRAM_4_H_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 4-21 high address region. (R/W)

PMS_DMA_RX_I_SRAM_4_H_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 4 21 high address region. (R/W)

Register 14.32: PMS_DMA_TX_I_1_REG (0x00B0)

31–29	28	27	26	25	24–8	7	6	5	4	3	2	1	0	
(reserved)	PMS_DMA_TX_I_SRAM_4_H_W	PMS_DMA_TX_I_SRAM_4_H_R	PMS_DMA_TX_I_SRAM_4_L_W	PMS_DMA_TX_I_SRAM_4_L_R	PMS_DMA_TX_I_SRAM_4_SPLTADDR	PMS_DMA_TX_I_SRAM_3_W	PMS_DMA_TX_I_SRAM_3_R	PMS_DMA_TX_I_SRAM_2_W	PMS_DMA_TX_I_SRAM_2_R	PMS_DMA_TX_I_SRAM_1_W	PMS_DMA_TX_I_SRAM_1_R	PMS_DMA_TX_I_SRAM_0_W	PMS_DMA_TX_I_SRAM_0_R	
0 0 0	1	1	1	1	0	1	1	1	1	1	1	1	1	Reset

PMS_DMA_TX_I_SRAM_0_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_TX_I_SRAM_0_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_TX_I_SRAM_1_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_TX_I_SRAM_1_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_TX_I_SRAM_2_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_TX_I_SRAM_2_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_TX_I_SRAM_3_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_TX_I_SRAM_3_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_TX_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for TX Copy DMA access. (R/W)

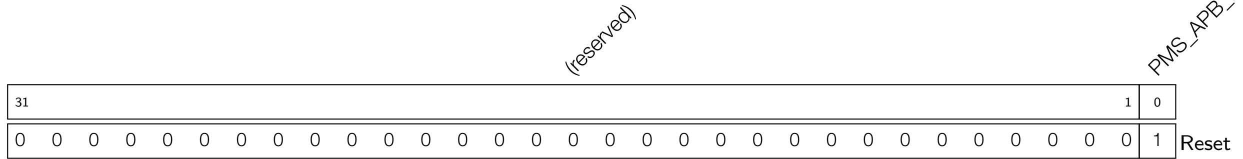
PMS_DMA_TX_I_SRAM_4_L_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_TX_I_SRAM_4_L_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_TX_I_SRAM_4_H_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 4-21 high address region. (R/W)

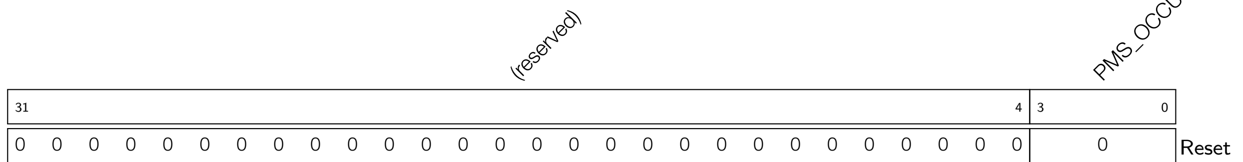
PMS_DMA_TX_I_SRAM_4_H_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 4-21 high address region. (R/W)

Register 14.33: PMS_APB_PERIPHERAL_1_REG (0x00D0)



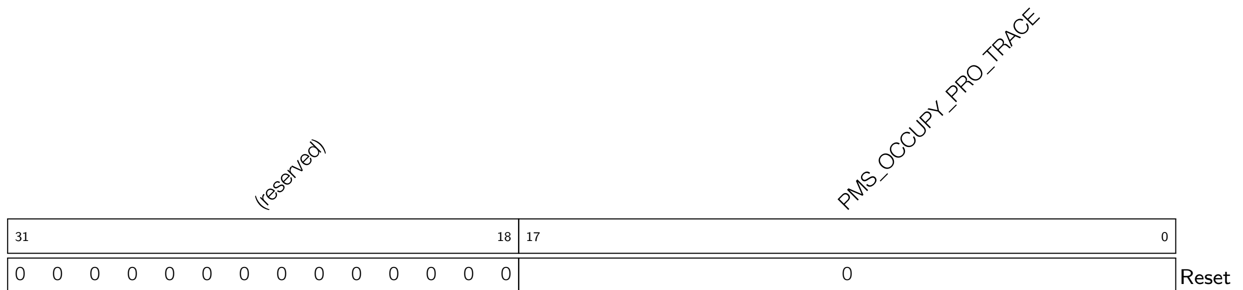
PMS_APB_PERIPHERAL_SPLIT_BURST Setting to 1 splits the data phase of the last access and the address phase of following access. (R/W)

Register 14.34: PMS_OCCUPY_1_REG (0x00D8)



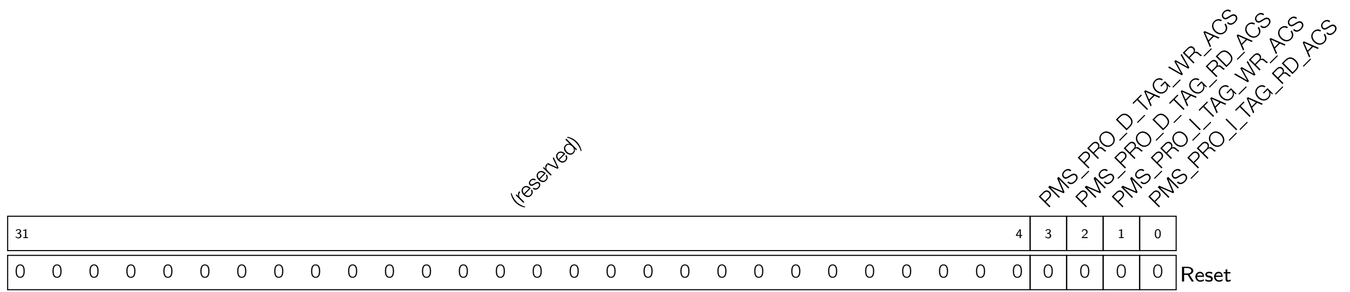
PMS_OCCUPY_CACHE Configure whether SRAM Block 0-3 is used as cache memory. (R/W)

Register 14.35: PMS_OCCUPY_3_REG (0x00E0)



PMS_OCCUPY_PRO_TRACE Configure one out block of Block 4-21 is used as trace memory. (R/W)

Register 14.36: PMS_CACHE_TAG_ACCESS_1_REG (0x00E8)



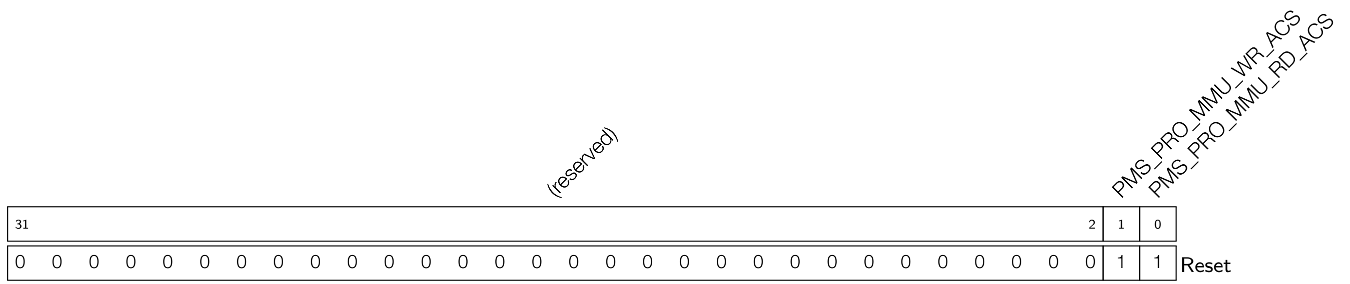
PMS_PRO_I_TAG_RD_ACS Setting to 1 permits read access to Icache tag memory. (R/W)

PMS_PRO_I_TAG_WR_ACS Setting to 1 permits write access to Icache tag memory. (R/W)

PMS_PRO_D_TAG_RD_ACS Setting to 1 permits read access to Dcache tag memory. (R/W)

PMS_PRO_D_TAG_WR_ACS Setting to 1 permits write access to Dcache tag memory. (R/W)

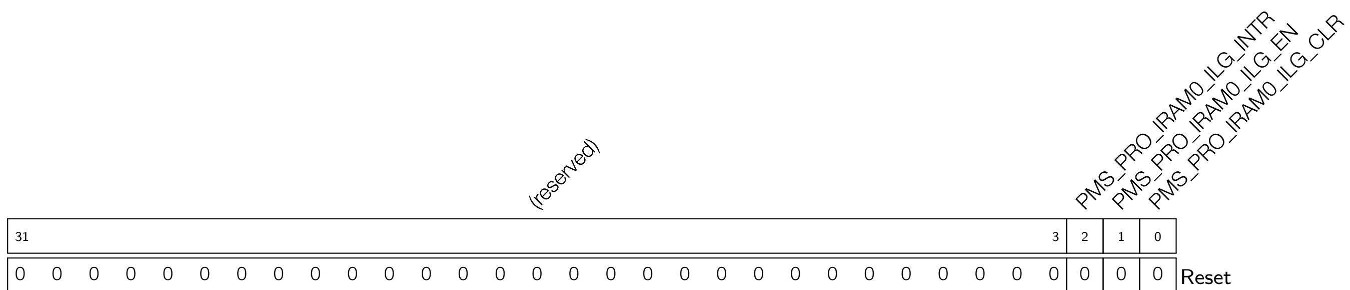
Register 14.37: PMS_CACHE_MMU_ACCESS_1_REG (0x00F0)



PMS_PRO_MMU_RD_ACS Setting to 1 permits read access to MMU memory. (R/W)

PMS_PRO_MMU_WR_ACS Setting to 1 permits write access to MMU memory. (R/W)

Register 14.38: PMS_PRO_IRAM0_4_REG (0x0020)



PMS_PRO_IRAM0_ILG_CLR The clear signal for IBUS access interrupt. (R/W)

PMS_PRO_IRAM0_ILG_EN The enable signal for IBUS access interrupt. (R/W)

PMS_PRO_IRAM0_ILG_INTR IBUS access interrupt signal. (RO)

Register 14.39: PMS_PRO_IRAM0_5_REG (0x0024)

(reserved)	PMS_PRO_IRAM0_ILG_ST	
31 ... 22	21 ... 0	
0 0 0 0 0 0 0 0 0 0	0	Reset

PMS_PRO_IRAM0_ILG_ST Record the illegitimate information of IBUS. [21:2]: store the bits [21:2] of IBUS address; [1]: 1 means data access, 0 means instruction access; [0]: 1 means write operation, 0 means read operation. (RO)

Register 14.40: PMS_PRO_DRAM0_3_REG (0x0034)

(reserved)	PMS_PRO_DRAM0_ILG_INTR	PMS_PRO_DRAM0_ILG_EN	PMS_PRO_DRAM0_ILG_CLR	
31 ... 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	Reset

PMS_PRO_DRAM0_ILG_CLR The clear signal for DBUS0 access interrupt. (R/W)

PMS_PRO_DRAM0_ILG_EN The enable signal for DBUS0 access interrupt. (R/W)

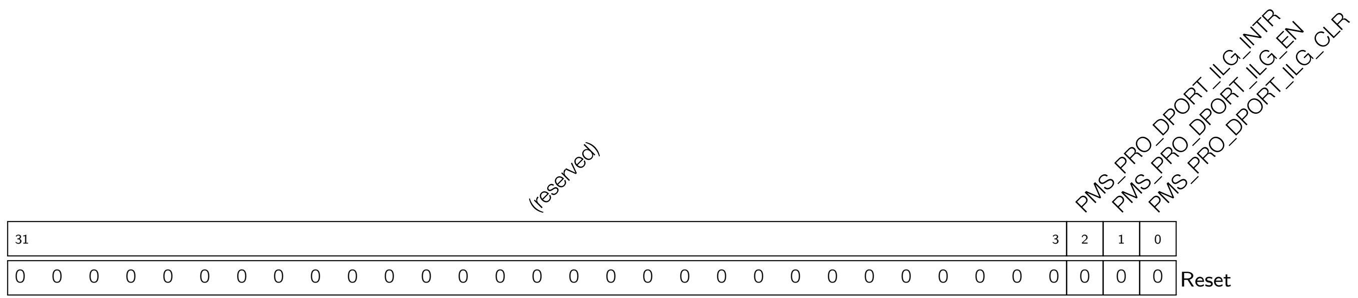
PMS_PRO_DRAM0_ILG_INTR DBUS0 access interrupt signal. (RO)

Register 14.41: PMS_PRO_DRAM0_4_REG (0x0038)

(reserved)	PMS_PRO_DRAM0_ILG_ST	
31 ... 26	25 ... 0	
0 0 0 0 0 0	0	Reset

PMS_PRO_DRAM0_ILG_ST Record the illegitimate information of DBUS. [25:6]: store the bits [21:2] of DBUS address; [5]: 1 means atomic access, 0 means nonatomic access; [4]: 1 means write operation, 0 means read operation; [3:0]: DBUS0 bus byte enables. (RO)

Register 14.42: PMS_PRO_DPORT_6_REG (0x0054)

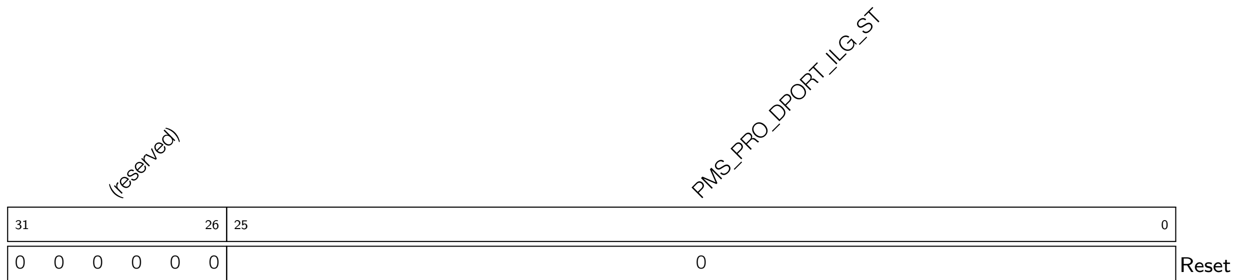


PMS_PRO_DPORT_ILG_CLR The clear signal for PeriBus1 access interrupt. (R/W)

PMS_PRO_DPORT_ILG_EN The enable signal for PeriBus1 access interrupt. (R/W)

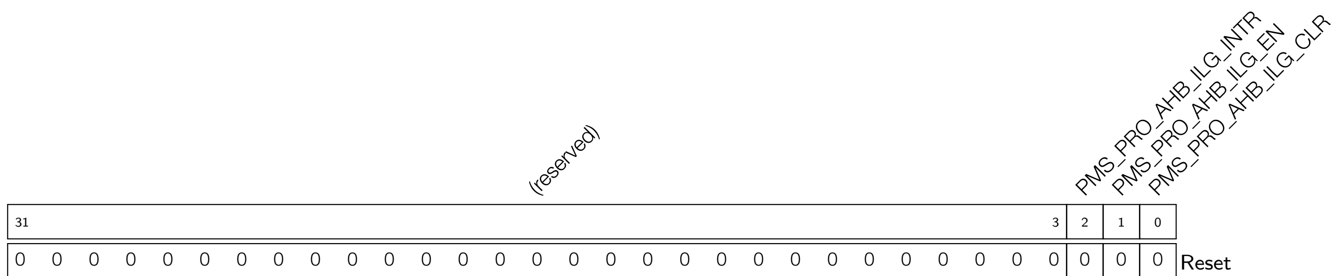
PMS_PRO_DPORT_ILG_INTR PeriBus1 access interrupt signal. (RO)

Register 14.43: PMS_PRO_DPORT_7_REG (0x0058)



PMS_PRO_DPORT_ILG_ST Record the illegitimate information of PeriBus1. [25:6]: store the bits [21:2] of PeriBus1 address; [5]: 1 means atomic access, 0 means nonatomic access; [4]: if bits [31:22] of PeriBus1 address are 0xfd, then the bit value is 1, otherwise it is 0; [3:0]: PeriBus 1 byte enables. (RO)

Register 14.44: PMS_PRO_AHB_3_REG (0x0068)

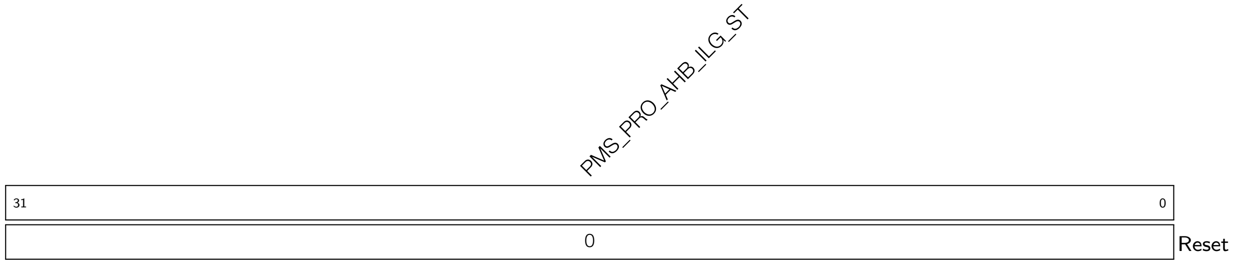


PMS_PRO_AHB_ILG_CLR The clear signal for PeriBus2 access interrupt. (R/W)

PMS_PRO_AHB_ILG_EN The enable signal for PeriBus2 access interrupt. (R/W)

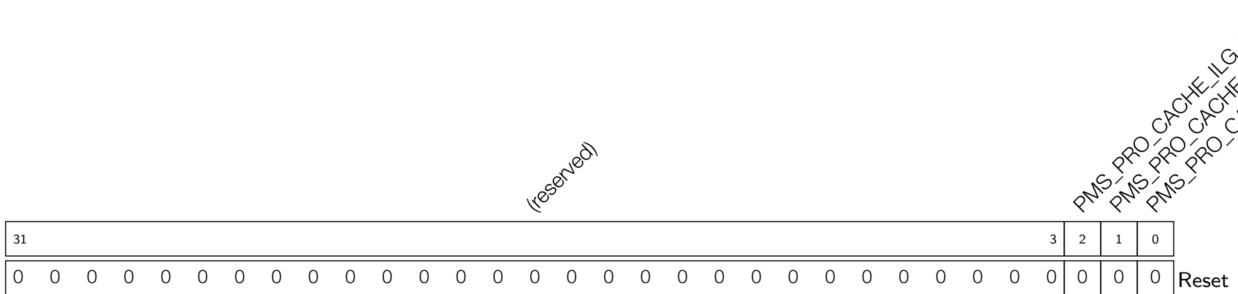
PMS_PRO_AHB_ILG_INTR PeriBus2 access interrupt signal. (RO)

Register 14.45: PMS_PRO_AHB_4_REG (0x006C)



PMS_PRO_AHB_ILG_ST Record the illegitimate information of PeriBus2. [31:2]: store the bits [31:2] of PeriBus2 address; [1]: 1 means data access, 0 means instruction access; [0]: 1 means write operation, 0 means read operation. (RO)

Register 14.46: PMS_PRO_CACHE_2_REG (0x0080)

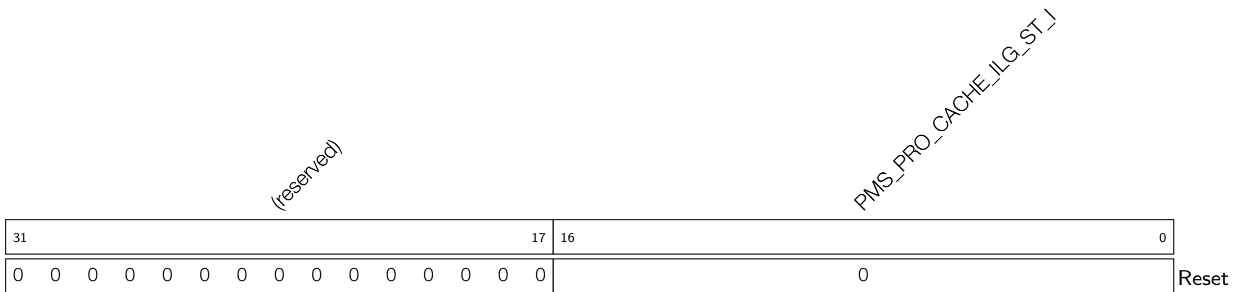


PMS_PRO_CACHE_ILG_CLR The clear signal for cache access interrupt. (R/W)

PMS_PRO_CACHE_ILG_EN The enable signal for cache access interrupt. (R/W)

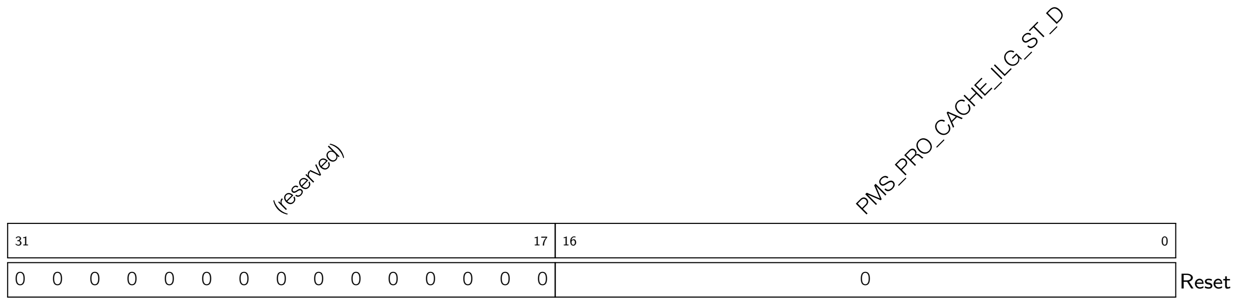
PMS_PRO_CACHE_ILG_INTR Cache access interrupt signal. (RO)

Register 14.47: PMS_PRO_CACHE_3_REG (0x0084)



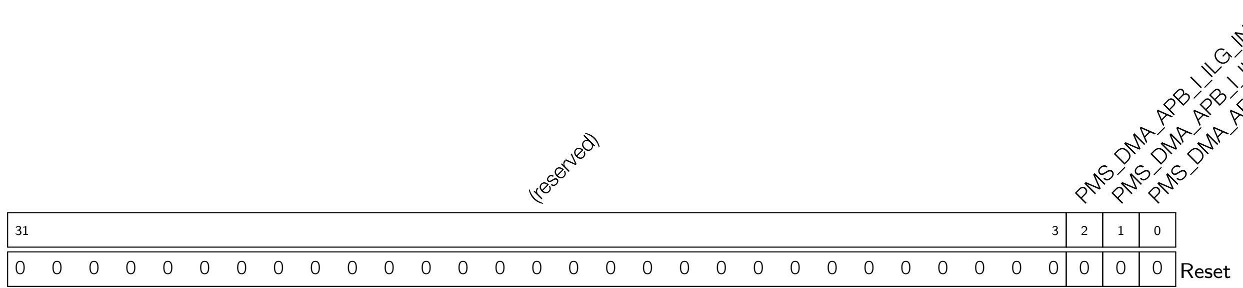
PMS_PRO_CACHE_ILG_ST_I Record the illegitimate information of Icache to access memory. [16]: access enable, active low; [15:4]: store the bits [11:0] of address; [3:0]: Icache bus byte enables, active low. (RO)

Register 14.48: PMS_PRO_CACHE_4_REG (0x0088)



PMS_PRO_CACHE_ILG_ST_D Record the illegitimate information of Dcache to access memory. [16]: access enable, active low; [15:4]: store the bits [11:0] of address; [3:0]: Dcache bus byte enables, active low. (RO)

Register 14.49: PMS_DMA_APB_I_2_REG (0x0094)

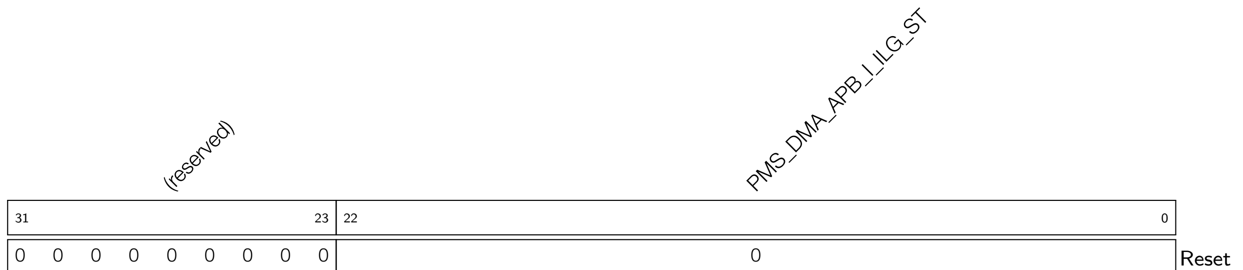


PMS_DMA_APB_I_ILG_CLR The clear signal for internal DMA access interrupt. (R/W)

PMS_DMA_APB_I_ILG_EN The enable signal for internal DMA access interrupt. (R/W)

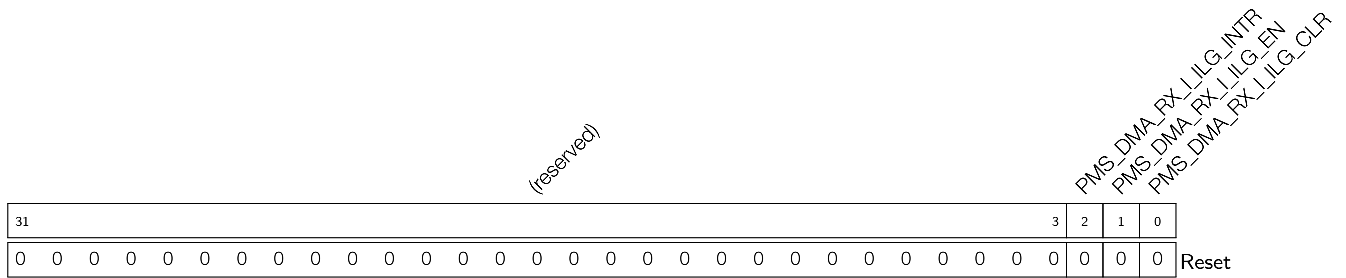
PMS_DMA_APB_I_ILG_INTR Internal DMA access interrupt signal. (RO)

Register 14.50: PMS_DMA_APB_I_3_REG (0x0098)



PMS_DMA_APB_I_ILG_ST Record the illegal information of Internal DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: Internal DMA bus byte enable; (RO)

Register 14.51: PMS_DMA_RX_I_2_REG (0x00A4)

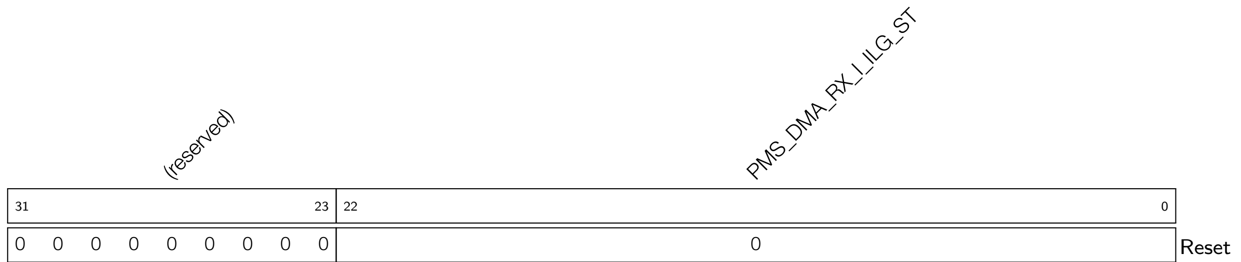


PMS_DMA_RX_I_ILG_CLR The clear signal for RX Copy DMA access interrupt. (R/W)

PMS_DMA_RX_I_ILG_EN The enable signal for RX Copy DMA access interrupt. (R/W)

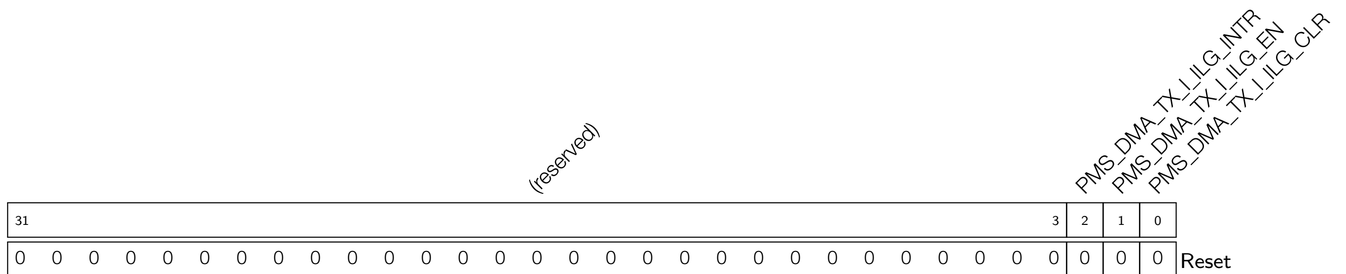
PMS_DMA_RX_I_ILG_INTR RX Copy DMA access interrupt signal. (RO)

Register 14.52: PMS_DMA_RX_I_3_REG (0x00A8)



PMS_DMA_RX_I_ILG_ST Record the illegitimate information of RX Copy DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: RX Copy DMA bus byte enables. (RO)

Register 14.53: PMS_DMA_TX_I_2_REG (0x00B4)



PMS_DMA_TX_I_ILG_CLR The clear signal for TX Copy DMA access interrupt. (R/W)

PMS_DMA_TX_I_ILG_EN The enable signal for TX Copy DMA access interrupt. (R/W)

PMS_DMA_TX_I_ILG_INTR TX Copy DMA access interrupt signal. (RO)

Register 14.54: PMS_DMA_TX_I_3_REG (0x00B8)

31–23 (reserved)	22–0 PMS_DMA_TX_I_ILG_ST	
0 0 0 0 0 0 0 0 0	0	Reset

PMS_DMA_TX_I_ILG_ST Record the illegitimate information of TX Copy DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: TX Copy DMA bus byte enables. (RO)

Register 14.55: PMS_APB_PERIPHERAL_INTR_REG (0x00F4)

31–3 (reserved)	2 PMS_APB_PERI_BYTE_ERROR_INTR	1 PMS_APB_PERI_BYTE_ERROR_EN	0 PMS_APB_PERI_BYTE_ERROR_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	Reset

PMS_APB_PERI_BYTE_ERROR_CLR The clear signal for APB peripheral interrupt. (R/W)

PMS_APB_PERI_BYTE_ERROR_EN The enable signal for APB peripheral access interrupt. (R/W)

PMS_APB_PERI_BYTE_ERROR_INTR APB peripheral access interrupt signal. (RO)

Register 14.56: PMS_APB_PERIPHERAL_STATUS_REG (0x00F8)

31–0 PMS_APB_PERI_BYTE_ERROR_ADDR	
0	Reset

PMS_APB_PERI_BYTE_ERROR_ADDR Record the illegitimate address of APB peripheral. (RO)

Register 14.57: PMS_CPU_PERIPHERAL_INTR_REG (0x00FC)

Bits	31–3	2	1	0
Field	(reserved)	PMS_CPU_PERI_BYTE_ERROR_INTR	PMS_CPU_PERI_BYTE_ERROR_EN	PMS_CPU_PERI_BYTE_ERROR_CLR
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0

PMS_CPU_PERI_BYTE_ERROR_CLR The clear signal for CPU peripheral access interrupt. (R/W)

PMS_CPU_PERI_BYTE_ERROR_EN The enable signal for CPU peripheral access interrupt. (R/W)

PMS_CPU_PERI_BYTE_ERROR_INTR CPU peripheral access interrupt signal. (RO)

Register 14.58: PMS_CPU_PERIPHERAL_STATUS_REG (0x0100)

Bits	31–0
Field	PMS_CPU_PERI_BYTE_ERROR_ADDR
Reset	0

PMS_CPU_PERI_BYTE_ERROR_ADDR Record the illegitimate address of CPU peripheral. (RO)

Register 14.59: PMS_DATE_REG (0x0FFC)

Bits	31–28	27–0
Field	(reserved)	PMS_DATE
Reset	0 0 0 0	0x1905090

PMS_DATE Version control register. (R/W)

15. System Registers (SYSTEM)

15.1 Overview

The ESP32-S2 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. Specifically, ESP32-S2 has a various of system configuration registers that can be used for the chip's clock management (clock gating), power management, and the configuration of peripherals and core-system modules. This chapter lists all these system registers and their functions.

15.2 Features

ESP32-S2 system registers can be used to control the following peripheral blocks and core modules:

- System and memory
- Reset and clock
- Interrupt matrix
- eFuse controller
- Low-power management
- Peripheral clock gating and reset

15.3 Function Description

15.3.1 System and Memory Registers

The following registers are used for system and memory configuration, such as cache configuration and memory remapping. For additional information, please refer to Chapter 3 *System and Memory*.

- SYSTEM_ROM_CTRL_0_REG
- SYSTEM_ROM_CTRL_1_REG
- SYSTEM_SRAM_CTRL_0_REG
- SYSTEM_SRAM_CTRL_1_REG
- SYSTEM_SRAM_CTRL_2_REG
- SYSTEM_RSA_PD_CTRL_REG
- SYSTEM_MEM_PD_MASK_REG
- SYSTEM_CACHE_CONTROL_REG
- SYSTEM_BUSTOEXTMEM_ENA_REG
- SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG

ROM Power Consumption Control

Registers SYSTEM_ROM_CTRL_0_REG and SYSTEM_ROM_CTRL_1_REG can be used to control the power consumption of ESP32-S2's ROM. Specifically:

- Setting different bits of the SYSTEM_ROM_FO field in register SYSTEM_ROM_CTRL_0_REG forces on the clock gates of different blocks of ROM.
- Setting different bits of the SYSTEM_ROM_FORCE_PD field in register SYSTEM_ROM_CTRL_1_REG powers down different blocks of internal ROM.
- Setting different bits of the SYSTEM_ROM_FORCE_PU field in register SYSTEM_ROM_CTRL_1_REG powers up different blocks of internal ROM.

For detailed information about the controlling bits of different blocks, please see Table 85 below.

Table 85: ROM Controlling Bit

ROM	Lowest Address1	Highest Address1	Lowest Address2	Highest Address2	Controlling Bit
Block0	0x4000_0000	0x4000_FFFF	-	-	Bit0
Block1	0x4001_2000	0x4001_FFFF	0x3FFA_0000	0x3FFA_FFFF	Bit1

SRAM Power Consumption Control

Registers SYSTEM_SRAM_CTRL_0_REG, SYSTEM_SRAM_CTRL_1_REG, and SYSTEM_SRAM_CTRL_2_REG can be used to control the power consumption of ESP32-S2's internal SRAM. Specifically,

- Setting different bits of the SYSTEM_SRAM_FO field in register SYSTEM_SRAM_CTRL_0_REG forces on the clock gates of different blocks of internal SRAM.
- Setting different bits of the SYSTEM_SRAM_FORCE_PD field in register SYSTEM_SRAM_CTRL_1_REG powers down different blocks of internal SRAM.
- Setting different bits of the SYSTEM_SRAM_FORCE_PU field in register SYSTEM_SRAM_CTRL_2_REG powers up different blocks of internal SRAM.

For detailed information about the controlling bits of different blocks, please see Table 86 below.

Table 86: SRAM Controlling Bit

SRAM	Lowest Address1	Highest Address1	Lowest Address2	Highest Address2	Controlling Bit
Block0	0x4002_0000	0x4002_1FFF	0x3FFB_0000	0x3FFB_1FFF	Bit0
Block1	0x4002_2000	0x4002_3FFF	0x3FFB_2000	0x3FFB_3FFF	Bit1
Block2	0x4002_4000	0x4002_5FFF	0x3FFB_4000	0x3FFB_5FFF	Bit2
Block3	0x4002_6000	0x4002_7FFF	0x3FFB_6000	0x3FFB_7FFF	Bit3
Block4	0x4002_8000	0x4002_BFFF	0x3FFB_8000	0x3FFB_BFFF	Bit4
Block5	0x4002_C000	0x4002_FFFF	0x3FFB_C000	0x3FFB_FFFF	Bit5
Block6	0x4003_0000	0x4003_3FFF	0x3FFC_0000	0x3FFC_3FFF	Bit6
Block7	0x4003_4000	0x4003_7FFF	0x3FFC_4000	0x3FFC_7FFF	Bit7
Block8	0x4003_8000	0x4003_BFFF	0x3FFC_8000	0x3FFC_BFFF	Bit8
Block9	0x4003_C000	0x4003_FFFF	0x3FFC_C000	0x3FFC_FFFF	Bit9
Block10	0x4004_0000	0x4004_3FFF	0x3FFD_0000	0x3FFD_3FFF	Bit10
Block11	0x4004_4000	0x4004_7FFF	0x3FFD_4000	0x3FFD_7FFF	Bit11
Block12	0x4004_8000	0x4004_BFFF	0x3FFD_8000	0x3FFD_BFFF	Bit12
Block13	0x4004_C000	0x4004_FFFF	0x3FFD_C000	0x3FFD_FFFF	Bit13
Block14	0x4005_0000	0x4005_3FFF	0x3FFE_0000	0x3FFE_3FFF	Bit14

Block15	0x4005_4000	0x4005_7FFF	0x3FFE_4000	0x3FFE_7FFF	Bit15
Block16	0x4005_8000	0x4005_BFFF	0x3FFE_8000	0x3FFE_BFFF	Bit16
Block17	0x4005_C000	0x4005_FFFF	0x3FFE_C000	0x3FFE_FFFF	Bit17
Block18	0x4006_0000	0x4006_3FFF	0x3FFF_0000	0x3FFF_3FFF	Bit18
Block19	0x4006_4000	0x4006_7FFF	0x3FFF_4000	0x3FFF_7FFF	Bit19
Block20	0x4006_8000	0x4006_BFFF	0x3FFF_8000	0x3FFF_BFFF	Bit20
Block21	0x4006_C000	0x4006_FFFF	0x3FFF_C000	0x3FFF_FFFF	Bit21

15.3.2 Reset and Clock Registers

The following registers are used for reset and clock. For additional information, please refer to Chapter 6 *Reset and Clock*.

- SYSTEM_CPU_PER_CONF_REG
- SYSTEM_SYSCLK_CONF_REG
- SYSTEM_BT_LPCK_DIV_FRAC_REG

15.3.3 Interrupt Matrix Registers

The following registers are used for generating the CPU interrupt signals for the interrupt matrix. For additional information, please refer to Chapter 8 *Interrupt Matrix (INTERRUPT)*

- SYSTEM_CPU_INTR_FROM_CPU_0_REG
- SYSTEM_CPU_INTR_FROM_CPU_1_REG
- SYSTEM_CPU_INTR_FROM_CPU_2_REG
- SYSTEM_CPU_INTR_FROM_CPU_3_REG

15.3.4 JTAG Software Enable Registers

The following registers are used for revoking the temporary disable of eFuse to JTAG. For additional information, please refer to Chapter 19 *HMAC Accelerator (HMAC)*.

- SYSTEM_JTAG_CTRL_0_REG
- SYSTEM_JTAG_CTRL_1_REG
- SYSTEM_JTAG_CTRL_2_REG
- SYSTEM_JTAG_CTRL_3_REG
- SYSTEM_JTAG_CTRL_4_REG
- SYSTEM_JTAG_CTRL_5_REG
- SYSTEM_JTAG_CTRL_6_REG
- SYSTEM_JTAG_CTRL_7_REG

15.3.5 Low-power Management Registers

The following registers are used for low-power management. For additional information, please refer to Chapter 9 *Low-Power Management (RTC_CNTL)*.

- SYSTEM_RTC_FASTMEM_CONFIG_REG

- SYSTEM_RTC_FASTMEM_CRC_REG

15.3.6 Peripheral Clock Gating and Reset Registers

The following registers are used for controlling the clock gating and reset of different peripherals. Details can be seen in Table 87.

- SYSTEM_CPU_PERI_CLK_EN_REG
- SYSTEM_CPU_PERI_RST_EN_REG
- SYSTEM_PERIP_CLK_EN0_REG
- SYSTEM_PERIP_RST_EN0_REG
- SYSTEM_PERIP_CLK_EN1_REG
- SYSTEM_PERIP_RST_EN1_REG

Table 87: Peripheral Clock Gating and Reset Bits

Peripheral	Clock Enabling Bit[1]	Reset Controlling Bit[2][3]
CPU Peripherals	**SYSTEM_CPU_PERI_CLK_EN_REG**	**SYSTEM_CPU_PERI_RST_EN_REG**
DEDICATED GPIO	SYSTEM_CLK_EN_DEDICATED_GPIO	SYSTEM_RST_EN_DEDICATED_GPIO
Peripherals	**SYSTEM_PERIP_CLK_EN0_REG**	**SYSTEM_PERIP_RST_EN0_REG**
Timers	SYSTEM_TIMERS_CLK_EN	SYSTEM_TIMERS_RST
Timer Group0	SYSTEM_TIMERGROUP_CLK_EN	SYSTEM_TIMERGROUP_RST
Timer Group1	SYSTEM_TIMERGROUP1_CLK_EN	SYSTEM_TIMERGROUP1_RST
System Timer	SYSTEM_SYSTIMER_CLK_EN	SYSTEM_SYSTIMER_RST
UART0	SYSTEM_UART_CLK_EN	SYSTEM_UART_RST
UART1	SYSTEM_UART1_CLK_EN	SYSTEM_UART1_RST
UART MEM	SYSTEM_UART_MEM_CLK_EN [4]	SYSTEM_UART_MEM_RST
SPI0, SPI1	SYSTEM_SPI01_CLK_EN	SYSTEM_SPI01_RST
SPI2	SYSTEM_SPI2_CLK_EN	SYSTEM_SPI2_RST
SPI3	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_RST
SPI4	SYSTEM_SPI4_CLK_EN	SYSTEM_SPI4_RST
SPI2 DMA	SYSTEM_SPI2_DMA_CLK_EN	SYSTEM_SPI2_DMA_RST
SPI3 DMA	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_DMA_RST
I2C0	SYSTEM_I2C_EXT0_CLK_EN	SYSTEM_I2C_EXT0_RST
I2C1	SYSTEM_I2C_EXT1_CLK_EN	SYSTEM_I2C_EXT1_RST
I2S0	SYSTEM_I2S0_CLK_EN	SYSTEM_I2S0_RST
I2S1	SYSTEM_I2S1_CLK_EN	SYSTEM_I2S1_RST
TWAI Controller	SYSTEM_CAN_CLK_EN	SYSTEM_CAN_RST
UHCI0	SYSTEM_UHCI0_CLK_EN	SYSTEM_UHCI0_RST
UHCI1	SYSTEM_UHCI1_CLK_EN	SYSTEM_UHCI1_RST
USB	SYSTEM_USB_CLK_EN	SYSTEM_USB_RST
RMT	SYSTEM_RMT_CLK_EN	SYSTEM_RMT_RST
PCNT	SYSTEM_PCNT_CLK_EN	SYSTEM_PCNT_RST
PWM0	SYSTEM_PWM0_CLK_EN	SYSTEM_PWM0_RST
PWM1	SYSTEM_PWM1_CLK_EN	SYSTEM_PWM1_RST

PWM2	SYSTEM_PWM2_CLK_EN	SYSTEM_PWM2_RST
PWM3	SYSTEM_PWM3_CLK_EN	SYSTEM_PWM3_RST
LED_PWM Controller	SYSTEM_LEDC_CLK_EN	SYSTEM_LEDC_RST
eFuse	SYSTEM_EFUSE_CLK_EN	SYSTEM_EFUSE_RST
APB SARADC	SYSTEM_APB_SARADC_CLK_EN	SYSTEM_APB_SARADC_RST
ADC2 ARB	SYSTEM_ADC2_ARB_CLK_EN	SYSTEM_ADC2_ARB_RST
WDG	SYSTEM_WDG_CLK_EN	SYSTEM_WDG_RST
Accelerators	**SYSTEM_PERIP_CLK_EN1_REG**	**SYSTEM_PERIP_RST_EN1_REG**
DMA	SYSTEM_CRYPTO_DMA_CLK_EN	SYSTEM_CRYPTO_DMA_RST [5]
HMAC	SYSTEM_CRYPTO_HMAC_CLK_EN	SYSTEM_CRYPTO_HMAC_RST [6]
Digital Signature	SYSTEM_CRYPTO_DS_CLK_EN	SYSTEM_CRYPTO_DS_RST [7]
RSA Accelerator	SYSTEM_CRYPTO_RSA_CLK_EN	SYSTEM_CRYPTO_RSA_RST
SHA Accelerator	SYSTEM_CRYPTO_SHA_CLK_EN	SYSTEM_CRYPTO_SHA_RST
AES Accelerator	SYSTEM_CRYPTO_AES_CLK_EN	SYSTEM_CRYPTO_AES_RST

Note:

1. Set the clock enable register to 1 to enable the clock, and to 0 to disable the clock;
2. Set the reset enabling register to 1 to reset a peripheral, and to 0 to disable the reset.
3. Reset registers are not cleared by hardware.
4. UART memory is shared by all UART peripherals, meaning having any active UART peripherals will prevent the UART memory from entering the clock-gated state.
5. Crypto DMA is shared by AES and SHA accelerators.
6. Resetting this bit also resets the SHA accelerator.
7. Resetting this bit also resets the AES, SHA, and RSA accelerators.

15.4 Base Address

Users can access the system registers with base address, which can be seen in the following table. For more information about accessing system registers, please see Chapter 3 *System and Memory*.

Table 88: System Register Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C0000

15.5 Register Summary

The addresses in the following table are relative to the system registers base addresses provided in Section 15.4.

Name	Description	Address	Access
System and Memory Registers			
SYSTEM_ROM_CTRL_0_REG	System ROM configuration register 0	0x0000	R/W
SYSTEM_ROM_CTRL_1_REG	System ROM configuration register 1	0x0004	R/W
SYSTEM_SRAM_CTRL_0_REG	System SRAM configuration register 0	0x0008	R/W
SYSTEM_SRAM_CTRL_1_REG	System SRAM configuration register 1	0x000C	R/W
SYSTEM_SRAM_CTRL_2_REG	System SRAM configuration register 2	0x0088	R/W
SYSTEM_MEM_PD_MASK_REG	Memory power-related controlling register (under low-sleep)	0x003C	R/W
SYSTEM_RSA_PD_CTRL_REG	RSA memory remapping register	0x0068	R/W
SYSTEM_BUSTOEXTMEM_ENA_REG	EDMA enable register	0x006C	R/W
SYSTEM_CACHE_CONTROL_REG	Cache control register	0x0070	R/W
SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	External memory encrypt and decrypt controlling register	0x0074	R/W
Reset and Clock Registers			
SYSTEM_CPU_PER_CONF_REG	CPU peripheral clock configuration register	0x0018	R/W
SYSTEM_BT_LPCK_DIV_FRAC_REG	Divider fraction configuration register for low-power clock	0x0054	R/W
SYSTEM_SYSCLK_CONF_REG	SoC clock configuration register	0x008C	Varies
Interrupt Matrix Registers			
SYSTEM_CPU_INTR_FROM_CPU_0_REG	CPU interrupt controlling register 0	0x0058	R/W
SYSTEM_CPU_INTR_FROM_CPU_1_REG	CPU interrupt controlling register 1	0x005C	R/W
SYSTEM_CPU_INTR_FROM_CPU_2_REG	CPU interrupt controlling register 2	0x0060	R/W
SYSTEM_CPU_INTR_FROM_CPU_3_REG	CPU interrupt controlling register 3	0x0064	R/W
JTAG Software Enable Registers			
SYSTEM_JTAG_CTRL_0_REG	JTAG configuration register 0	0x001C	WO
SYSTEM_JTAG_CTRL_1_REG	JTAG configuration register 1	0x0020	WO
SYSTEM_JTAG_CTRL_2_REG	JTAG configuration register 2	0x0024	WO
SYSTEM_JTAG_CTRL_3_REG	JTAG configuration register 3	0x0028	WO
SYSTEM_JTAG_CTRL_4_REG	JTAG configuration register 4	0x002C	WO
SYSTEM_JTAG_CTRL_5_REG	JTAG configuration register 5	0x0030	WO
SYSTEM_JTAG_CTRL_6_REG	JTAG configuration register 6	0x0034	WO

Name	Description	Address	Access
SYSTEM_JTAG_CTRL_7_REG	JTAG configuration register 7	0x0038	WO
Low-Power Management Registers			
SYSTEM_RTC_FASTMEM_CONFIG_REG	RTC fast memory configuration register	0x0078	Varies
SYSTEM_RTC_FASTMEM_CRC_REG	RTC fast memory CRC controlling register	0x007C	RO
Peripheral Clock Gating and Reset Registers			
SYSTEM_CPU_PERI_CLK_EN_REG	CPU peripheral clock enable register	0x0010	R/W
SYSTEM_CPU_PERI_RST_EN_REG	CPU peripheral reset register	0x0014	R/W
SYSTEM_PERIP_CLK_EN0_REG	System peripheral clock (for hardware accelerators) enable register 0	0x0040	R/W
SYSTEM_PERIP_CLK_EN1_REG	System peripheral clock (for hardware accelerators) enable register 1	0x0044	R/W
SYSTEM_PERIP_RST_EN0_REG	System peripheral (hardware accelerators) reset register 0	0x0048	R/W
SYSTEM_PERIP_RST_EN1_REG	System peripheral (hardware accelerators) reset register 1	0x004C	R/W
Version Register			
SYSTEM_DATE_REG	Version control register	0x0FFC	R/W

15.6 Registers

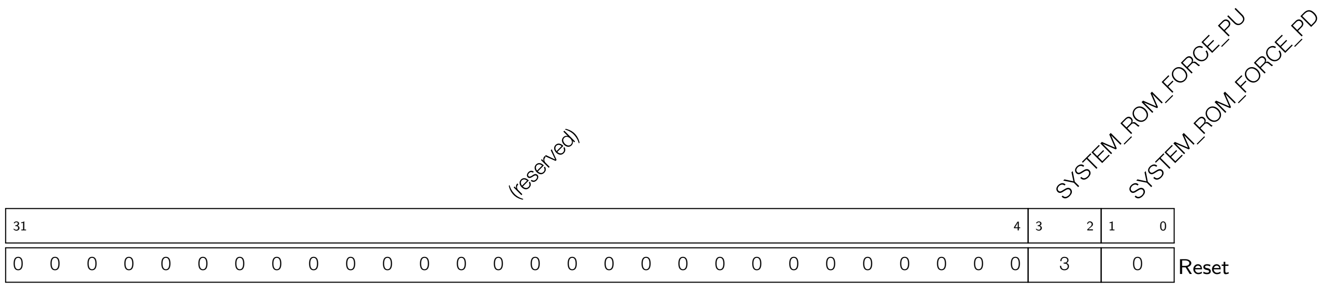
The addresses below are relative to the system registers base addresses provided in Section 15.4.

Register 15.1: SYSTEM_ROM_CTRL_0_REG (0x0000)

(reserved)	SYSTEM_ROM_FO	
31 … 2	1 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x3	Reset

SYSTEM_ROM_FO This field is used to force on clock gate of internal ROM. For details, please refer to Table 85. (R/W)

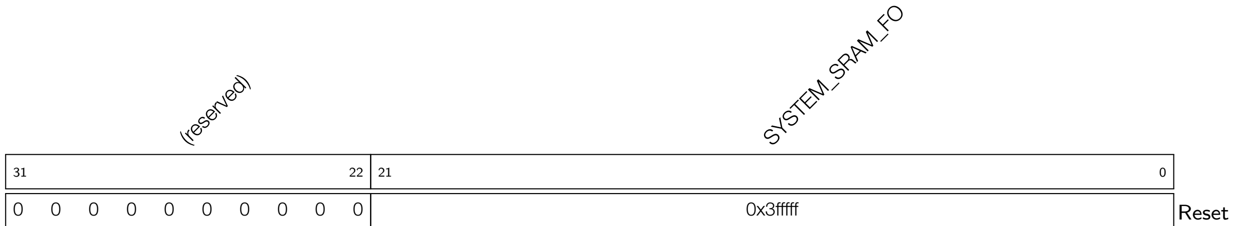
Register 15.2: SYSTEM_ROM_CTRL_1_REG (0x0004)



SYSTEM_ROM_FORCE_PD This field is used to power down internal ROM. For details, please refer to Table 85. (R/W)

SYSTEM_ROM_FORCE_PU This field is used to power up internal ROM. For details, please refer to Table 85. (R/W)

Register 15.3: SYSTEM_SRAM_CTRL_0_REG (0x0008)



SYSTEM_SRAM_FO This field is used to force on clock gate of internal SRAM. For details, please refer to Table 86. (R/W)

Register 15.4: SYSTEM_SRAM_CTRL_1_REG (0x000C)

SYSTEM_SRAM_FORCE_PD This field is used to power down internal SRAM. For details, please refer to Table 86. (R/W)

Register 15.5: SYSTEM_CPU_PERI_CLK_EN_REG (0x0010)

(reserved)	SYSTEM_CLK_EN_DEDICATED_GPIO	(reserved)	(reserved)	
31 … 8	7	6	5 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0	Reset

SYSTEM_CLK_EN_DEDICATED_GPIO Set this bit to enable clock of DEDICATED GPIO module. For details, please refer to Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*. (R/W)

Register 15.6: SYSTEM_CPU_PERI_RST_EN_REG (0x0014)

(reserved)	SYSTEM_RST_EN_DEDICATED_GPIO	(reserved)	(reserved)	
31 … 8	7	6	5 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	0 0 0 0 0 0	Reset

SYSTEM_RST_EN_DEDICATED_GPIO Set this bit to reset DEDICATED GPIO module. (R/W)

Register 15.7: SYSTEM_CPU_PER_CONF_REG (0x0018)

(reserved)	SYSTEM_CPU_WAITI_DELAY_NUM	SYSTEM_CPU_WAIT_MODE_FORCE_ON	SYSTEM_PLL_FREQ_SEL	SYSTEM_CPUPERIOD_SEL	
31 ... 8	7 ... 4	3	2	1 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	1	1	0	Reset

SYSTEM_CPUPERIOD_SEL This field is used to select the clock frequency of CPU or CPU period. For details, please refer to Table 46 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_PLL_FREQ_SEL This field is used to select the PLL clock frequency based on CPU period. For details, please refer to Table 46 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_CPU_WAIT_MODE_FORCE_ON Set this bit to force on CPU wait mode. In this mode, the clock gate of CPU is turned off until any interrupts happen. This mode could also be force on via WAITI instruction. (R/W)

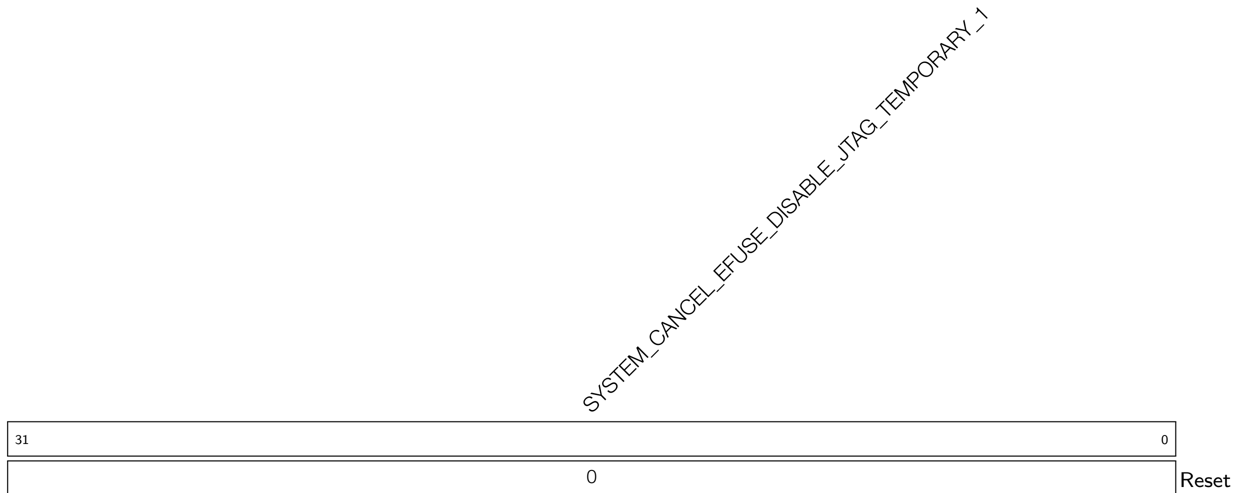
SYSTEM_CPU_WAITI_DELAY_NUM Sets the number of delay cycles to enter CPU wait mode after a WAITI instruction. (R/W)

Register 15.8: SYSTEM_JTAG_CTRL_0_REG (0x001C)

SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0	
31 ... 0	
0	Reset

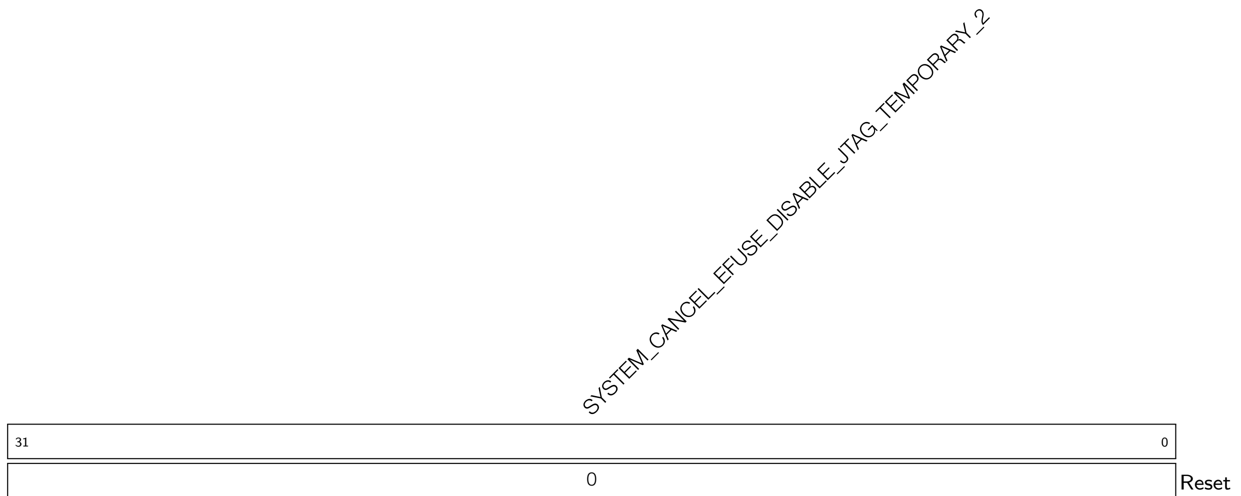
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 Stores the 0 to 31 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.9: SYSTEM_JTAG_CTRL_1_REG (0x0020)



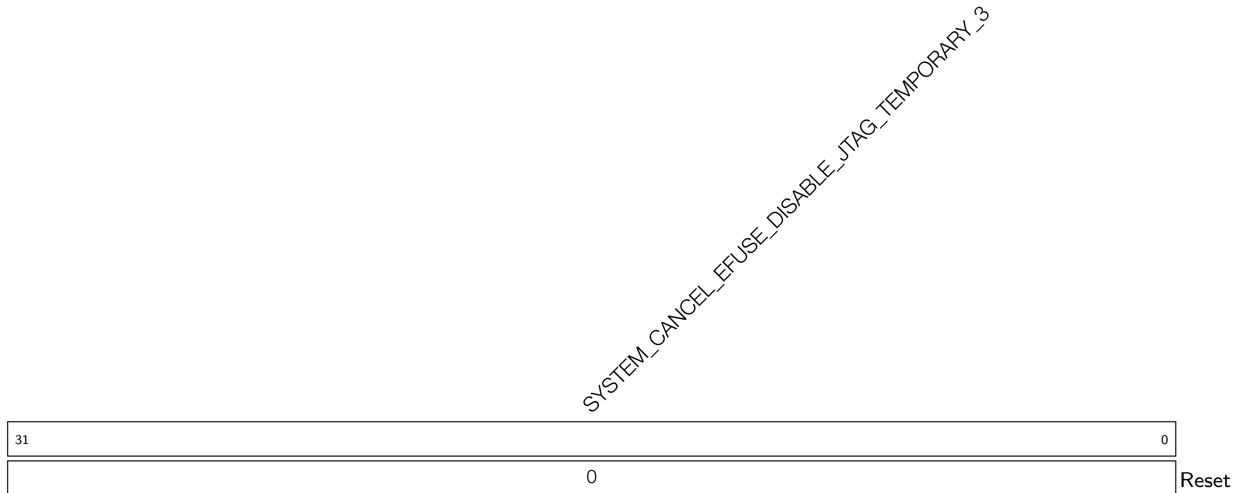
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_1 Stores the 32 to 63 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.10: SYSTEM_JTAG_CTRL_2_REG (0x0024)



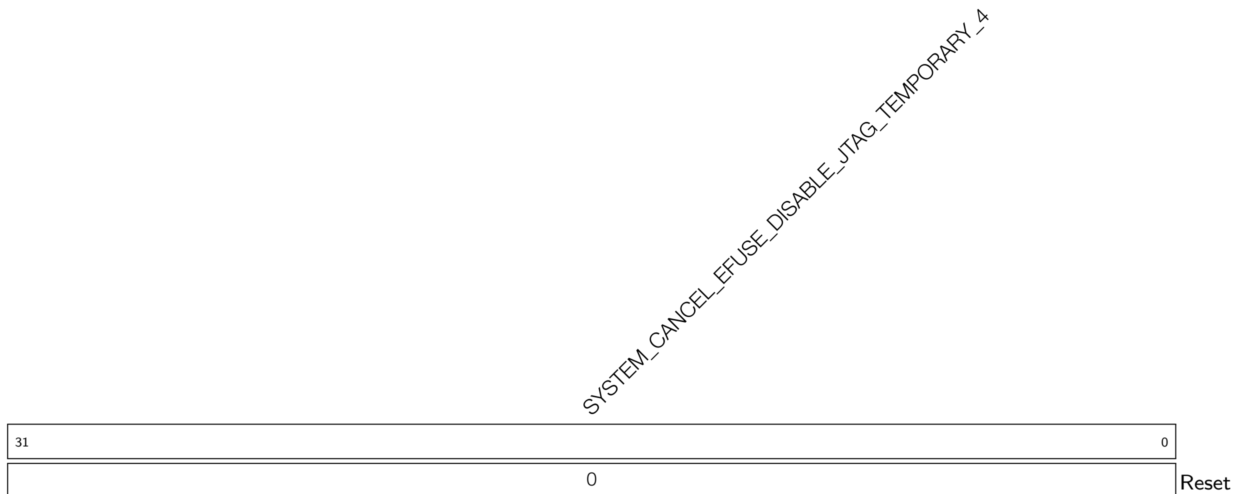
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_2 Stores the 64 to 95 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.11: SYSTEM_JTAG_CTRL_3_REG (0x0028)



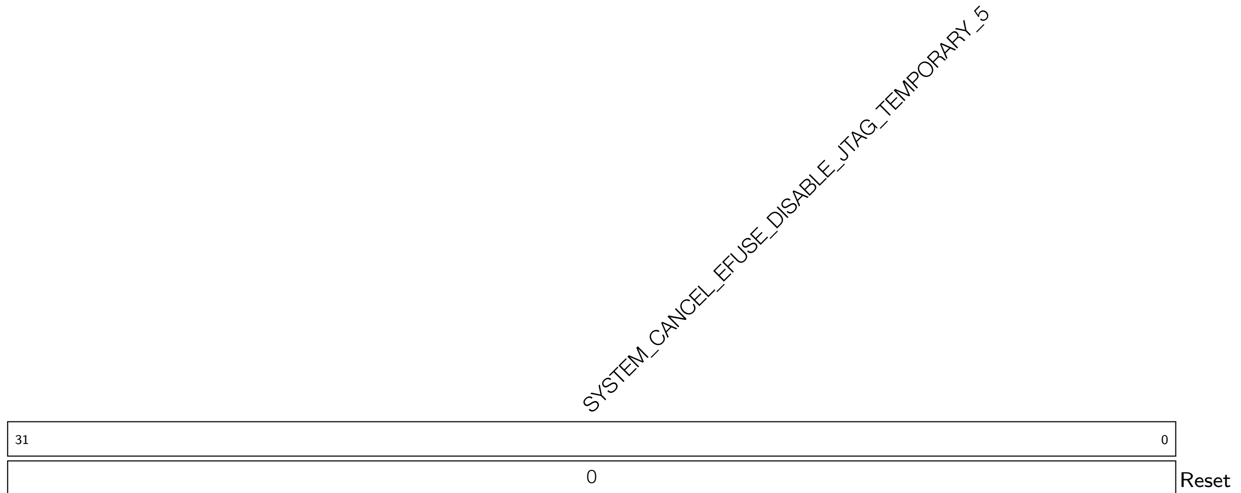
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_3 Stores the 96 to 127 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.12: SYSTEM_JTAG_CTRL_4_REG (0x002C)



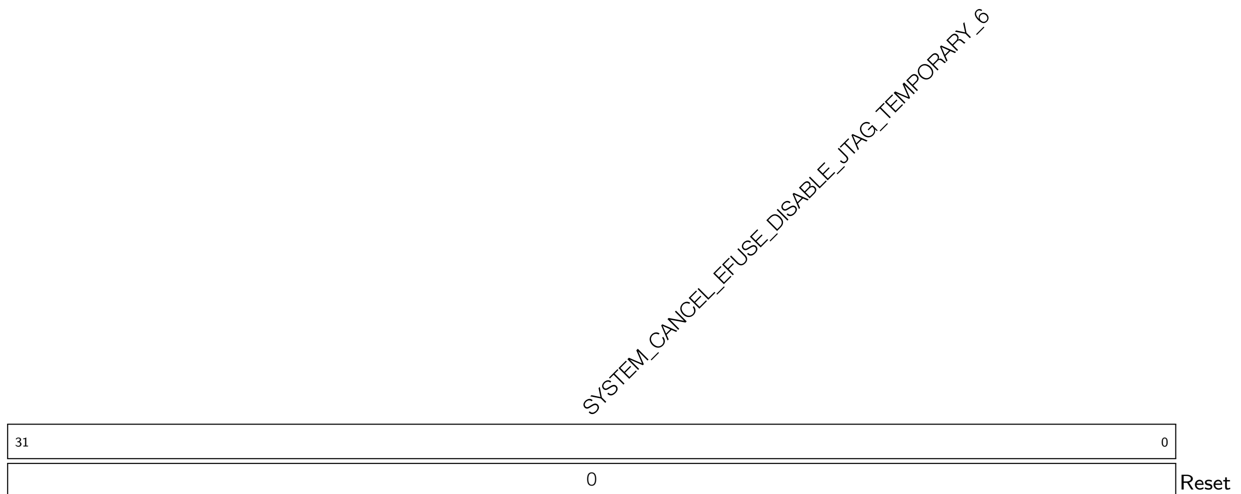
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_4 Stores the 128 to 159 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.13: SYSTEM_JTAG_CTRL_5_REG (0x0030)



SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_5 Stores the 160 to 191 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.14: SYSTEM_JTAG_CTRL_6_REG (0x0034)



SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_6 Stores the 192 to 223 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.15: SYSTEM_JTAG_CTRL_7_REG (0x0038)

SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7 (31..0)	
0	Reset

SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7 Stores the 224 to 255 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG. For details, please refer to Chapter 19 *HMAC Accelerator (HMAC)*. (WO)

Register 15.16: SYSTEM_MEM_PD_MASK_REG (0x003C)

(reserved) (31..1)	SYSTEM_LSLP_MEM_PD_MASK (0)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

SYSTEM_LSLP_MEM_PD_MASK Set this bit to allow the memory to work as usual when the chip enters light sleep. (R/W)

Register 15.17: SYSTEM_PERIP_CLK_EN0_REG (0x0040)

Bit	Field	Reset
31	SYSTEM_SPI4_CLK_EN	1
30	SYSTEM_ADC2_ARB_CLK_EN	1
29	SYSTEM_SYSTIMER_CLK_EN	1
28	SYSTEM_APB_SARADC_CLK_EN	1
27	SYSTEM_SPI3_DMA_CLK_EN	1
26	SYSTEM_PWM3_CLK_EN	0
25	SYSTEM_PWM2_CLK_EN	0
24	SYSTEM_UART_MEM_CLK_EN	1
23	SYSTEM_USB_CLK_EN	1
22	SYSTEM_SPI2_DMA_CLK_EN	1
21	SYSTEM_I2S1_CLK_EN	0
20	SYSTEM_PWM1_CLK_EN	0
19	SYSTEM_CAN_CLK_EN	0
18	SYSTEM_I2C_EXT1_CLK_EN	0
17	SYSTEM_PWM0_CLK_EN	0
16	SYSTEM_SPI3_CLK_EN	1
15	SYSTEM_TIMERGROUP1_CLK_EN	1
14	SYSTEM_EFUSE_CLK_EN	1
13	SYSTEM_TIMERGROUP_CLK_EN	1
12	SYSTEM_UHCI1_CLK_EN	0
11	SYSTEM_LEDC_CLK_EN	0
10	SYSTEM_PCNT_CLK_EN	0
9	SYSTEM_RMT_CLK_EN	0
8	SYSTEM_UHCI0_CLK_EN	0
7	SYSTEM_I2C_EXT0_CLK_EN	0
6	SYSTEM_SPI2_CLK_EN	1
5	SYSTEM_UART1_CLK_EN	1
4	SYSTEM_I2S0_CLK_EN	0
3	SYSTEM_WDG_CLK_EN	1
2	SYSTEM_UART_CLK_EN	1
1	SYSTEM_SPI01_CLK_EN	1
0	SYSTEM_TIMERS_CLK_EN	1

SYSTEM_CPU_PERI_CLK_EN0_REG Configures this register to enable different peripheral clocks. For details, please refer to Table 87.

Register 15.18: SYSTEM_PERIP_CLK_EN1_REG (0x0044)

Bit	Field	Reset
31:7	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6	SYSTEM_CRYPTO_DMA_CLK_EN	0
5	SYSTEM_CRYPTO_HMAC_CLK_EN	0
4	SYSTEM_CRYPTO_DS_CLK_EN	0
3	SYSTEM_CRYPTO_RSA_CLK_EN	0
2	SYSTEM_CRYPTO_SHA_CLK_EN	0
1	SYSTEM_CRYPTO_AES_CLK_EN	0
0	(reserved)	0

SYSTEM_CRYPTO_AES_CLK_EN Set this bit to enable clock of cryptography AES. (R/W)

SYSTEM_CRYPTO_SHA_CLK_EN Set this bit to enable clock of cryptography SHA. (R/W)

SYSTEM_CRYPTO_RSA_CLK_EN Set this bit to enable clock of cryptography RSA. (R/W)

SYSTEM_CRYPTO_DS_CLK_EN Set this bit to enable clock of cryptography digital signature. (R/W)

SYSTEM_CRYPTO_HMAC_CLK_EN Set this bit to enable clock of cryptography HMAC. (R/W)

SYSTEM_CRYPTO_DMA_CLK_EN Set this bit to enable clock of cryptography DMA. (R/W)

Register 15.19: SYSTEM_PERIP_RST_EN0_REG (0x0048)

Bit	Field	Reset
31	SYSTEM_SPI4_RST	0
30	SYSTEM_ADC2_ARB_RST	0
29	SYSTEM_SYSTIMER_RST	0
28	SYSTEM_APB_SARADC_RST	0
27	SYSTEM_SPI3_DMA_RST	0
26	SYSTEM_PWM3_RST	0
25	SYSTEM_PWM2_RST	0
24	SYSTEM_UART_MEM_RST	0
23	SYSTEM_USB_RST	0
22	SYSTEM_SPI2_DMA_RST	0
21	SYSTEM_I2S1_RST	0
20	SYSTEM_PWM1_RST	0
19	SYSTEM_CAN_RST	0
18	SYSTEM_I2C_EXT1_RST	0
17	SYSTEM_PWM0_RST	0
16	SYSTEM_SPI3_RST	0
15	SYSTEM_TIMERGROUP1_RST	0
14	SYSTEM_EFUSE_RST	0
13	SYSTEM_TIMERGROUP_RST	0
12	SYSTEM_UHCI1_RST	0
11	SYSTEM_LEDC_RST	0
10	SYSTEM_PCNT_RST	0
9	SYSTEM_RMT_RST	0
8	SYSTEM_UHCI0_RST	0
7	SYSTEM_I2C_EXT0_RST	0
6	SYSTEM_SPI2_RST	0
5	SYSTEM_UART1_RST	0
4	SYSTEM_I2S0_RST	0
3	SYSTEM_WDG_RST	0
2	SYSTEM_UART_RST	0
1	SYSTEM_SPI01_RST	0
0	SYSTEM_TIMERS_RST	0

SYSTEM_PERIP_RST_EN0_REG Configures this register to reset different peripherals. For details, please refer to Table 87.

Register 15.20: SYSTEM_PERIP_RST_EN1_REG (0x004C)

Bit	Field	Reset
31:7	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6	SYSTEM_CRYPTO_DMA_RST	1
5	SYSTEM_CRYPTO_HMAC_RST	1
4	SYSTEM_CRYPTO_DS_RST	1
3	SYSTEM_CRYPTO_RSA_RST	1
2	SYSTEM_CRYPTO_SHA_RST	1
1	SYSTEM_CRYPTO_AES_RST	1
0	(reserved)	0

SYSTEM_PERIP_RST_EN1_REG Configures this register to reset different accelerators. For details, please refer to Table 87.

Register 15.21: SYSTEM_BT_LPCK_DIV_FRAC_REG (0x0054)

(reserved)	SYSTEM_LPCLK_RTC_EN	SYSTEM_LPCLK_SEL_XTAL32K	SYSTEM_LPCLK_SEL_XTAL	SYSTEM_LPCLK_SEL_8M	SYSTEM_LPCLK_SEL_RTC_SLOW	(reserved)	(reserved)	
31 29	28	27	26	25	24	23 12	11 0	
0 0 0	0	0	0	1	0	1	1	Reset

SYSTEM_LPCLK_SEL_RTC_SLOW Set this bit to select RTC slow clock as the low power clock. (R/W)

SYSTEM_LPCLK_SEL_8M Set this bit to select 8m clock as the low power clock. (R/W)

SYSTEM_LPCLK_SEL_XTAL Set this bit to select xtal clock as the low power clock. (R/W)

SYSTEM_LPCLK_SEL_XTAL32K Set this bit to select xtal32k clock as the low power clock. (R/W)

SYSTEM_LPCLK_RTC_EN Set this bit to enable the RTC low power clock. (R/W)

Register 15.22: SYSTEM_CPU_INTR_FROM_CPU_0_REG (0x0058)

(reserved)	SYSTEM_CPU_INTR_FROM_CPU_0	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SYSTEM_CPU_INTR_FROM_CPU_0 Set this bit to generate CPU interrupt 0. This bit needs to be reset by software in the ISR process. (R/W)

Register 15.23: SYSTEM_CPU_INTR_FROM_CPU_1_REG (0x005C)

(reserved)	SYSTEM_CPU_INTR_FROM_CPU_1	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SYSTEM_CPU_INTR_FROM_CPU_1 Set this bit to generate CPU interrupt 1. This bit needs to be reset by software in the ISR process. (R/W)

Register 15.24: SYSTEM_CPU_INTR_FROM_CPU_2_REG (0x0060)

(reserved)	SYSTEM_CPU_INTR_FROM_CPU_2	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SYSTEM_CPU_INTR_FROM_CPU_2 Set this bit to generate CPU interrupt 2. This bit needs to be reset by software in the ISR process. (R/W)

Register 15.25: SYSTEM_CPU_INTR_FROM_CPU_3_REG (0x0064)

(reserved)	SYSTEM_CPU_INTR_FROM_CPU_3	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SYSTEM_CPU_INTR_FROM_CPU_3 Set this bit to generate CPU interrupt 3. This bit needs to be reset by software in the ISR process. (R/W)

Register 15.26: SYSTEM_RSA_PD_CTRL_REG (0x0068)

(reserved)	SYSTEM_RSA_MEM_FORCE_PD	SYSTEM_RSA_MEM_FORCE_PU	SYSTEM_RSA_MEM_PD	
31 ... 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	1	Reset

SYSTEM_RSA_MEM_PD Set this bit to power down RSA memory. This bit has the lowest priority. When Digital Signature occupies the RSA, this bit is invalid. (R/W)

SYSTEM_RSA_MEM_FORCE_PU Set this bit to force power up RSA memory. This bit has the second highest priority. (R/W)

SYSTEM_RSA_MEM_FORCE_PD Set this bit to force power down RSA memory. This bit has the highest priority. (R/W)

Register 15.27: SYSTEM_BUSTOEXTMEM_ENA_REG (0x006C)

(reserved)	SYSTEM_BUSTOEXTMEM_ENA	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

SYSTEM_BUSTOEXTMEM_ENA Set this bit to enable bus to EDMA. (R/W)

Register 15.28: SYSTEM_CACHE_CONTROL_REG (0x0070)

(reserved)	SYSTEM_PRO_CACHE_RESET	SYSTEM_PRO_DCACHE_CLK_ON	SYSTEM_PRO_ICACHE_CLK_ON	
31 ... 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	1	Reset

SYSTEM_PRO_ICACHE_CLK_ON Set this bit to enable clock of i-cache. (R/W)

SYSTEM_PRO_DCACHE_CLK_ON Set this bit to enable clock of d-cache. (R/W)

SYSTEM_PRO_CACHE_RESET Set this bit to reset cache. (R/W)

Register 15.29: SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0074)

(reserved)	SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT	SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT	SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT	SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT	
31 … 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT Set this bit to enable Manual Encryption under SPI Boot mode. (R/W)

SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT Set this bit to enable Auto Encryption under Download Boot mode. (R/W)

SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT Set this bit to enable Auto Decryption under Download Boot mode. (R/W)

SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT Set this bit to enable Manual Encryption under Download Boot mode. (R/W)

Register 15.30: SYSTEM_RTC_FASTMEM_CONFIG_REG (0x0078)

SYSTEM_RTC_MEM_CRC_FINISH	SYSTEM_RTC_MEM_CRC_LEN	SYSTEM_RTC_MEM_CRC_ADDR	SYSTEM_RTC_MEM_CRC_START	(reserved)	
31	30 … 20	19 … 9	8	7 … 0	
0	0x7ff	0x0	0	0 0 0 0 0 0 0 0	Reset

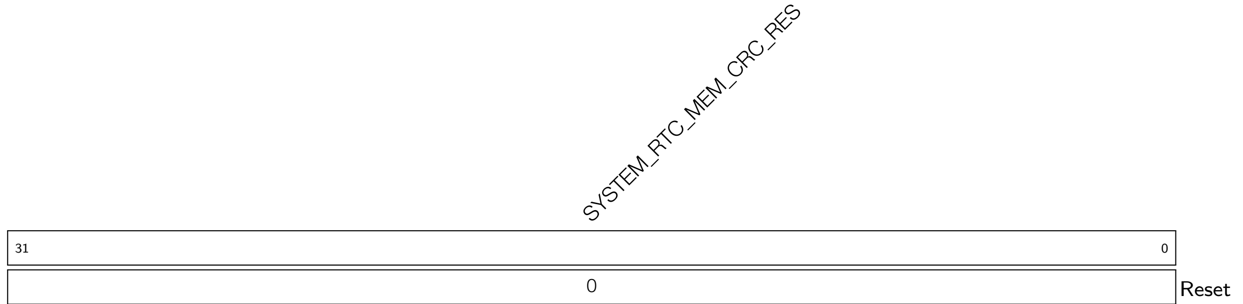
SYSTEM_RTC_MEM_CRC_START Set this bit to start the CRC of RTC memory. (R/W)

SYSTEM_RTC_MEM_CRC_ADDR This field is used to set address of RTC memory for CRC. (R/W)

SYSTEM_RTC_MEM_CRC_LEN This field is used to set length of RTC memory for CRC based on start address. (R/W)

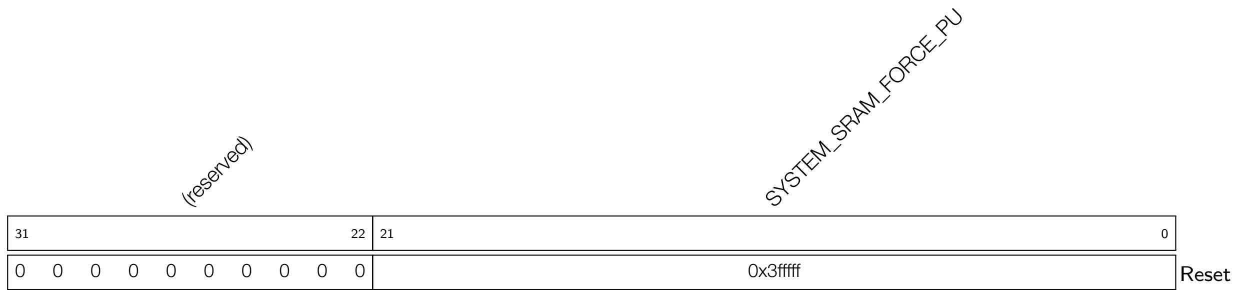
SYSTEM_RTC_MEM_CRC_FINISH This bit stores the status of RTC memory CRC. High level means finished while low level means not finished. (RO)

Register 15.31: SYSTEM_RTC_FASTMEM_CRC_REG (0x007C)



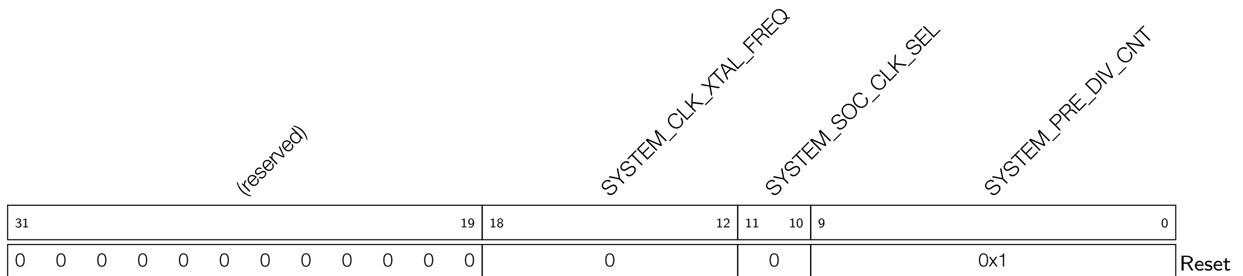
SYSTEM_RTC_MEM_CRC_RES This field stores the CRC result of RTC memory. (RO)

Register 15.32: SYSTEM_SRAM_CTRL_2_REG (0x0088)



SYSTEM_SRAM_FORCE_PU This field is used to power up internal SRAM. For details, please refer to Table 86. (R/W)

Register 15.33: SYSTEM_SYSCLK_CONF_REG (0x008C)

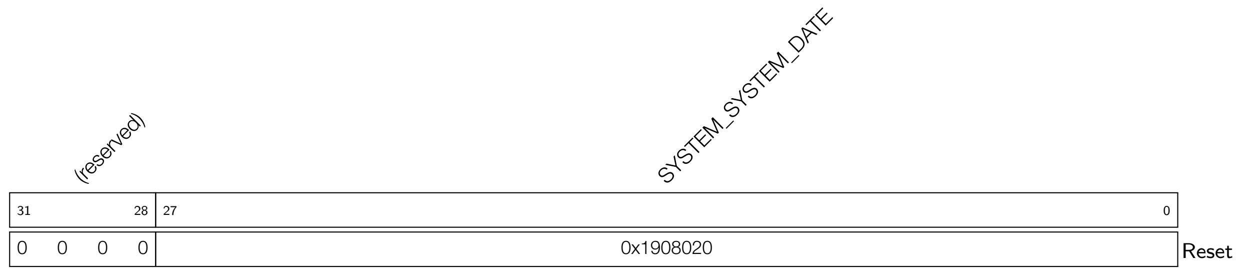


SYSTEM_PRE_DIV_CNT This field is used to set the count of prescaler of XTAL_CLK. For details, please refer to Table 48 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_SOC_CLK_SEL This field is used to select SOC clock. For details, please refer to Table 46 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_CLK_XTAL_FREQ This field is used to read XTAL frequency in MHz. (RO)

Register 15.34: SYSTEM_DATE_REG (0x0FFC)



SYSTEM_DATE Version control register. (R/W)

16. SHA Accelerator (SHA)

16.1 Introduction

ESP32-S2 integrates an SHA accelerator, which is a hardware device that speeds up SHA algorithm significantly, compared to SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-S2 has two working modes, which are Typical SHA and DMA-SHA.

16.2 Features

The following functionality is supported:

- All the hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/256
  - SHA-512/*t*
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleave function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

16.3 Working Modes

The SHA accelerator integrated in ESP32-S2 has two working modes: Typical SHA and DMA-SHA.

- Typical SHA Working Mode: all the data is written and read via CPU directly.
- DMA-SHA Working Mode: all the data is read via crypto DMA. That is, users can configure the DMA controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

Users can start the SHA accelerator with different working modes by configuring registers SHA_START_REG and SHA_DMA_START_REG. For details, please see Table 90.

Table 90: SHA Accelerator Working Mode

Working Mode	Configuration Method
Typical SHA	Set SHA_START_REG to 1
DMA-SHA	Set SHA_DMA_START_REG to 1

Users can choose hash algorithms by configuring the SHA_MODE_REG register. For details, please see Table 91.

Table 91: SHA Hash Algorithm

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/t	7

Notice:
ESP32-S2's Digital Signature (DS) and HMAC Accelerator (HMAC) modules also call the SHA accelerator. Therefore, users cannot access the SHA accelerator when these modules are working.

16.4 Function Description

SHA accelerator can generate the message digest via two steps: Preprocessing and Hash operation.

16.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

16.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 or 1024 bits, depending on the algorithm. Thus, all the messages should be padded to a multiple of 512 or 1024 bits before the hash computation.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

- **SHA-1**, **SHA-224** and **SHA-256**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
  3. Last, append the 64-bit block that is equal to the number m expressed using a binary representation.

- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/**t
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896\ mod\ 1024$;
  3. Last, append the 128-bit block that is equal to the number m expressed using a binary representation.

For more details, please refer to Section "5.1 Padding the Message" in FIPS PUB 180-4 Spec.

16.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit or 1024-bit blocks.

- For **SHA-1**, **SHA-224** and **SHA-256**: the message and its padding are parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.
- For **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/**t: the message and its padding are parsed into N 1024-bit blocks. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

In Typical SHA working mode, all the message blocks are written into the SHA_M_*n*_REG, following the rules below:

- For **SHA-1, SHA-224 and SHA-256**: $M_0^{(i)}$ is stored in SHA_M_0_REG, $M_1^{(i)}$ stored in SHA_M_1_REG, ..., and $M_{15}^{(i)}$ stored in SHA_M_15_REG.
- For **SHA-384, SHA-512, SHA-512/224 and SHA-512/256**: the most significant 32 bits and the least significant 32 bits of $M_0^{(i)}$ are stored in SHA_M_0_REG and SHA_M_1_REG, respectively, ..., the most significant 32 bits and the least significant 32 bits of $M_{15}^{(i)}$ are stored in SHA_M_30_REG and SHA_M_31_REG, respectively.

> **Note:**
> For more information about "message block", please refer to Section "2.1 Glossary of Terms and Acronyms" in FIPS PUB 180-4 Spec.

In DMA-SHA working mode, please complete the following configuration:

1. Create an external linked list;
2. Configure this linked list based on the instruction described in Chapter 2 *DMA Controller (DMA)*, including but not limited to assigning the starting address of the input message to the buffer address pointer of the linked list;
3. Configure the CRYPTO_DMA_OUTLINK_ADDR to the first out-link linked list;
4. Write 1 to register CRYPTO_DMA_OUTLINK_START, so the DMA starts to move data;
5. Write 1 to register CRYPTO_DMA_AES_SHA_SELECT_REG, so the SHA accelerator gets to use the DMA resource shared by AES and SHA accelerators.

16.4.1.3 Initial Hash Value

Before hash computation begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t_length = 7'h48$ and t_string is padded in the following format:

$8'h3t_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and t_string = $32'h38800000$.

   - If $10 <= t <= 99$, then $t_length = 7'h50$ and t_string is padded in the following format:

$8'h3t_1$	$8'h3t_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and t_string = $32'h35368000$.

   - If $100 <= t < 512$, then $t_length = 7'h58$ and t_string is padded in the following format:

$8'h3t_2$	$8'h3t_1$	$8'h3t_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and t_string = $32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, that is performing SHA-512 operation (with its initial hash value set to the result of 8-bitwise XOR operation of C and 0xa5) from the hexadecimal representation of the string "SHA-512/t".

16.4.2 Hash Computation Process

After the preprocessing, the ESP32-S2 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP32-S2 SHA accelerator supports two working modes, which are Typical SHA and DMA-SHA. The operation process for the SHA accelerator under two working modes is described in the following subsections.

16.4.2.1 Typical SHA Process

ESP32-S2 SHA accelerator supports "interleave" functionality when working under Typical SHA mode:

- **Type "alone"**: Users do not insert any new computation before the SHA accelerator completes all the message blocks.
- **Type "interleave"**: Users can insert new computations (both Typical SHA task and DMA-SHA task) every time the SHA accelerator completes one message block. To be more specific, users can store the message digest in registers SHA_H_*n*_REG after completing each message block, and assign the accelerator with other higher priority tasks. After the inserted task completes, users can put the message digest stored back to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused computation.

Typical SHA Process (except for SHA-512/t)

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 91.
2. Process the current message block.
   (a) Write the current message block in registers SHA_M_*n*_REG;
   (b) Start the SHA accelerator [1]:
      - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the computation;
      - If this is not the first time to execute this step, set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start computation.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the computation for the current message block and now is in the "idle" status. Then, go to step 3.
3. Decide if you want to insert other computations.
   - If yes, please get ready for handing over the SHA accelerator to the new task:
      (a) Read and store the hash algorithm selected for the current computation stored in the SHA_MODE_REG register;
      (b) Read and store the message digest stored in registers SHA_H_*n*_REG;
      (c) Last, please go to perform the inserted computation. For the detailed process of the inserted computation, please refer to Typical SHA or DMA-SHA, depending on the working mode.

- Otherwise, please continue to execute Step 4.

4. Decide if you have more message blocks following the previous computation:
   - If yes, please go back to 2.
   - Otherwise, go to Step 5.

5. Decide if you need to return the SHA accelerator to a previous computation (i.e., decide whether the current task is an inserted task or not):
   - If yes, please get ready to return the SHA accelerator for the previous computation:
     (a) Write the previously stored hash algorithm back to register SHA_MODE_REG;
     (b) Write the previously stored message digest back to registers SHA_H_*n*_REG;
     (c) Then, go to Step 2.
   - Otherwise, there is no need to return SHA Control. Therefore, please go to Step 6 directly.

6. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/*t*)

1. Select a hash algorithm.
   - Select SHA-512/*t* algorithm by configuring the SHA_MODE_REG register to 7.

2. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 16.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

3. Process the current message block.
   (a) Write the current message block in registers SHA_M_*n*_REG;
   (b) Start the SHA accelerator [1]:
     - Set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start computation.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the computation for the current message block and now is in the "idle" status. Then, go to step 4.

4. Decide if you want to insert other computations.
   - If yes, please get ready for handing over the SHA accelerator to the new task:
     (a) Read and store the hash algorithm selected for the current computation stored in the SHA_MODE_REG register;
     (b) Read and store the message digest stored in registers SHA_H_*n*_REG;

(c) Last, please go to perform the inserted computation. For the detailed process of the inserted computation, please refer to Typical SHA or DMA-SHA, depending on the working mode.

- Otherwise, please continue to execute Step 5.

5. Decide if you have more message blocks following the previous computation:
   - If yes, please go back to 3.
   - Otherwise, go to Step 6.
6. Decide if you need to return the SHA accelerator to a previous computation (i.e., decide whether the current task is an inserted task or not):
   - If yes, please get ready to return the SHA accelerator for the previous computation:
     (a) Write the previously stored hash algorithm back to register SHA_MODE_REG;
     (b) Write the previously stored message digest back to registers SHA_H_*n*_REG;
     (c) Then, go to Step 3.
   - Otherwise, there is no need to return SHA Control. Therefore, please go to Step 7 directly.
7. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

Note:

1. In Step 2b, the software can also write the next message block (to be processed) in registers SHA_M_*n*_REG, if any, while the hardware starts SHA computation, to save time.

16.4.2.2 DMA-SHA Process

ESP32-S2 SHA accelerator does not support type “interleave” computation, which means you cannot insert new computation before the whole DMA-SHA process completes. In this mode, users who need task insertion are recommended to divide your message blocks and perform several DMA-SHA computations, instead of trying to compute all the messages in one go.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via crypto DMA.

Therefore, users are required to configure the DMA controller as instructed in Subsection 16.4.1.2. Please refer to Chapter 2 *DMA Controller (DMA)* for more information.

DMA-SHA process (except SHA-512/t)

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 91.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.

4. Start the DMA-SHA computation.
   - If the current DMA-SHA computation follows a previous computation, firstly write the message digest from the previous computation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA computation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
6. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/t

1. Select a hash algorithm.
   - Select SHA-512/t algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
   (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 16.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
4. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
5. Start the DMA-SHA computation.
   - Write 1 to register SHA_DMA_CONTINUE_REG to start the accelerator.
6. Wait till the completion of the DMA-SHA computation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
7. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

16.4.3 Message Digest

After the hash computation completes, the SHA accelerator writes the message digest from the computation to registers SHA_H_*n*_REG(*n*: 0~15). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 95 below:

Table 95: The Storage and Length of Message digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage [1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t* [2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG

Note:

1. The message digest are stored in registers from most significant bits to the least significant bits, with the first word stored in register SHA_H_0_REG and the second word stored in register SHA_H_1_REG... For details, please see subsection 16.4.1.2.
2. The registers used for SHA-512/*t* algorithm depend on the value of t. *x*+1 indicates the number of 32-bit registers used to store *t* bits of message digest, so that *x* = $roundup(t/32)$-1. For example:
   - When *t* = 8, then *x* = 0, indicating that the 8-bit long message digest is stored in the most significant 8 bits of register SHA_H_0_REG;
   - When *t* = 32, then *x* = 0, indicating that the 32-bit long message digest is stored in register SHA_H_0_REG;
   - When *t* = 132, then *x* = 4, indicating that the 132-bit long message digest is stored in registers SHA_H_0_REG, SHA_H_1_REG, SHA_H_2_REG, SHA_H_3_REG, and SHA_H_4_REG.

16.4.4 Interrupt

SHA accelerator supports interrupt on the completion of computation when working in the DMA-SHA mode. To enable this function, write 1 to register SHA_INT_ENA_REG. Note that the interrupt should be cleared by software after use via setting the SHA_INT_CLEAR_REG register to 1.

16.5 Base Address

Users can access SHA with two base addresses, which can be seen in Table 96. For more information about accessing peripherals from different buses, please see Chapter 3 *System and Memory*.

Table 96: SHA Accelerator Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43B000
PeriBUS2	0x6003B000

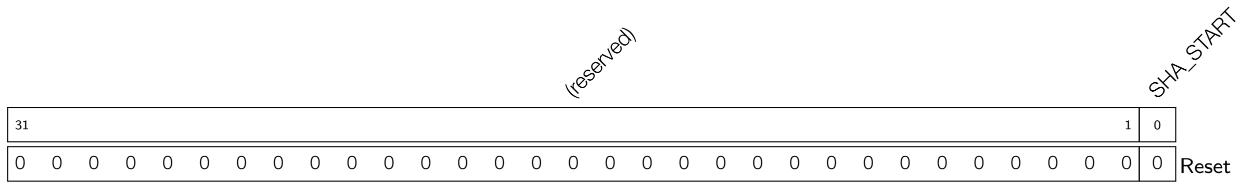
16.6 Register Summary

Name	Description	Address	Access
Control/Status registers			
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_BUSY_REG	Indicates if SHA Accelerator is busy or not	0x0018	RO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
SHA_DMA_CONTINUE_REG	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
SHA_INT_CLEAR_REG	DMA-SHA interrupt clear register	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA interrupt enable register	0x0028	R/W
Version Register			
SHA_DATE_REG	Version control register	0x002C	R/W
Configuration Registers			
SHA_MODE_REG	Defines the algorithm of SHA accelerator	0x0000	R/W
SHA_T_STRING_REG	String content register for calculating initial Hash Value (only effective for SHA-512/t)	0x0004	R/W
SHA_T_LENGTH_REG	String length register for calculating initial Hash Value (only effective for SHA-512/t)	0x0008	R/W
Memories			
SHA_DMA_BLOCK_NUM_REG	Block number register (only effective for DMA-SHA)	0x000C	R/W
SHA_H_0_REG	Hash value	0x0040	R/W
SHA_H_1_REG	Hash value	0x0044	R/W
SHA_H_2_REG	Hash value	0x0048	R/W
SHA_H_3_REG	Hash value	0x004C	R/W
SHA_H_4_REG	Hash value	0x0050	R/W
SHA_H_5_REG	Hash value	0x0054	R/W
SHA_H_6_REG	Hash value	0x0058	R/W
SHA_H_7_REG	Hash value	0x005C	R/W
SHA_H_8_REG	Hash value	0x0060	R/W
SHA_H_9_REG	Hash value	0x0064	R/W

Name	Description	Address	Access
SHA_H_10_REG	Hash value	0x0068	R/W
SHA_H_11_REG	Hash value	0x006C	R/W
SHA_H_12_REG	Hash value	0x0070	R/W
SHA_H_13_REG	Hash value	0x0074	R/W
SHA_H_14_REG	Hash value	0x0078	R/W
SHA_H_15_REG	Hash value	0x007C	R/W
SHA_M_0_REG	Message	0x0080	R/W
SHA_M_1_REG	Message	0x0084	R/W
SHA_M_2_REG	Message	0x0088	R/W
SHA_M_3_REG	Message	0x008C	R/W
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	0x00AC	R/W
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W
SHA_M_16_REG	Message	0x00C0	R/W
SHA_M_17_REG	Message	0x00C4	R/W
SHA_M_18_REG	Message	0x00C8	R/W
SHA_M_19_REG	Message	0x00CC	R/W
SHA_M_20_REG	Message	0x00D0	R/W
SHA_M_21_REG	Message	0x00D4	R/W
SHA_M_22_REG	Message	0x00D8	R/W
SHA_M_23_REG	Message	0x00DC	R/W
SHA_M_24_REG	Message	0x00E0	R/W
SHA_M_25_REG	Message	0x00E4	R/W
SHA_M_26_REG	Message	0x00E8	R/W
SHA_M_27_REG	Message	0x00EC	R/W
SHA_M_28_REG	Message	0x00F0	R/W
SHA_M_29_REG	Message	0x00F4	R/W
SHA_M_30_REG	Message	0x00F8	R/W
SHA_M_31_REG	Message	0x00FC	R/W

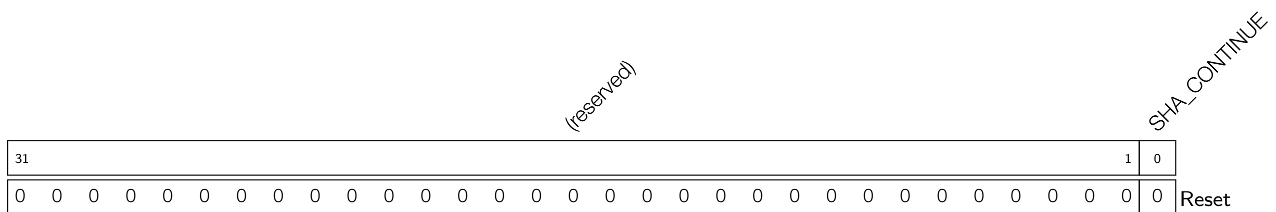
16.7 Registers

Register 16.1: SHA_START_REG (0x0010)



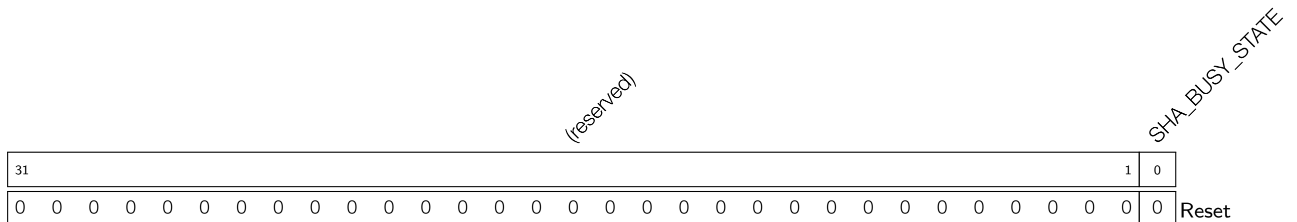
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 16.2: SHA_CONTINUE_REG (0x0014)



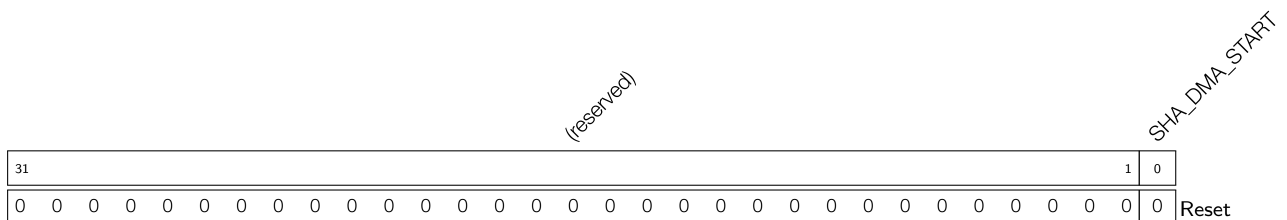
SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 16.3: SHA_BUSY_REG (0x0018)



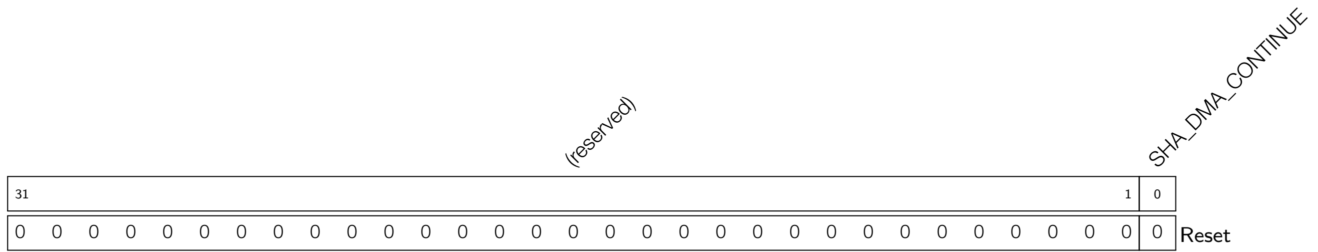
SHA_BUSY_STATE Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

Register 16.4: SHA_DMA_START_REG (0x001C)



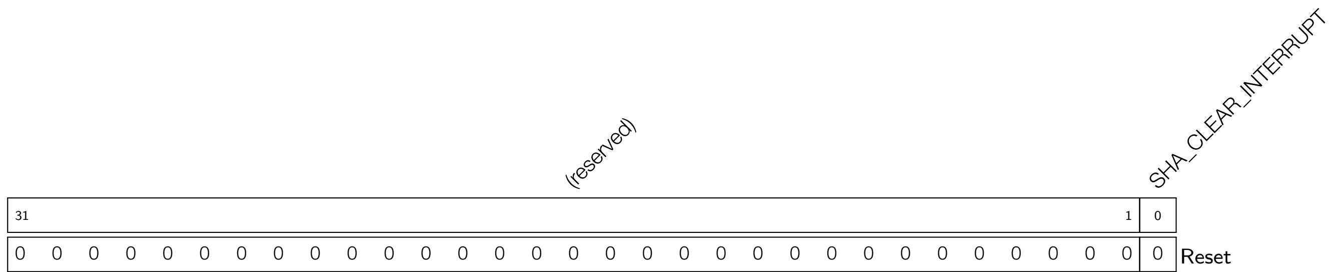
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 16.5: SHA_DMA_CONTINUE_REG (0x0020)



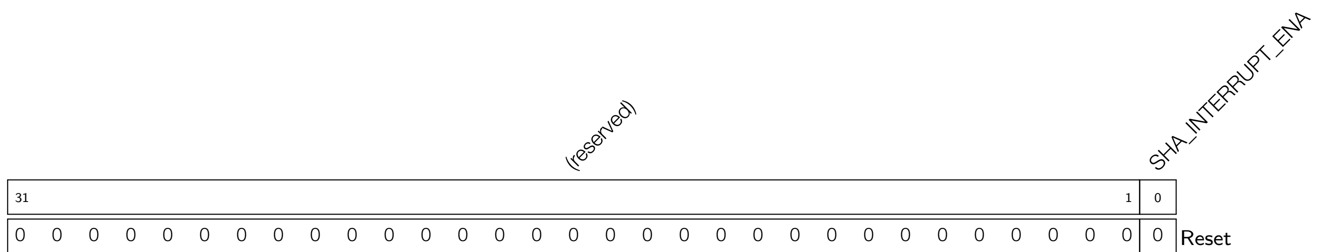
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 16.6: SHA_INT_CLEAR_REG (0x0024)



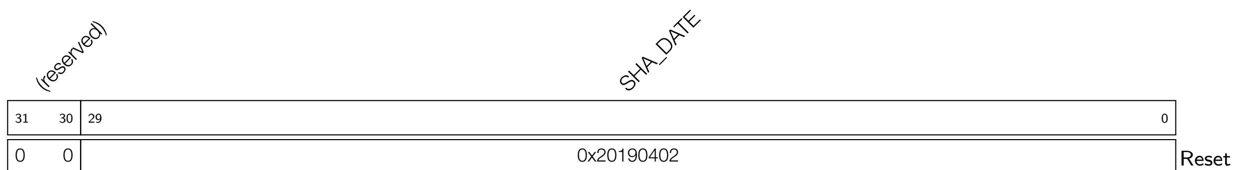
SHA_CLEAR_INTERRUPT Clears DMA-SHA interrupt. (WO)

Register 16.7: SHA_INT_ENA_REG (0x0028)



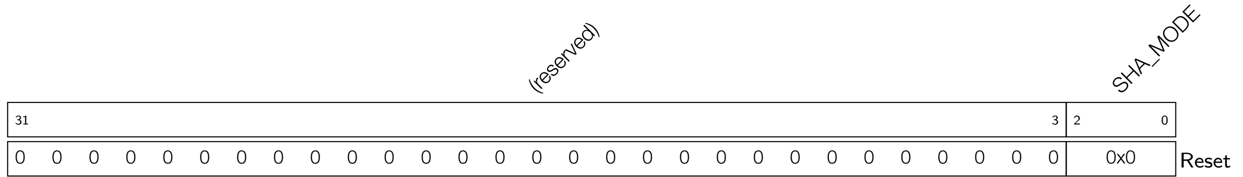
SHA_INTERRUPT_ENA Enables DMA-SHA interrupt. (R/W)

Register 16.8: SHA_DATE_REG (0x002C)



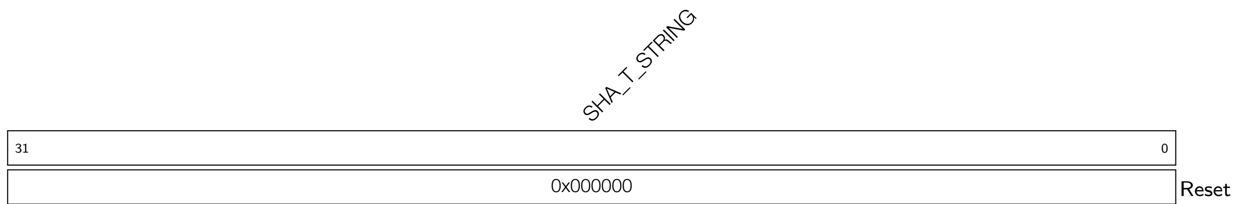
SHA_DATE Version control register. (R/W)

Register 16.9: SHA_MODE_REG (0x0000)



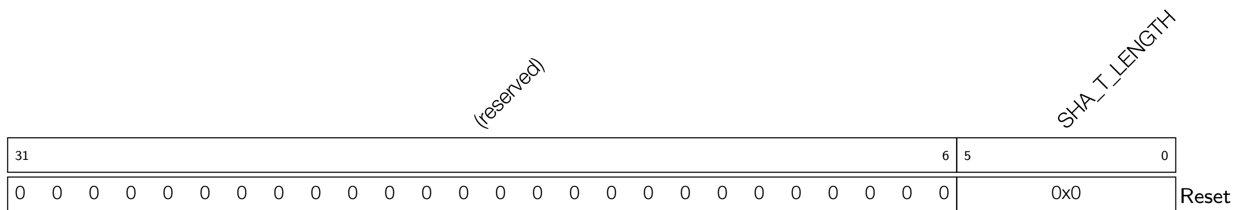
SHA_MODE Defines the SHA algorithm. For details, please see Table 91. (R/W)

Register 16.10: SHA_T_STRING_REG (0x0004)



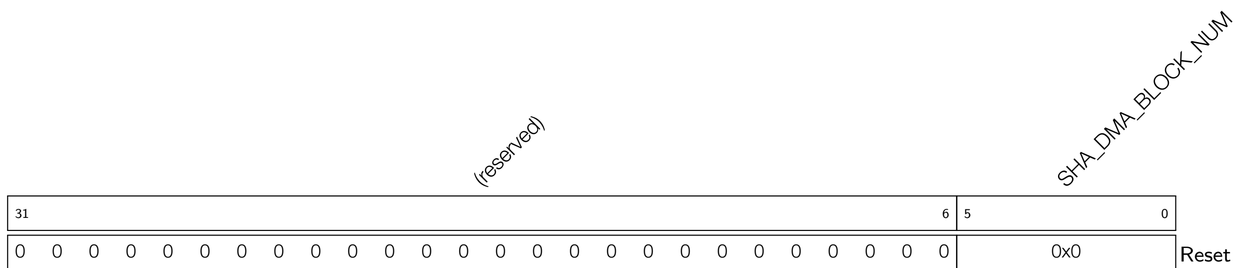
SHA_T_STRING Defines t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 16.11: SHA_T_LENGTH_REG (0x0008)



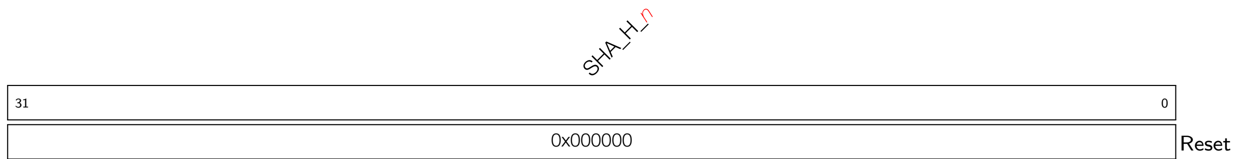
SHA_T_LENGTH Defines t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 16.12: SHA_DMA_BLOCK_NUM_REG (0x000C)



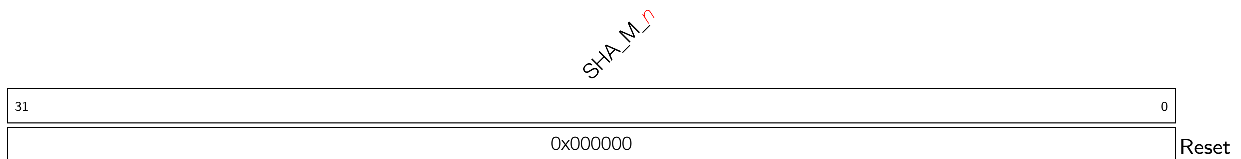
SHA_DMA_BLOCK_NUM Defines the DMA-SHA block number. (R/W)

Register 16.13: SHA_H_*n*_REG (*n*: 0-15) (0x0040+4n*)**



SHA_H_*n* Stores the *n*th 32-bit piece of the Hash value. (R/W)

Register 16.14: SHA_M_*n*_REG (*n*: 0-31) (0x0080+4n*)**



SHA_M_*n* Stores the *n*th 32-bit piece of the message. (R/W)

17. AES Accelerator (AES)

17.1 Introduction

ESP32-S2 integrates an Advanced Encryption Standard (AES) Accelerator, which is a hardware device that speeds up AES Algorithm significantly, compared to AES algorithms implemented solely in software. The AES Accelerator integrated in ESP32-S2 has two working modes, which are Typical AES and DMA-AES.

17.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-192/AES-256 encryption and decryption
  - Four variations of key endianness and four variations of text endianness
- DMA-AES working mode
  - Block mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - GCM (Galois/Counter Mode)
  - Interrupt on completion of computation

17.3 Working Modes

The AES Accelerator integrated in ESP32-S2 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode: supports AES-128/AES-192/AES-256 encryption and decryption under NIST FIPS 197. In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode: supports block cipher algorithms ECB/CBC/OFB/CTR/CFB8/CFB128 under NIST SP 800-38A, and GCM mode of operation under NIST SP 800-38D. In this working mode, the plaintext and ciphertext is written and read via crypto DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 98 below.

Table 98: AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode
0	Typical AES
1	DMA-AES

For detailed introduction on these two working modes, please refer to Section 17.4 and Section 17.5 below.

Notice:
ESP32-S2's Digital Signature (DS) and External Memory Manual Encryption modules also call the AES accelerator. Therefore, users cannot access the AES accelerator when these modules are working.

17.4 Typical AES Working Mode

In the Typical AES working mode, the AES accelerator is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data, i.e. AES-128/AES-192/AES-256 encryption and decryption. Users can choose the operation type for AES accelerator working in Typical AES working mode by configuring the AES_MODE_REG register according to Table 99 below.

Table 99: Operation Type under Typical AES Working Mode

AES_MODE_REG[2:0]	Operation Type
0	AES-128 encryption
1	AES-192 encryption
2	AES-256 encryption
4	AES-128 decryption
5	AES-192 decryption
6	AES-256 decryption

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 100 below.

Table 100: Working Status under Typical AES Working Mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle or completed operation.
1	WORK	The AES accelerator is in the middle of an operation.

In the Typical AES working mode, the AES accelerator requires 11 ~ 15 clock cycles to encrypt a message block, and 21 or 22 clock cycles to decrypt a message block.

17.4.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.

- For AES-192 encryption/decryption, the 192-bit key is stored in AES_KEY_0_REG ~ AES_KEY_5_REG.
- For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_m_REG and AES_TEXT_OUT_m_REG, which are two sets of four 32-bit registers.

- For AES-128/AES-192/AES-256 encryption, the AES_TEXT_IN_m_REG registers are initialized with plaintext. Then, the AES Accelerator stores the ciphertext into AES_TEXT_OUT_m_REG after operation.
- For AES-128/AES-192/AES-256 decryption, the AES_TEXT_IN_m_REG registers are initialized with ciphertext. Then, the AES Accelerator stores the plaintext into AES_TEXT_OUT_m_REG after operation.

17.4.2 Endianness

Text Endianness

In Typical AES working mode, the AES Accelerator uses cryptograohic keys to encrypt and decrypt data in blocks of 128 bits. The Bit 2 and Bit 3 of the AES_ENDIAN_REG register define the endianness of input text, while the Bit 4 and Bit 5 define the endianness of output text. To be more specific, Bit 2 and Bit 4 control how the four bytes are stored in each word, and Bit 3 and Bit 5 control how the four words are stored in each message block.

Users can choose one of the four text endianness types provided by the AES Accelerator by configuring the AES_ENDIAN_REG register. Details can been seen in Table 101.

Table 101: Text Endianness Types for Typical AES

Word Endian Controlling Bit	Byte Endian Controlling Bit	Plaintext/Ciphertext[2]					
0	0	State[1]		c			
				0	1	2	3
		r	0	AES_TEXT_x_3_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_0_REG[31:24]
			1	AES_TEXT_x_3_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_0_REG[23:16]
			2	AES_TEXT_x_3_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_0_REG[15:8]
			3	AES_TEXT_x_3_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_0_REG[7:0]
0	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_x_3_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_0_REG[7:0]
			1	AES_TEXT_x_3_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_0_REG[15:8]
			2	AES_TEXT_x_3_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_0_REG[23:16]
			3	AES_TEXT_x_3_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_0_REG[31:24]
1	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_x_0_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_3_REG[31:24]
			1	AES_TEXT_x_0_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_3_REG[23:16]
			2	AES_TEXT_x_0_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_3_REG[15:8]
			3	AES_TEXT_x_0_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_3_REG[7:0]
1	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_x_0_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_3_REG[7:0]
			1	AES_TEXT_x_0_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_3_REG[15:8]
			2	AES_TEXT_x_0_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_3_REG[23:16]
			3	AES_TEXT_x_0_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_3_REG[31:24]

Note:

1. The definition of "State" is described in Section 3.4 The State in NIST FIPS 197.
2. Where,
   - When *x* = IN, the Word Endian and Byte Endian controlling bits of AES_TEXT_IN_*m*_REG are the Bit 2 and Bit 3 of AES_ENDIAN_REG, respectively;
   - When *x* = OUT, the Word Endian and Byte Endian controlling bits of AES_TEXT_OUT_*m*_REG are the Bit 4 and Bit 5 of AES_ENDIAN_REG, respectively.

Key Endianness

In Typical AES working mode, Bit 0 and bit 1 in AES_ENDIAN_REG define the key endianness.

Users can choose one of the four key endianness types provided by the AES accelerator by configuring the AES_ENDIAN_REG register. Details can been seen in Table 102, Table 103, and Table 104.

Table 102: Key Endianness Types for AES-128 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3][1]
0	0	[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

Note:

1. w[0] ~ w[3] are "the first Nk words of the expanded key" as specified in Section 5.2 Key Expansion in NIST FIPS 197.
2. "Column Bit" specifies the bytes of each word stored in w[0] ~ w[3].

Table 103: Key Endianness Types for AES-192 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5][1]
0	0	[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]

Note:

1. w[0] ~ w[5] are "the first Nk words of the expanded key" as specified in Chapter 5.2 Key Expansion in NIST FIPS 197.
2. "Column Bit" specifies the bytes of each word stored in w[0] ~ w[5].

Table 104: Key Endianness Types for AES-256 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][1]
0	0	[31:24]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

Note:

1. w[0] ~ w[7] are "the first Nk words of the expanded key" as specified in Chapter 5.2 Key Expansion in NIST FIPS 197.
2. "Column Bit" specifies the bytes of each word stored in w[0] ~ w[7].

17.4.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_IN_*m*_REG, and AES_ENDIAN_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

17.5 DMA-AES Working Mode

In the DMA-AES working mode, the AES accelerator supports six block operations including ECB/CBC/OFB/CTR/CFB8/CFB128 as well as GCM operation. Users can choose the operation type for AES accelerator working in the DMA-AES working mode by configuring the AES_BLOCK_MODE_REG register according to Table 105 below.

Table 105: Operation Type under DMA-AES Working Mode

AES_BLOCK_MODE_REG[2:0]	Operation Type
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	GCM (Galois/Counter Mode)

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 106 below.

Table 106: Working Status under DMA-AES Working mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle.
1	WORK	The AES accelerator is in the middle of an operation.
2	DONE	The AES accelerator completed operations.

When working in the DMA-AES working mode, the AES accelerator supports interrupt on the completion of computation. To enable this function, write 1 to the AES_INT_ENA_REG register. By default, the interrupt function is enabled. Also, note that the interrupt should be cleared by software after use.

17.5.1 Key, Plaintext, and Cipertext

Block Operation

During the block operations, the AES Accelerator reads source data (in_stream) from DMA, and write result data (out_stream) to DMA after the computation.

- For encryption, DMA reads plaintext from memory, then passes it to AES as source data. After computation, AES passes ciphertext as result data back to DMA to write into memory.
- For decryption, DMA reads ciphertext from memory, then passes it to AES as source data. After computation, AES passes plaintext as result data back to DMA to write into memory.

During block operations, the lengths of the source data and result data are the same. The total computation time is reduced because the DMA data operation and AES computation can happen concurrently.

The length of source data for AES Accelerator under DMA-AES working mode must be 128 bits or the integral multiples of 128 bits. Otherwise, trailing zeros will be added to the original source data, so the length of source data equals to the nearest integral multiples of 128 bits. Please see details in Table 107 below.

Table 107: TEXT-PADDING

<table>
<tr><th>Function : TEXT-PADDING( )</th></tr>
<tr><td>Input : X, bit string.
Output : Y = TEXT-PADDING(X), whose length is the nearest integral multiples of 128 bits.</td></tr>
<tr><td>Steps
Let us assume that X is a data-stream that can be split into n parts as following:
$X = X_1||X_2||\cdots||X_{n-1}||X_n$
Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is t (0<=t<=127).
If $t = 0$, then
TEXT-PADDING(X) = X;
If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n||0^{128-t}$, then
TEXT-PADDING(X) = $X_1||X_2||\cdots||X_{n-1}||X_n^*$ = $X||0^{128-t}$</td></tr>
</table>

17.5.2 Endianness

Under the DMA-AES working mode, the transmission of source data and result data for AES Accelerator is solely controlled by DMA. Therefore, the AES Accelerator cannot control the Endianness of the source data and result data, but does have requirement on how these data should be stored in memory and on the length of the data.

For example, let us assume DMA needs to write the following data into memory at address 0x0280.

- Data represented in hexadecimal:
  - 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 108 below.

Table 108: Text Endianness for DMA-AES

Address	Byte	Address	Byte	Address	Byte	Address	Byte
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

DMA can access both internal memory and PSRAM outside ESP32-S2. When you use DMA to access external PSRAM, please use base addresses that meet the requirements for DMA. When you use DMA to access internal memory, base addresses do not have such requirements. Details can be found in Chapter 2 *DMA Controller (DMA)*.

17.5.3 Standard Incrementing Function

AES accelerator provides two Standard Incrementing Functions for the CTR block operation, which are INC_{32} and INC_{128} Standard Incrementing Functions. By setting the AES_INC_SEL_REG register to 0 or 1, users can choose the INC_{32} or INC_{128} functions respectively. For details on the Standard Incrementing Function, please see Chapter B.1 The Standard Incrementing Function in NIST SP 800-38A.

17.5.4 Block Number

Register AES_BLOCK_NUM_REG stores the Block Number of plaintext P or cipertext C. The length of this register equals to length(**TEXT-PADDING**(P))/128 or length(**TEXT-PADDING**(C))/128. The AES Accelerator only uses this register when working in the DMA-AES mode.

17.5.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES Accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 $\cdots$ Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 108, i.e. the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

For more details on IV and ICB, please refer to NIST SP 800-38A.

17.5.6 Block Operation Process

1. Write 0 to the CRYPTO_DMA_AES_SHA_SELECT_REG register.
2. Configure Crypto DMA chained list and start DMA. For details, please refer to Chapter 2 *DMA Controller (DMA)*.
3. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG.
   - Select operation type by configuring the AES_BLOCK_MODE_REG register. For details, see Table 105.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 17.5.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES Accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (This is always needed except for ECB block operation).
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.
6. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 2 *DMA*

Controller (DMA).

7. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occured during the computation.

8. Release the AES Accelerator by writing 0 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 5 is completed.

17.5.7 GCM Operation Process

1. Write 0 to the CRYPTO_DMA_AES_SHA_SELECT_REG register.

2. Configure Crypto DMA chained list and start DMA. For details on DMA, please refer to Chapter 2 *DMA Controller (DMA)*.

3. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG AES_ENDIAN_REG.
   - Write 6 to the AES_BLOCK_MODE_REG register.
   - Initialize the AES_BLOCK_NUM_REG register. Details about this register are described in Section 17.5.4.
   - Initialize the AES_AAD_BLOCK_NUM_REG register. Details about this register are described in Section 17.6.4.
   - Initialize the AES_REMAINDER_BIT_NUM_REG register. Details about this register is described in Section 17.6.5.

4. Start operation by writing 1 to the AES_TRIGGER_REG register.

5. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2. For details on the working status of AES Accelerator, please refer to Table 106. At this step, no interrupt occurs.

6. Obtain the H value from the AES_H_MEM memory.

7. Generate J_0 and write it to the AES_J0_MEM memory.

8. Continue operating by writing 1 to the AES_CONTINUE_REG register.

9. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs. For details on the working status of AES Accelerator, please refer to Table 106.

10. Obtain T_0 by reading AES_T0_MEM, which is already ready at this step.

11. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 2 *DMA Controller (DMA)*.

12. Clear interrupt by writting 1 to the AES_INT_CLR_REG register, if any AES interrupt occured during the computation.

13. Exit DMA by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG becomes 0. Note that, you can exit DMA earlier, but only after Step 9 is completed.

17.6 GCM Algorithm

ESP32-S2's AES accelerator fully supports GCM Algorithm. In reality, the AAD, C and P that are longer than 2^{32}-1 bits are seldom used. Therefore, we specify that the length of AAD, C and P should be no longer than 2^{32}-1 here. Figure 17-1 below demonstrates how GCM encryption is implemented in the AES Accelerator of ESP32-S2.

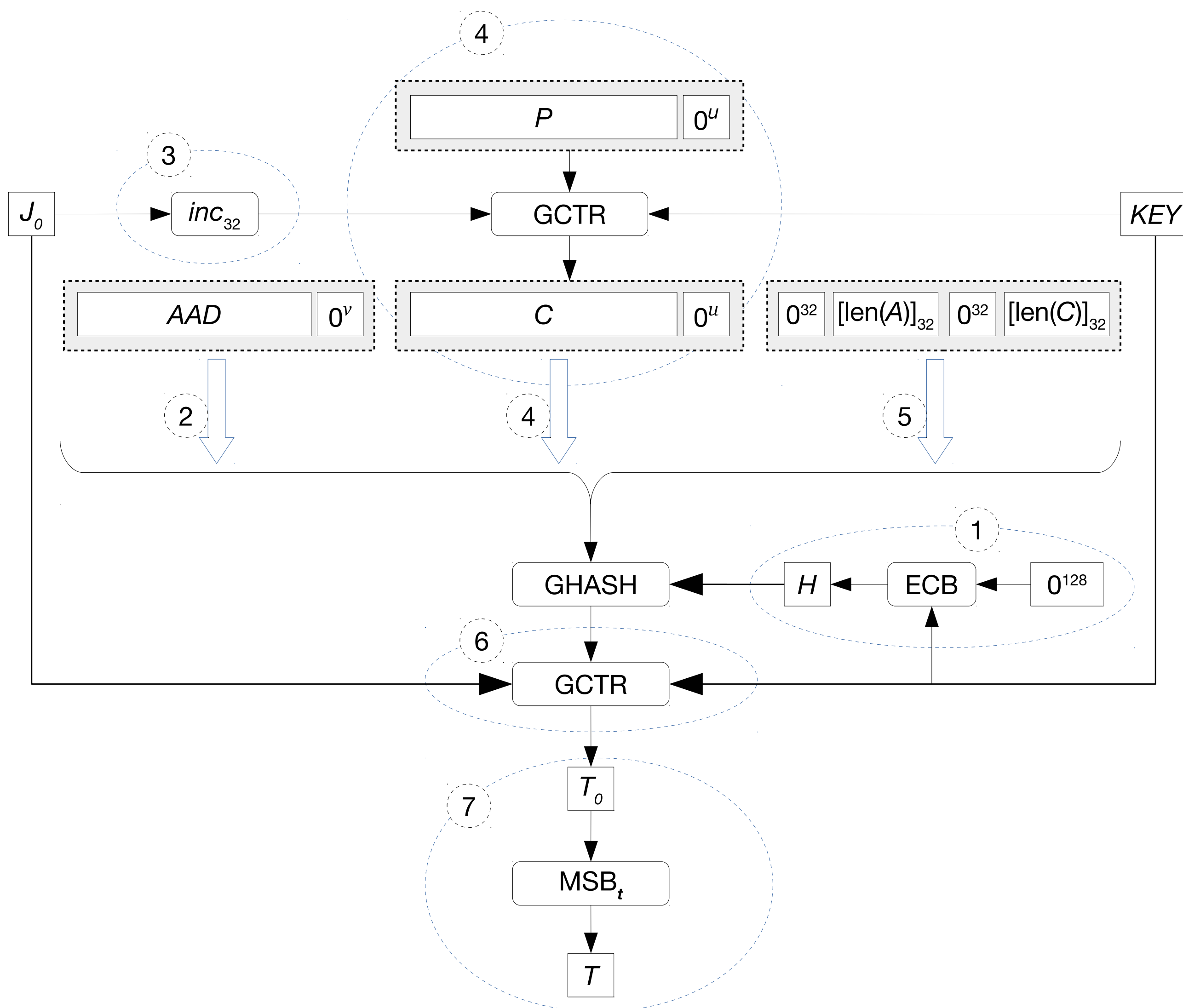


Figure 17-1. GCM Encryption Process

GCM encryption is implemented as follows:

1. Hardware executes the ECB Algorithm to obtain the Hash subkey H, which is needed in the Hash computation.
2. Hardware executes the GHASH Algorithm to perform Hash computation with the padded AAD.
3. Hardware gets ready for CTR encryption by obtaining the result of applying Standard Incrementing Function INC_{32} to J_0.
4. Hardware executes the GCTR Algorithm to encrypt the padded plaintext P, then executes the GHASH Algorithm to perform Hash computation on the padded cipertext C.
5. Hardware executes the GHASH Algorithm to perform Hash computation on AAD Blocks, obtaining a

128-bit Hash result.

6. Hardware executes the GCTR Algorithm to encrypt J_0, obtaining T_0.

7. Software obtains the result T_0 from hardware, and execute MSB$_t$ Algorithm to obtain the final result Authenticated Tag T.

The only difference between GCM decryption and GCM encryption lies in Step 4 in Figure 17-1. To be more specific, instead of executing GCTR Algorithm to encrypt the padded plaintext, the AES Accelerator executes the same Algorithm to decrypt the padded cipertext in GCM decryption. For details, please see NIST SP 800-38D.

17.6.1 Hash Subkey

During GCM operation, the Hash subkey H is a 128-bit value computed by hardware, which is demonstrated in Step 1 in Figure 17-1. Also you can find more information about Hash subkey at "Step 1. Let H = CIPH$_K(0^{128})$" in Chapter 7 GCM Specification of NIST SP 800-38D.

Just like all other Endianness, the Hash subkey H is stored in the AES_H_MEM memory with its most significant (i.e., left-most) byte Byte0 stored at the lowest address and least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 108.

17.6.2 J_0

The J_0 is a 128-bit value computed by hardware, which is required during Step 3 and Step 6 of the GCM process in Figure 17-1. For details on the generation of J_0, please see Chapter 7 GCM Specification in NIST SP 800-38D Specification.

The J_0 is stored in the AES_J0_MEM memory as Endianness. Just like all other Endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 108.

17.6.3 Authenticated Tag

Authenticated Tag (Tag for short) is one of the key results of GCM computation, which is demonstrated in Step 7 of Figure 17-1. The value of Tag is determined by the length of Authenticated Tag t $(1 <= t <= 128)$:

- When $t = 128$, the value of Tag equals to T_0, a 128-bit string that is stored in the AES_T0_MEM as Endianness. Just like all other Endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 108.

- When $1 <= t < 128$, the value of Tag equals to the t most significant (i.e., left-most) bits of T_0. In this case, Tag is represented as MSB$_t(T_0)$, which returns the t most significant bits of T_0. For example, MSB$_4$(111011010) = 1110 and MSB$_5$(11010011010) = 11010. For details on the MSB$_t$( ) function, please refer to Chapter 6 Mathematical Components of GCM in the NIST SP 800-38D specification.

17.6.4 AAD Block Number

Register AES_AAD_BLOCK_NUM_REG stores the Block Number of Additional Authenticated Data (AAD). The length of this register equals to length(**TEXT-PADDING**(AAD))/128. AES Accelerator only uses this register when working in the DMA-AES mode.

17.6.5 Remainder Bit Number

Register AES_REMAINDER_BIT_NUM_REG stores the Remainder Bit Number, which indicates the number of effective bits of incomplete blocks in plaintext/cipertext. The value stored in this register equals to length(P)%128 or length(C)%128. AES Accelerator only uses this register when working in the DMA-AES mode.

Register AES_REMAINDER_BIT_NUM_REG does not affect the results of plaintext or cipertext, but does impact the value of T_0, therefore the Tag value too.

The GCM Algorithm can be viewed as the combination of GCTR operation and GHASH operation, among which, the GCTR performs the encryption and decryption, while the GHASH sloves the Tag.

Note that the AES_REMAINDER_BIT_NUM_REG register is only effective for GCM encryption. To be more specific:

- For GCM encryption, the Hardware firstly computes C, then passes it in the form of **TEXT-PADDING**(C) as the input of GHASH operation. In this case, hardware determines how many trailing "0" should be added based on the content of AES_REMAINDER_BIT_NUM_REG.
- For GCM decrpytion, the padding is completed with the **TEXT-PADDING**(C) function. In this case, the AES_REMAINDER_BIT_NUM_REG register is not effective.

17.7 Base Address

Users can access AES with two base addresses, which can be seen in Table 109. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 109: AES Accelerator Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

17.8 Memory Summary

Both the starting address and ending address in the following table are relative to AES base addresses provided in Section 17.7.

Table 110: AES Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W
AES_H_MEM	Memory H	16 bytes	0x0060	0x006F	RO
AES_J0_MEM	Memory J0	16 bytes	0x0070	0x007F	R/W
AES_T0_MEM	Memory T0	16 bytes	0x0080	0x008F	RO

17.9 Register Summary

The addresses in the following table are relative to AES base addresses provided in Section 17.7.

Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key register 0	0x0000	R/W
AES_KEY_1_REG	AES key register 1	0x0004	R/W
AES_KEY_2_REG	AES key register 2	0x0008	R/W
AES_KEY_3_REG	AES key register 3	0x000C	R/W
AES_KEY_4_REG	AES key register 4	0x0010	R/W
AES_KEY_5_REG	AES key register 5	0x0014	R/W
AES_KEY_6_REG	AES key register 6	0x0018	R/W
AES_KEY_7_REG	AES key register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result data register 3	0x003C	RO
Configuration Registers			
AES_MODE_REG	AES working mode configuration register	0x0040	R/W
AES_ENDIAN_REG	Endian configuration register	0x0044	R/W
AES_DMA_ENABLE_REG	DMA enable register	0x0090	R/W
AES_BLOCK_MODE_REG	Block operation type register	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
AES_AAD_BLOCK_NUM_REG	AAD block number configuration register	0x00A0	R/W
AES_REMAINDER_BIT_NUM_REG	Remainder bit number of plaintext/ciphertext	0x00A4	R/W
Controlling / Status Registers			
AES_TRIGGER_REG	Operation start controlling register	0x0048	WO
AES_STATE_REG	Operation status register	0x004C	RO
AES_CONTINUE_REG	Operation continue controlling register	0x00A8	WO
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
Interruption Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WO
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

17.10 Registers

Register 17.1: AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**

31	0	
0x000000000		Reset

AES_KEY_*n*_REG (*n*: 0-7) Stores AES keys. (R/W)

Register 17.2: AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**

31	0	
0x000000000		Reset

AES_TEXT_IN_*m*_REG (*m*: 0-3) Stores the source data when the AES Accelerator operates in the Typical AES working mode. (R/W)

Register 17.3: AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**

31	0	
0x000000000		Reset

AES_TEXT_OUT_*m*_REG (*m*: 0-3) Stores the result data when the AES Accelerator operates in the Typical AES working mode. (RO)

Register 17.4: AES_MODE_REG (0x0040)

(reserved)	AES_MODE	
31 … 3	2 … 0	
0x00000000	0	Reset

AES_MODE Defines the operation type of the AES Accelerator operating under the Typical AES working mode. For details, see Table 99. (R/W)

Register 17.5: AES_ENDIAN_REG (0x0044)

(reserved) 31–6	AES_ENDIAN 5–0	
0x0000000	0 0 0 0 0 0	Reset

AES_ENDIAN Defines the endianness of input and output texts. For details, please see Table 101. (R/W)

Register 17.6: AES_DMA_ENABLE_REG (0x0090)

(reserved) 31–1	AES_DMA_ENABLE 0	
0x00000000	0	Reset

AES_DMA_ENABLE Defines the working mode of the AES Accelerator. For details, see Table 98. (R/W)

Register 17.7: AES_BLOCK_MODE_REG (0x0094)

(reserved) 31–3	AES_BLOCK_MODE 2–0	
0x00000000	0	Reset

AES_BLOCK_MODE Defines the operation type of the AES Accelerator operating under the DMA-AES working mode. For details, see Table 105. (R/W)

Register 17.8: AES_BLOCK_NUM_REG (0x0098)

31–0	
0x00000000	Reset

AES_BLOCK_NUM Stores the Block Number of plaintext or cipertext when the AES Accelerator operates under the DMA-AES working mode. For details, see Section 17.5.4. (R/W)

Register 17.9: AES_INC_SEL_REG (0x009C)

(reserved)	AES_INC_SEL
31 ... 1	0
0x00000000	0

Reset

AES_INC_SEL Defines the Standard Incrementing Function for CTR block operation. Set this bit to 0 or 1 to choose INC_{32} or INC_{128}. (R/W)

Register 17.10: AES_AAD_BLOCK_NUM_REG (0x00A0)

31 ... 0
0x00000000

Reset

AES_AAD_BLOCK_NUM Stores the ADD Block Number for the GCM operation. (R/W) For details, see Section 17.6.4.

Register 17.11: AES_REMAINDER_BIT_NUM_REG (0x00A4)

(reserved)	AES_REMAINDER_BIT_NUM
31 ... 7	6 ... 0
0x0000000	0

Reset

AES_REMAINDER_BIT_NUM Stores the Remainder Bit Number for the GCM operation. For details, see Section 17.6.5. (R/W)

Register 17.12: AES_TRIGGER_REG (0x0048)

(reserved)	AES_TRIGGER
31 ... 1	0
0x00000000	x

Reset

AES_TRIGGER Set this bit to 1 to start AES operation. (WO)

Register 17.13: AES_STATE_REG (0x004C)

(reserved)		AES_STATE	
31	2	1	0
0x00000000		0x0	

Reset

AES_STATE Stores the working status of the AES Accelerator. For details, see Table 100 for Typical AES working mode and Table 106 for DMA AES working mode. (RO)

Register 17.14: AES_CONTINUE_REG (0x00A8)

(reserved)		AES_CONTINUE
31	1	0
0x00000000		x

Reset

AES_CONTINUE Set this bit to 1 to continue AES operation. (WO)

Register 17.15: AES_DMA_EXIT_REG (0x00B8)

(reserved)		AES_DMA_EXIT
31	1	0
0x00000000		x

Reset

AES_DMA_EXIT Set this bit to 1 to exit AES operation. This register is only effective for DMA-AES operation. (WO)

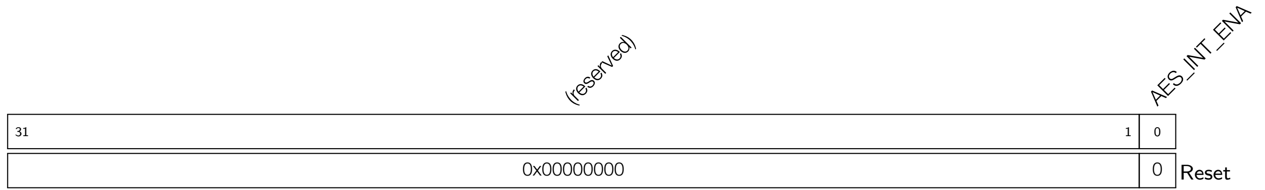
Register 17.16: AES_INT_CLR_REG (0x00AC)

(reserved)		AES_INT_CLR
31	1	0
0x00000000		x

Reset

AES_INT_CLR Set this bit to 1 to clear AES interrupt. (WO)

Register 17.17: AES_INT_ENA_REG (0x00B0)



AES_INT_ENA Set this bit to 1 to enable AES interrupt and 0 to disable interrupt. (R/W)

18. RSA Accelerator (RSA)

18.1 Introduction

The RSA Accelerator provides hardware support for high precision computation used in various RSA asymmetric cipher algorithms by significantly reducing their software complexity. Compared with RSA algorithms implemented solely in software, this hardware accelerator can speed up RSA algorithms significantly. Besides, the RSA Accelerator also supports operands of different lengths, which provides more flexibility during the computation.

18.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two optional acceleration options
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

18.3 Functional Description

The RSA Accelerator is activated by setting the SYSTEM_CRYPTO_RSA_CLK_EN bit in the SYSTEM_PERIP_CLK_EN1_REG register and clearing the SYSTEM_RSA_MEM_PD bit in the SYSTEM_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

The RSA Accelerator is only available after the RSA-related memories are initialized. The content of the RSA_CLEAN_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, it is advised to wait until RSA_CLEAN_REG becomes 1 before using the RSA Accelerator.

The RSA_INTERRUPT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this register to enable or disable interrupt. By default, the interrupt function of the RSA Accelerator is enabled.

> **Notice:**
> ESP32-S2's Digital Signature (DS) module also calls the RSA accelerator. Therefore, users cannot access the RSA accelerator when Digital Signature (DS) is working.

18.3.1 Large Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

RSA Accelerator supports operands of length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2} \cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2} \cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2} \cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2} \cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2} \cdots \overline{r}_0)_b$$

Each of the n values in $Z_{n-1} \cdots Z_0$, $X_{n-1} \cdots X_0$, $Y_{n-1} \cdots Y_0$, $M_{n-1} \cdots M_0$, $\overline{r}_{n-1} \cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional arguments can be calculated as $\overline{r} = R^2 \bmod M$, where $R = b^n$.

The following equation in the form compatible with the extended binary GCD algorithm can be written as

$$M^{-1} \times M + 1 = R \times R^{-1}$$
$$M' = M^{-1} \bmod b$$

Large-number modular exponentiation can be implemented as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
   (c) Configure registers related to the acceleration options, which are described later in Section 18.3.4.
3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MODEXP_START_REG register to start computation.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.

6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

18.3.2 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. The same values $\overline{r}$ and M' are derived by software.

The RSA Accelerator supports large-number modular multiplication with operands of 128 different lengths.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n\}$ to registers RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MODMULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

18.3.3 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA Accelerator only supports Large Number Multiplication with operand length $N = 32 \times x$, where $x \in \{0, 1, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e. $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.

3. Write X_i and Y_i for $\in \{0, 1, \ldots, n\}$ to registers RSA_X_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n\}$ to the address of the i words of the RSA_X_MEM register. Note that Y_i for $i \in \{0, 1, \ldots, n\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e. the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.

4. Write 1 to the RSA_MULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

18.3.4 Acceleration Options

ESP32-S2 RSA provides two acceleration options for the large-number modular exponentiation, which are SEARCH Option and the CONSTANT_TIME Option. These two options are both disabled by default, but can be enabled at the same time.

When neither of these two options are enabled, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. However, when either one of these two options is enabled, the time required is also correlated with the 0/1 distribution of Y.

To better illustrate the acceleration options, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2} \cdots \widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1} \cdots \widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, …, $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, …, $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e. the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When the acceleration options are enabled, the RSA accelerator:

- SEARCH Option

  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. The maximum value of α is N-1, which leads to the same result when this accleration option is disabled. The best acceleration performance can be achieved by setting α to t, in which case, all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.

- CONSTANT_TIME Option
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better the acceleration performance is.

We provide an example to demonstrate the performance of the RSA Accelerator when different acceleration options are enabled. Here we perform $Z = X^Y \bmod M$ with N = 3072 and Y = 65537. Table 112 below demonstrates the time costs when different acceleration options are enabled. It's obvious that the time cost can be dramatically reduced when acceleration option(s) is enabled. Here, we should also mention that, α is set to 16 when the SEARCH option is enabled.

Table 112: Acceleration Performace

SEARCH Option	CONSTANT_TIME Option	Time Cost	Acceleration Performance by Percentage
Disabled	Disabled	376.405 ms	0%
Enabled	Disabled	2.260 ms	99.41%
Disabled	Enabled	1.203 ms	99.68%
Enabled	Enabled	1.165 ms	99.69%

18.4 Base Address

Users can access RSA with two base addresses, which can be seen in Table 113. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 113: RSA Accelerator Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43C000
PeriBUS2	0x6003C000

18.5 Memory Summary

Both the starting address and ending address in the following table are relative to RSA base addresses provided in Section 18.4.

Table 114: RSA Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	512	0x0000	0x01FF	WO
RSA_Z_MEM	Memory Z	512	0x0200	0x03FF	R/W
RSA_Y_MEM	Memory Y	512	0x0400	0x05FF	WO
RSA_X_MEM	Memory X	512	0x0600	0x07FF	WO

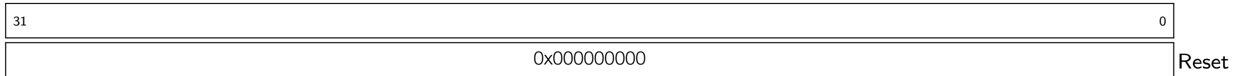
18.6 Register Summary

The addresses in the following table are relative to RSA base addresses provided in Section 18.4.

Name	Description	Address	Access
Configuration Registers			
RSA_M_PRIME_REG	Register to store M'	0x0800	R/W
RSA_MODE_REG	RSA length mode	0x0804	R/W
RSA_CONSTANT_TIME_REG	The constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	The search option	0x0824	R/W
RSA_SEARCH_POS_REG	The search position	0x0828	R/W
Status/Control Registers			
RSA_CLEAN_REG	RSA clean register	0x0808	RO
RSA_MODEXP_START_REG	Modular exponentiation starting bit	0x080C	WO
RSA_MODMULT_START_REG	Modular multiplication starting bit	0x0810	WO
RSA_MULT_START_REG	Normal multiplication starting bit	0x0814	WO
RSA_IDLE_REG	RSA idle register	0x0818	RO
Interupt Registers			
RSA_CLEAR_INTERRUPT_REG	RSA clear interrupt register	0x081C	WO
RSA_INTERRUPT_ENA_REG	RSA interrupt enable register	0x082C	R/W
Version Register			
RSA_DATE_REG	Version control register	0x0830	R/W

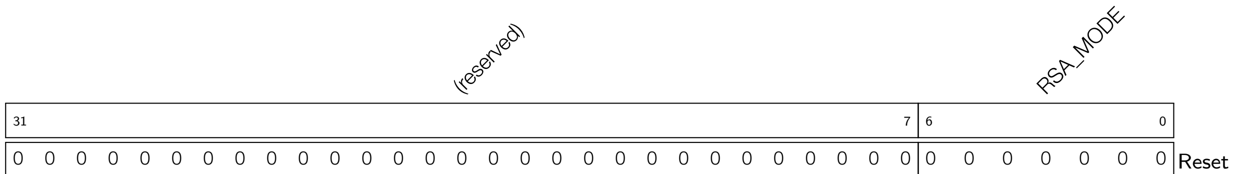
18.7 Registers

Register 18.1: RSA_M_PRIME_REG (0x800)



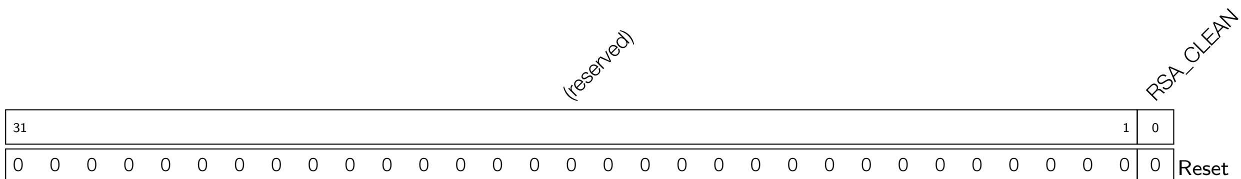
RSA_M_PRIME_REG Stores M'.(R/W)

Register 18.2: RSA_MODE_REG (0x804)



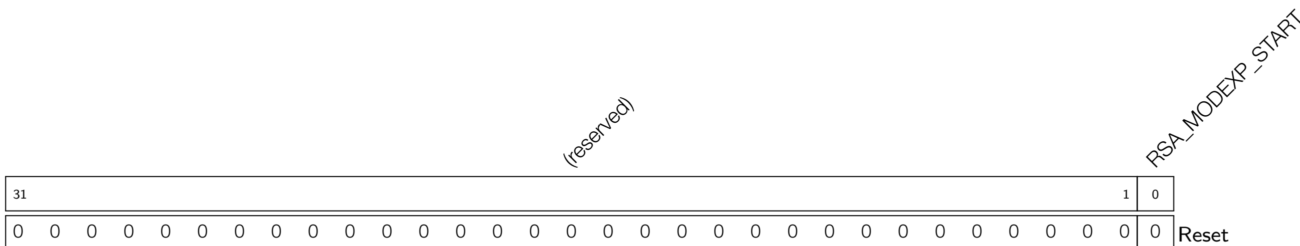
RSA_MODE Stores the mode of modular exponentiation. (R/W)

Register 18.3: RSA_CLEAN_REG (0x0808)



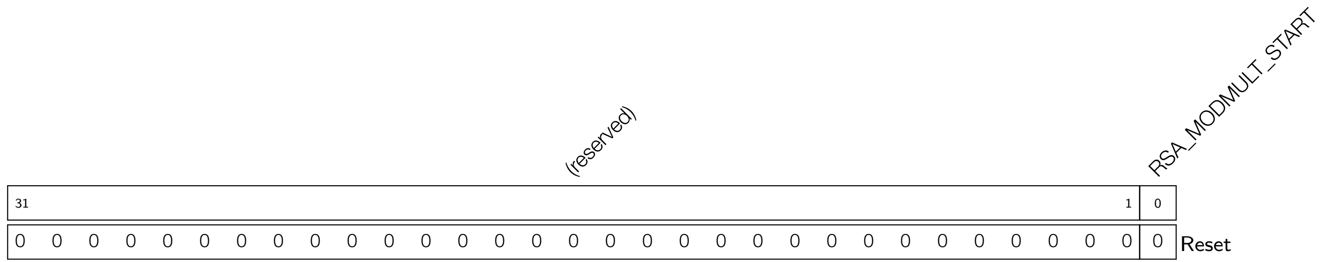
RSA_CLEAN The content of this bit is 1 when memories complete initialization. (RO)

Register 18.4: RSA_MODEXP_START_REG (0x080C)



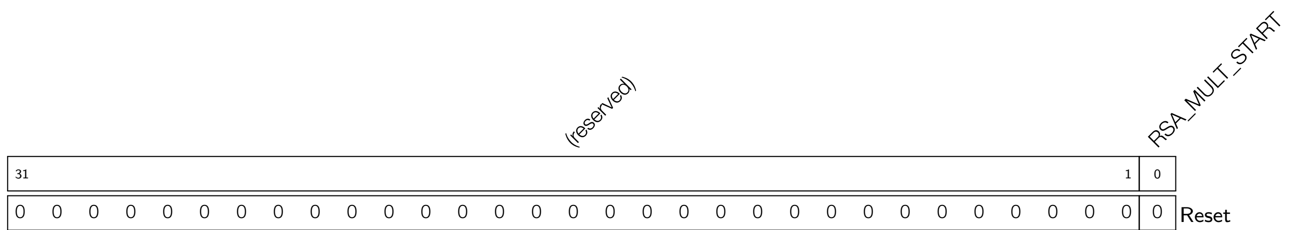
RSA_MODEXP_START Set this bit to 1 to start the modular exponentiation. (WO)

Register 18.5: RSA_MODMULT_START_REG (0x0810)



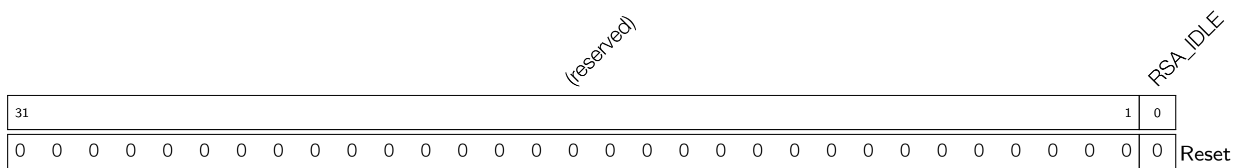
RSA_MODMULT_START Set this bit to 1 to start the modular multiplication. (WO)

Register 18.6: RSA_MULT_START_REG (0x0814)



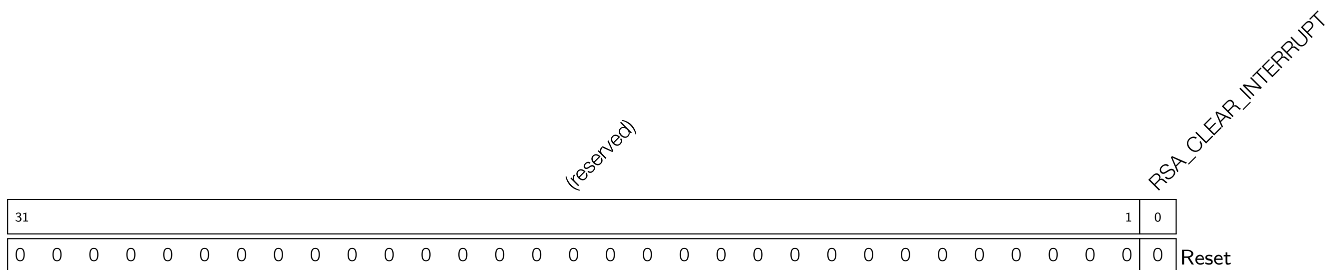
RSA_MULT_START Set this bit to 1 to start the multiplication. (WO)

Register 18.7: RSA_IDLE_REG (0x0818)



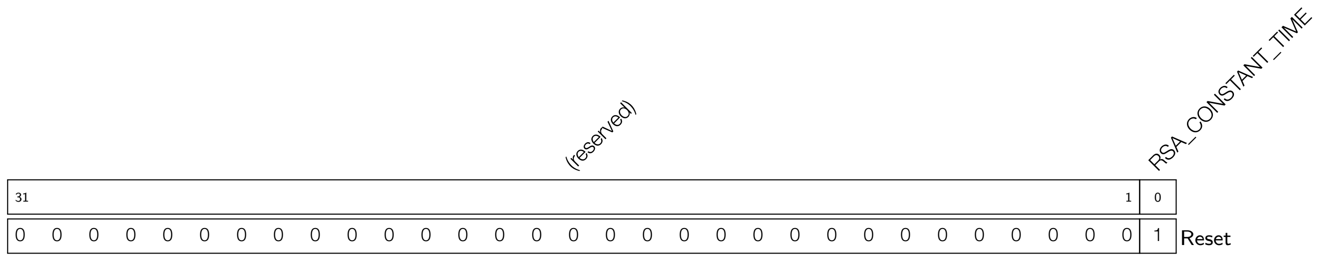
RSA_IDLE The content of this bit is 1 when the RSA accelerator is idle. (RO)

Register 18.8: RSA_CLEAR_INTERRUPT_REG (0x081C)



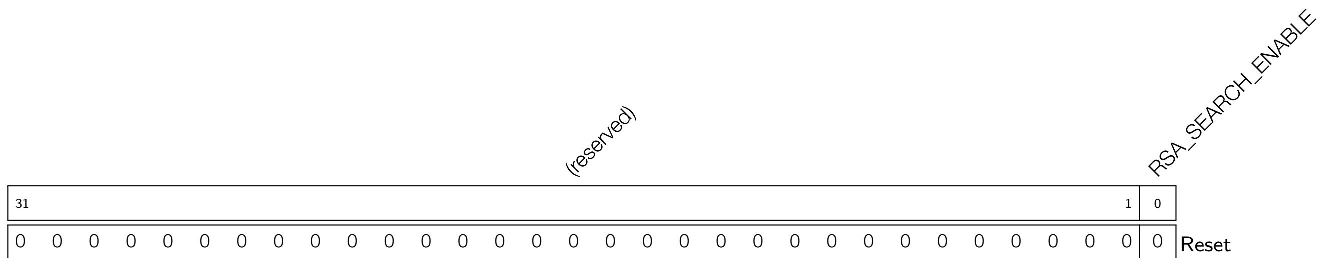
RSA_CLEAR_INTERRUPT Set this bit to 1 to clear the RSA interrupts. (WO)

Register 18.9: RSA_CONSTANT_TIME_REG (0x0820)



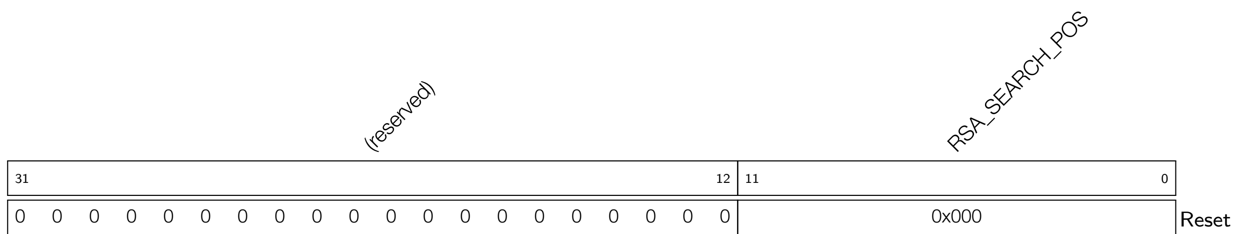
RSA_CONSTANT_TIME_REG Set this bit to 0 to enable the acceleration option of constant_time for modular exponentiation. Set to 1 to disable the acceleration (by default). (R/W)

Register 18.10: RSA_SEARCH_ENABLE_REG (0x0824)



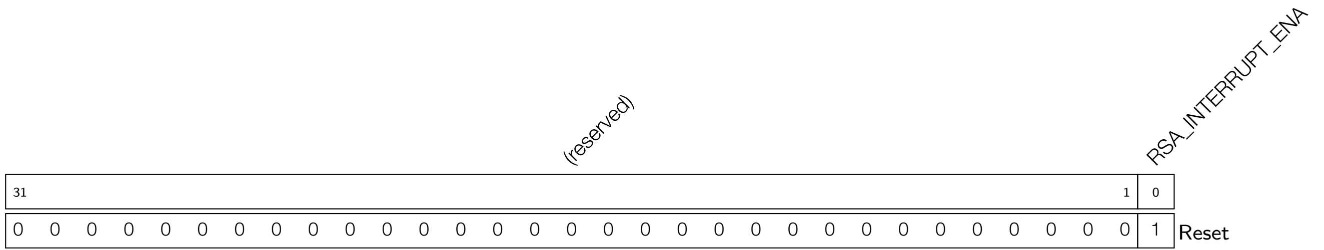
RSA_SEARCH_ENABLE Set this bit to 1 to enable the acceleration option of search for modular exponentiation. Set to 0 to disable the acceleration (by default). (R/W)

Register 18.11: RSA_SEARCH_POS_REG (0x0828)



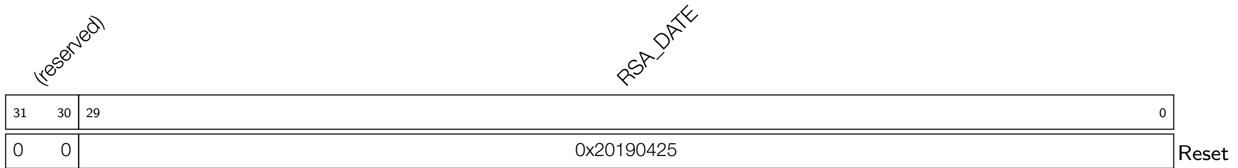
RSA_SEARCH_POS Is used to configure the starting address when the acceleration option of search is used. (R/W)

Register 18.12: RSA_INTERRUPT_ENA_REG (0x082C)



RSA_INTERRUPT_ENA Set this bit to 1 to enable the RSA interrupt. This option is enabled by default. (R/W)

Register 18.13: RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

19. HMAC Accelerator (HMAC)

19.1 Overview

The Hash-based Message Authentication Code (HMAC) module computes Message Authentication Codes (MACs) through Hash algorithms and keys as described in RFC 2104. The underlying hash algorithm is SHA-256, and the 256-bit HMAC key is stored in an eFuse key block and can be configured as not readable by software.

The HMAC module can be used in two modes - in ”upstream” mode the HMAC message is supplied by the user and the calculation result is read back by the user. In ”downstream” mode the HMAC module is used as a Key Derivation Function (KDF) for other internal hardwares.

19.2 Main Features

- Standard HMAC-SHA-256 algorithm
- The hash result is only accessible by the configurable hardware peripheral (downstream mode)
- Supports identity verification challenge-response algorithms
- Supports Digital Signature peripheral (downstream mode)
- Supports re-enabling of Soft-Disabled JTAG (downstream mode)

19.3 Functional Description

19.3.1 Upstream Mode

In Upstream mode, the HMAC message is provided by the user and the result is read back by the user.

This allows the key stored in eFuse (can be configured as not readable by software) to become a shared secret between the user and another party. Any challenge-response protocol that supports HMAC-SHA-256 can be used in this way.

The generalized form of these protocols is as follows:

- A calculates a unique nonce message M
- A sends M to B
- B calculates HMAC (M, KEY) and sends to A
- A also calculates HMAC (M, KEY) internally
- A compares both results, If the same, then the identity of B is verified

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with corresponding purpose eFuse set to EFUSE_KEY_PURPOSE_HMAC_UP. See Chapter 4 for details.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. A copy of this secret key should be kept by any other party that wants to authenticate this device.

To calculate an HMAC:

1. User initializes the HMAC module,and enter upstream mode.

2. User writes the correctly padded message to the peripheral, one block at a time.
3. User reads back the HMAC result from peripheral registers.

See Section 19.3.5 for detailed steps of this process.

19.3.2 Downstream JTAG Enable Mode

eFuse memory has two parameters to disable JTAG debugging: EFUSE_HARD_DIS_JTAG and EFUSE_SOFT_DIS_JTAG. JTAG will be disabled permanently if the former is programmed to 1, and it will be disabled temporarily if the latter is programmed to 1. See Chapter 4 for details.

The HMAC peripheral can be used to re-enable JTAG when EFUSE_SOFT_DIS_JTAG is programmed.

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with purpose set to either EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG or EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL. In the latter case, the same key can be used for both DS and JTAG re-enable functions.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. User stores the randomly generated HMAC key in the process securely elsewhere.
3. Program the eFuse EFUSE_SOFT_DIS_JTAG to 1.

To re-enable JTAG:

1. User performs HMAC calculation on the 32-byte 0x00 locally using SHA-256 and the known random key, and inputs this pre-calculated value into the registers SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 ~ SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7.
2. User enables the HMAC module, and enters downstream JTAG enable mode.
3. If the HMAC calculated result matches the value supplied in the registers SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 ~ SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7, JTAG is re-enabled. Otherwise, JTAG remains disabled.
4. JTAG remains the status in step 3 until the user writes 1 in register HMAC_SET_INVALIDATE_JTAG_REG, or restarts the system.

See Section 19.3.5 for detailed steps of this process.

19.3.3 Downstream Digital Signature Mode

The Digital Signature (DS) module encrypts its parameters using AES-CBC. The HMAC module is used as a Key Derivation Function (KDF) to derive the AES key used for this encryption.

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with purpose set to either EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE or EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL. In the latter case, the same key can be used for both DS and JTAG re-enable functions.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. If necessary a copy of the key can also be stored in a secure location.

Before using the DS module, software needs to enable the calculation task of HMAC module's downstream DS mode. The above calculation result will be used as the key when the DS mode performs the calculation task. Consult the 20 chapter for details.

19.3.4 HMAC eFuse Configuration

The correct implementation of the HMAC module depends on whether the selected eFuse key block is consistent with the configured HMAC function.

Configure HMAC Function

Currently, HMAC module supports three functions: JTAG re-enable in downstream mode, DS Key Derivation in downstream mode, and HMAC calculation in upstream mode. Table 116 lists the configuration register value corresponding to each function. The values corresponding to the function in use should be written into the register HMAC_SET_PARA_PURPOSE_REG (see Section 19.3.5).

Table 116: HMAC Function and Configuration Value

Functions	Mode Type	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
DS Key Derivation	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and DS KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Select eFuse Key Blocks

The eFuse controller provides six key blocks, KEY0 ~ 5. To select a particular KEYn for HMAC module use at runtime, user writes the number n into register HMAC_SET_PARA_KEY_REG.

Note that the purpose of the key is also programmed into eFuse memory. Only when the configured HMAC purpose matches the purpose defined in KEYn, will the HMAC module execute the configured computation.

For more information see 4 chapter.

For example, suppose the user selects KEY3 for the computation, and the value programmed in KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 116, KEY3 is the key used for JTAG restart. If the configured value of HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC peripheral will allow the JTAG start the computation of JTAG re-enable.

19.3.5 HMAC Process (Detailed)

The process to call HMAC in ESP32-S2 is as follows:

1. Enable HMAC module
   (a) Enable the peripheral clock bits for HMAC and SHA peripherals, and clear the corresponding peripheral reset bits.
   (b) Write 1 into register HMAC_SET_START_REG.
2. Configure HMAC keys and key functions

(a) Write m representing key functions into register HMAC_SET_PARA_PURPOSE_REG. Correlation between value m and key functions is shown in Table 27. Refer to Section 19.3.4.

(b) Select KEYn of eFuse memory as the key by writing n into register HMAC_SET_PARA_KEY_REG (n in the range of 0 to 5). Refer to Section 19.3.4.

(c) Finish the configuration by writing 1 into register HMAC_SET_PARA_FINISH_REG.

(d) Read register HMAC_QUERY_ERROR_REG. Value of 1 means the selected key block does not match the configured key purpose, and computation will not proceed. Value of 0 means the selected key block matches the configured key purpose, and computation can proceed.

(e) Setting HMAC_SET_PARA_PURPOSE_REG to values other than 8 means HMAC module will operate in downstream mode, proceed with Step 3. Setting value 8 means HMAC module will operate in upstream mode, proceed with Step 4.

3. Downstream Mode

(a) Poll state register HMAC_QUERY_BUSY_REG. The register value of 0 means HMAC computation in downstream mode is finished.

(b) In downstream mode, the result is used by JTAG module or DS module in the hardware. Users can write 1 into register HMAC_SET_INVALIDATE_JTAG_REG to clean the result generated by JTAG key; or can write 1 into register HMAC_SET_INVALIDATE_DS_REG to clean the result generated by digital signature key.

(c) This is the end of the HMAC downstream operation.

4. Upstream Mode Transmit message block Block_n ($n >= 1$)

(a) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

(b) Write 512-bit message block Block_n into register range HMAC_WDATA0~15_REG. Then write 1 in register HMAC_SET_MESSAGE_ONE_REG, and HMAC module will compute this message block.

(c) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

(d) Subsequent message blocks are different, depending on whether the size of to-be-processed data is a multiple of 512 bits.

- If the bit length of the message is a multiple of 512, there are three possible options:

  i. If Block_n+1 exists, write 1 in register HMAC_SET_MESSAGE_ING_REG to make $n = n + 1$, then jump to step 4.(b).

  ii. If Block_n is the last block of the message, and user wishes to apply SHA padding through hardware, write 1 in register HMAC_SET_MESSAGE_END_REG, then jump to step 6.

  iii. If Block_n is the last block of the padded message, and the user has applied SHA padding in software, write 1 in register HMAC_SET_MESSAGE_PAD_REG, and jump to step 5.

- If the bit length of the message is not a multiple of 512, there are three possible options. Note that in this case the user is required to apply SHA padding to the message, after which the padded message length will be a multiple of 512 bits.

  i. If Block_n is the only message block, $n = 1$, and Block_1 has included all padding bits, write 1 in register HMAC_ONE_BLOCK_REG, and then jump to step 6.

ii. If Block_n is the second last block of the padded message, write 1 in register HMAC_SET_MESSAGE_PAD_REG, and jump to step 5.

iii. If Block_n is neither the last nor the second last message block, write 1 in register HMAC_SET_MESSAGE_ING_REG and make $n = n + 1$, then jump to step 4.(b).

5. Apply SHA Padding to Message

   (a) Users apply SHA padding to the final message block as described in Section 19.4.1, Write this block in register HMAC_WDATA0~15_REG, then write 1 in register HMAC_SET_MESSAGE_ONE_REG. HMAC module will compute the message block.

   (b) Jump to step 6.

6. Read hash result in upstream mode

   (a) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

   (b) Read hash result from register HMAC_RDATA0~7_REG.

   (c) Write 1 in register HMAC_SET_RESULT_FINISH_REG to finish the computation.

Note:
The SHA accelerator can be called directly, or used internally by the DS module and the HMAC module. However, they can not share the hardware resources simultaneously. Therefore, SHA module can not be called by CPU or DS module when HMAC module is in use.

19.4 HMAC Algorithm Details

19.4.1 Padding Bits

HMAC module uses the SHA-256 hash algorithm. If the input message is not a multiple of 512 bits, the user must apply SHA-256 padding algorithm in software. The SHA-256 padding algorithm is described here, and is the same as "5.1 Padding the Message" of "FIPS PUB 180-4".

As shown in Figure 19-1, suppose the length of the unpadded message is m bits. Padding steps are as follows:

1. Append a single 1-bit "1" to the end of unpadded message;
2. Append k bits of value "0", where k is the smallest non-negative number which satisfies $m + 1 + k \equiv 448 (mod 512)$;
3. Append a 64-bit integer value as a binary block. The content of this block is the length of the the unpadded message as a big-endian binary integer value m.

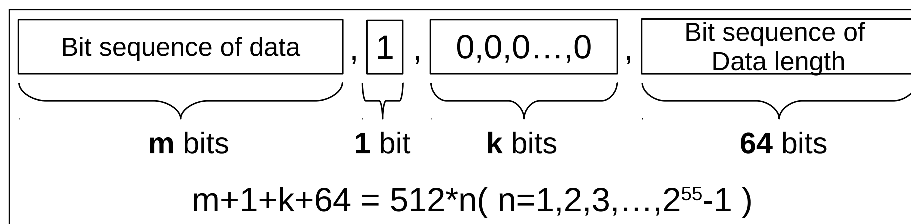


Figure 19-1. HMAC SHA-256 Padding Diagram

In downstream mode, there is no need for users to input any message data or apply padding. In upstream mode, if the total number of bits in the unpadded message is a multiple of 512, users can choose to configure hardware to apply SHA padding. If the total number of bits in the unpadded message is not a multiple of 512, SHA padding must be applied by the user. See Section 19.3.5 for the steps involved.

19.4.2 HMAC Algorithm Structure

The Structure of HMAC algorithm as implemented in the HMAC module is shown in Figure 19-2. This is the standard HMAC algorithm as described in RFC 2104.

Figure 19-2. HMAC Structure Schematic Diagram

In Figure 19-2,

1. ipad is 512-bit message block composed of sixty-four 0x36 byte.
2. opad is 512-bit message block composed of sixty-four 0x5c byte.

HMAC module appends 256-bit 0 sequence after the bit sequence of 256-bit key k, and gets 512-bit K_0. Afterwards, HMAC module XORs K_0 with ipad to get 512-bit S1. Then, HMAC module appends the input message which is a multiple of 512 after 512-bit S1, and processes SHA-256 algorithm to get 256-bit H1.

HMAC module appends the 256-bit SHA-256 hash result to the 512-bit S2 value, which is calculated by XOR between K_0 and opad, this produces a 768-bit sequence. Then, HMAC module uses SHA padding algorithm described in Section 19.4.1 to pad 768-bit sequence into 1024-bit sequence, and applies SHA-256 algorithm to get the final hash result.

19.5 Base Address

Users can access the HMAC module with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and*

Memory.

Table 117: HMAC Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43E000
PeriBUS2	0x6003E000

19.6 Register Summary

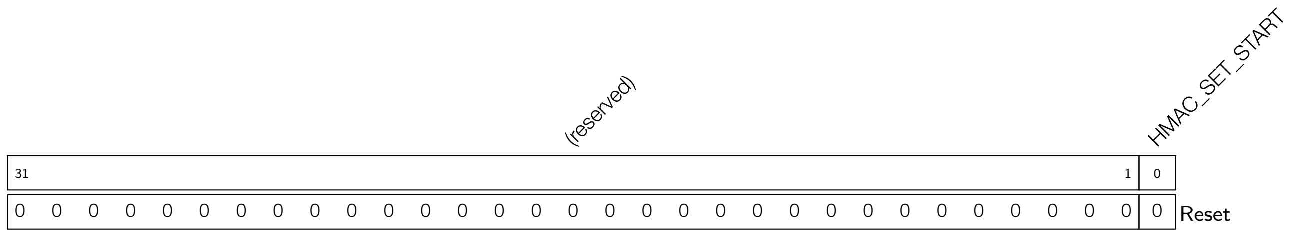
The addresses in the following table are relative to the system registers base addresses provided in Section 19.5.

Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC one message control register	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC read result completion register	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	Invalidate digital signature result register	0x0064	WO
HMAC_QUERY_ERROR_REG	The matching result between key and purpose user configured	0x0068	RO
HMAC_QUERY_BUSY_REG	The busy state of HMAC module	0x006C	RO
configuration Registers			
HMAC_SET_PARA_PURPOSE_REG	HMAC parameter configuration register	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC key configuration register	0x0048	WO
HMAC Message Block			
HMAC_WR_MESSAGE_0_REG	Message register 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	Message register 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	Message register 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	Message register 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	Message register 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	Message register 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	Message register 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	Message register 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	Message register 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	Message register 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	Message register 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	Message register 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	Message register 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	Message register 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	Message register 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	Message register 15	0x00BC	WO

Name	Description	Address	Access
HMAC Upstream Result			
HMAC_RD_RESULT_0_REG	Hash result register 0	0x00C0	RO
HMAC_RD_RESULT_1_REG	Hash result register 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash result register 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash result register 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash result register 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash result register 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash result register 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash result register 7	0x00DC	RO
Control/Status Registers			
HMAC_SET_MESSAGE_PAD_REG	Software padding register	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block message register	0x00F4	WO
Version Register			
HMAC_DATE_REG	Version control register	0x00F8	R/W

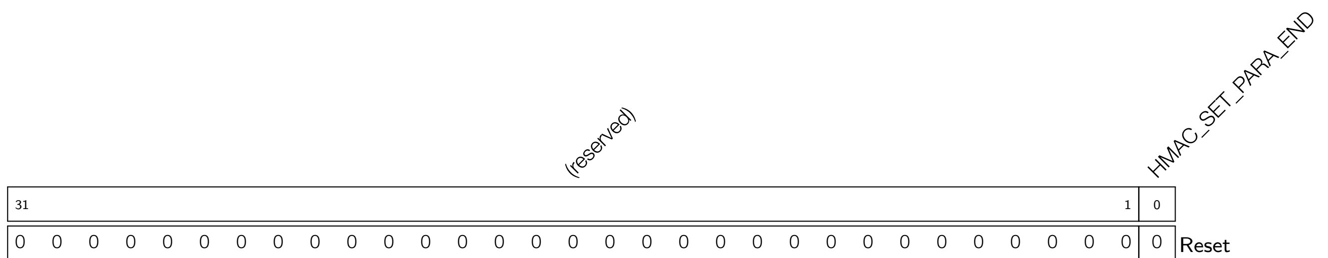
19.7 Registers

Register 19.1: HMAC_SET_START_REG (0x0040)



HMAC_SET_START Set this bit to enable HMAC. (WO)

Register 19.2: HMAC_SET_PARA_FINISH_REG (0x004C)



HMAC_SET_PARA_END Set this bit to finish HMAC configuration. (WO)

Register 19.3: HMAC_SET_MESSAGE_ONE_REG (0x0050)

(reserved) 31–1	HMAC_SET_TEXT_ONE 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_TEXT_ONE Call SHA to calculate one message block. (WO)

Register 19.4: HMAC_SET_MESSAGE_ING_REG (0x0054)

(reserved) 31–1	HMAC_SET_TEXT_ING 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_TEXT_ING Set this bit to show there are still some message blocks to be processed. (WO)

Register 19.5: HMAC_SET_MESSAGE_END_REG (0x0058)

(reserved) 31–1	HMAC_SET_TEXT_END 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_TEXT_END Set this bit to start hardware padding. (WO)

Register 19.6: HMAC_SET_RESULT_FINISH_REG (0x005C)

(reserved)	HMAC_SET_RESULT_END	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_RESULT_END Set this bit to end upstream and clear the calculation result. (WO)

Register 19.7: HMAC_SET_INVALIDATE_JTAG_REG (0x0060)

(reserved)	HMAC_SET_INVALIDATE_JTAG	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_INVALIDATE_JTAG Set this bit to clear calculation results in JTAG re-enable function under downstream mode. (WO)

Register 19.8: HMAC_SET_INVALIDATE_DS_REG (0x0064)

(reserved)	HMAC_SET_INVALIDATE_DS	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

HMAC_SET_INVALIDATE_DS Set this bit to clear calculation results in DS function under downstream mode. (WO)

Register 19.9: HMAC_QUERY_ERROR_REG (0x0068)

(reserved)	HMAC_QUREY_CHECK
31 … 1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0

Reset

HMAC_QUREY_CHECK HMAC error status. 0: HMAC key and purpose match. 1: error. (RO)

Register 19.10: HMAC_QUERY_BUSY_REG (0x006C)

(reserved)	HMAC_BUSY_STATE
31 … 1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0

Reset

HMAC_BUSY_STATE The state of HMAC. 1'b0: idle. 1'b1: busy. (RO)

Register 19.11: HMAC_SET_PARA_PURPOSE_REG (0x0044)

(reserved)	HMAC_PURPOSE_SET
31 … 4	3 … 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0

Reset

HMAC_PURPOSE_SET Set HMAC purpose. (WO)

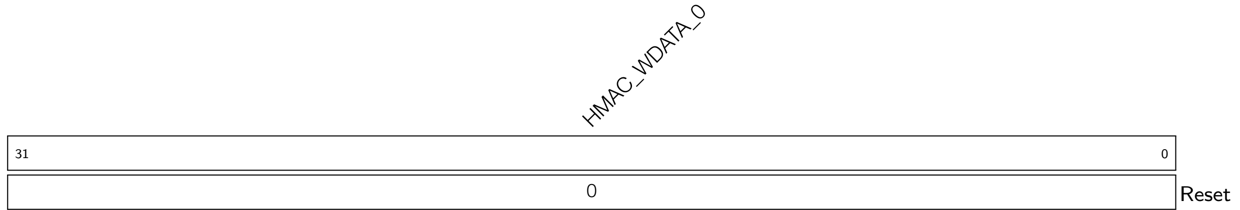
Register 19.12: HMAC_SET_PARA_KEY_REG (0x0048)

(reserved)	HMAC_KEY_SET
31 … 3	2 … 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0

Reset

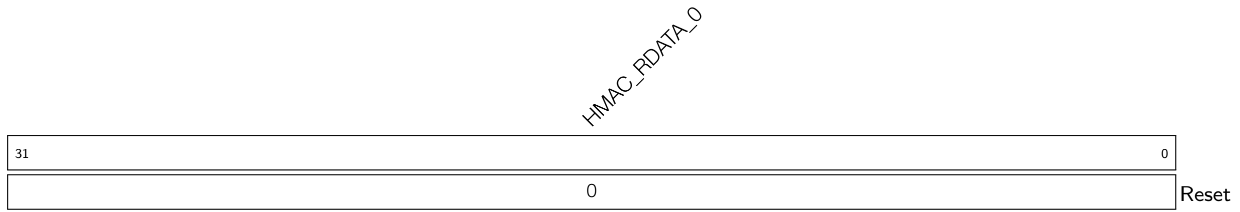
HMAC_KEY_SET Select HMAC key. (WO)

Register 19.13: HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4**n*)



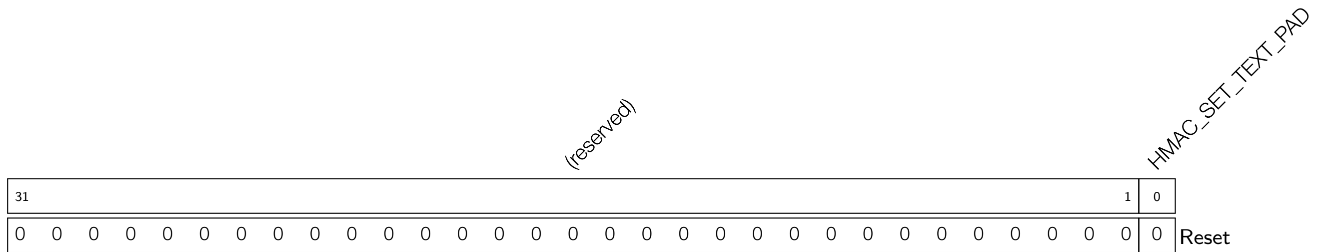
HMAC_WDATA_*n* Store the *n*th 32-bit of message. (WO)

Register 19.14: HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4**n*)



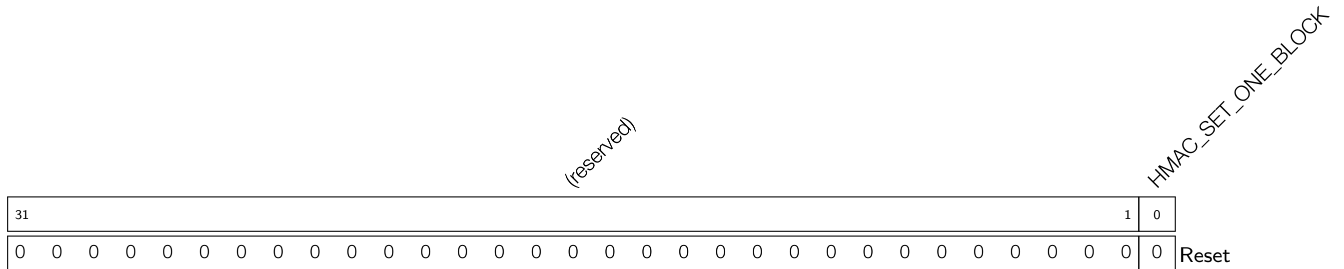
HMAC_RDATA_*n* Read the *n*th 32-bit of hash result. (RO)

Register 19.15: HMAC_SET_MESSAGE_PAD_REG (0x00F0)



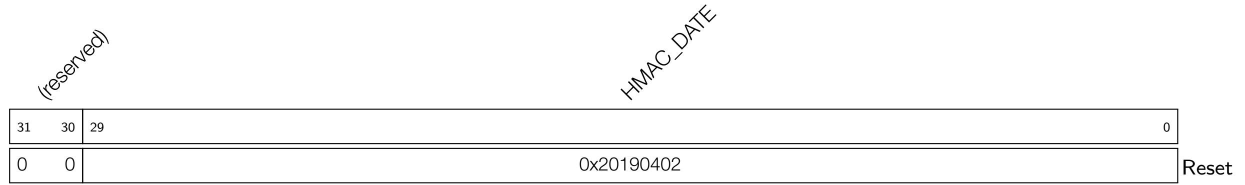
HMAC_SET_TEXT_PAD Set this bit to let software do padding job. (WO)

Register 19.16: HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK Set this bit to show no padding is required. (WO)

Register 19.17: HMAC_DATE_REG (0x00F8)



HMAC_DATE Version control register. (R/W)

20. Digital Signature (DS)

20.1 Overview

Digital signatures provide a way to cryptographically authenticate a message using a private key, to be verified using the corresponding public key. This can be used to validate a device's identity to a server, or to authenticate the integrity of a message has not been tampered with.

ESP32-S2 includes a digital signature (DS) peripheral which produces hardware accelerated RSA digital signatures, without the RSA private key being accessible by software.

20.2 Features

- RSA Digital Signatures with key lengths up to 4096 bits
- Private key data is encrypted and only readable by DS peripheral
- SHA-256 digest is used to protect private key data against tampering by an attacker

20.3 Functional Description

20.3.1 Overview

The DS peripheral calculates the RSA encryption operation $Z = X^Y \bmod M$ where Z is the signature, X is the input message, Y and M are the RSA private key parameters.

Private key parameters are stored in flash or another form of storage, in an encrypted form. They are encrypted using a key which can only be read by the DS peripheral via the HMAC peripheral. The required inputs to generate the key are stored in eFuse and can only be accessed by the HMAC peripheral. This means that only the DS peripheral hardware can decrypt the private key, and the plaintext private key data is never accessed by software.

The input message X is input directly to the DS peripheral by software, each time a signature is needed. After the operation, the signature Z is read back by software.

20.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They will have a particular RSA key length (up to 4096 bits). A corresponding public key is also generated and stored separately, it can be used independently to verify DS signatures.

Two additional private key operands are needed — $\overline{r}$ and M'. These two operands are derived from Y and M, but they are calculated in advance by software.

Operands Y, M, $\overline{r}$, and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is input to the DS peripheral in this encrypted format, then the hardware decrypts C and uses the key data to generate the signature. Detailed description of the encryption process to prepare C is provided in Section 20.3.4.

The DS peripheral needs to activate RSA to perform $Z = X^Y \bmod M$. For detailed information on the RSA algorithm, please refer to Section 18.3.1 *Large Number Modular Exponentiation* in Chapter 18 *RSA Accelerator (RSA)*.

20.3.3 Conventions

The following sections of this chapter will use the following symbols and functions:

- 1^s A bit string that consists of s "1" bits.
- $[x]_s$ A bit string of length s bits. If x is a number ($x < 2^s$), it is represented in little endian byte order in the bit string. x may be a variable value such as $[Y]_{4096}$ or as a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]$ is right-padded with 0s to reach s bits in length. For example: $[0x5]_4 = 0101$, $[0x5]_8 = 00000101$, $[0x5]_{16} = 0000010100000000$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$.
- $||$ A bit string concatenation operator for joining multiple bit strings into a longer bit string.

20.3.4 Software Storage of Private Key Data

To store a private key for use with the DS peripheral, users need to complete the following preparations:

- Generate the RSA private key (Y, M) and associated operands $\bar{r}$ and M', as described in Section 20.3.2.
- Generate a 256-bit HMAC key ($[HMAC_KEY]_{256}$) that is stored in eFuse. This HMAC key is read by the HMAC peripheral to derive a key, as DS_KEY = HMAC-SHA256($[HMAC_KEY]_{256}$, 1^{256}). This key is used to securely encrypt and decrypt the stored RSA private key data. For more information, please refer to Chapter 19 *HMAC Accelerator (HMAC)*.
- Prepare encrypted private key parameters as ciphertext C, 1584 bytes in length.

Figure 20-1 below describes the preparations at the software level (the left part) and the DS peripheral operation at the hardware level (the right part).

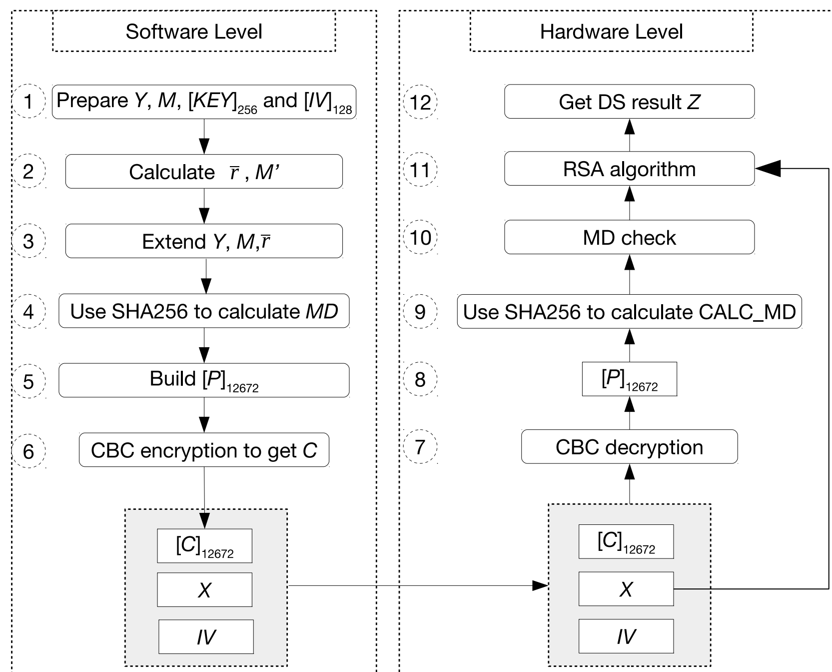


Figure 20-1. Preparations and DS Operation

Users need to follow the steps shown in the left part of Figure 20-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare Y and M whose lengths should meet the aforementioned requirements. Define $[L]_{32} = \frac{N}{32}$ (i.e., for RSA 4096, $[L]_{32}$ == $[0x80]_{32}$). Prepare $[DS_KEY]_{256}$ and generate a random $[IV]_{128}$ which

should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 17 *AES Accelerator (AES)*.

- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$, in order to get $[Y]_{4096}$, $[M]_{4096}$, and $[\overline{r}]_{4096}$, respectively. Since the largest operand length for Y, M, and $\overline{r}$ is 4096 bits, this step is only required for lengths smaller than 4096 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256 algorithm:

  $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$), where $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a 64-bit string $[\text{0x0808080808080808}]_{64}$ that is composed of eight bytes (value = 0x08). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.
- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[DS_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext that includes RSA operands Y, M, $\overline{r}$, M', and L as well as the MD authentication code and $[\beta]_{64}$. DS_KEY is derived from the $HMAC_KEY$ stored in eFuse, as described above in Section 20.3.4.

20.3.5 DS Operation at the Hardware Level

The hardware operation is triggered each time a Digital Signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV.

The DS operation at the hardware level is a reverse process of preparing C described in Section 20.3.4. The hardware operation can be divided into the following three stages.

1. **Decryption: Step 7 and 8**

   The decryption process is the reverse of Step 6. The DS peripheral will call AES accelerator to decrypt C in CBC block mode and get the resulted plaintext. The decryption process can be represented by P = AES-CBC-DEC (C, DS_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by users. $[DS_KEY]_{256}$ is provided by HMAC module, derived from $HMAC_KEY$ stored in eFuse. $[DS_KEY]_{256}$ is not readable by software. Please refer to Chapter 19 *HMAC Accelerator (HMAC)* for more information.

   With P, the DS peripheral can work out $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the reverse of Step 5.

2. **Check: Step 9 and 10**

   The DS peripheral will perform two check operations: MD check and padding check. Padding check is not shown in Figure 20-1, as it happens at the same time with MD check.

   - MD check: The DS peripheral calls SHA-256 to get the hash value $[CALC_MD]_{256}$. This step is the reverse of Step 4. Then, $[CALC_MD]_{256}$ is compared against $[MD]_{256}$. Only when the two match, MD check passes.
   - Padding check: The DS peripheral checks if $[\beta]_{64}$ complies with the aforementioned PKCS#7 format. Only when $[\beta]_{64}$ complies with the format, padding check passes.

   If MD check passes, the DS peripheral will perform subsequent operations, otherwise, it will not. If padding check fails, an error bit is set in the query register, but it does not affect the subsequent operations.

3. **Calculation: Step 11 and 12**

The DS peripheral treats X, Y, M, and $\bar{r}$ as big numbers. With M', all operands to perform X^Y mod M are in place. The operand length is defined by L. The DS peripheral will get the signed result Z by calling RSA to perform $Z = X^Y$ mod M.

20.3.6 DS Operation at the Software Level

The following software steps should be followed each time a Digital Signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV. These software steps trigger the hardware steps described in Section 20.3.5.

1. **Activate the DS peripheral**: Write 1 to DS_SET_START_REG.
2. **Check if DS_KEY is ready**: Poll DS_QUERY_BUSY_REG until it reads 0.

   If DS_QUERY_BUSY_REG does not read 0 after approximately 1 ms, it indicates a problem with HMAC initialization. In such case, software can read register DS_QUERY_KEY_WRONG_REG to get more information.

   - If DS_QUERY_KEY_WRONG_REG reads 0, it indicates that HMAC peripheral was not activated.
   - If DS_QUERY_KEY_WRONG_REG reads any value from 1 to 15, it indicates that HMAC was activated, but the DS peripheral did not successfully receive the DS_KEY value from the HMAC peripheral. This may indicate that the HMAC operation was interrupted due to a software concurrency problem.
3. **Configure register**: Write IV block to register DS_IV_*m*_REG (*m*: 0-3). For more information on IV block, please refer to Chapter 17 *AES Accelerator (AES)*.
4. **Write X to memory block DS_X_MEM**: Write X_i $(i \in [0, n) \cap \mathbb{N})$ to memory block DS_X_MEM whose capacity is 128 words. Each word can store one base-b digit. The memory block uses the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address. Words in DS_X_MEM block after the configured length of X (N bits, as described in Section 20.3.2) are ignored.
5. **Write C to memory block DS_C_MEM**: Write C_i $(i \in [0, 396) \cap \mathbb{N})$ to memory block DS_C_MEM whose capacity is 396 words. Each word can store one base-b digit.
6. **Start DS operation**: Write 1 to register DS_SET_ME_REG.
7. **Wait for the operation to be completed**: Poll register DS_QUERY_BUSY_REG until it reads 0.
8. **Query check result**: Read register DS_QUERY_CHECK_REG and determine the subsequent operations based on the return value.
   - If the value is 0, it indicates that both padding check and MD check pass. Users can continue to get the signed result Z.
   - If the value is 1, it indicates that the padding check passes but MD check fails. The signed result Z is invalid. The operation would resume directly from Step 10.
   - If the value is 2, it indicates that the padding check fails but MD check passes. Users can continue to get the signed result Z.
   - If the value is 3, it indicates that both padding check and MD check fail. The signed result Z is invalid. The operation would resume directly from Step 10.
9. **Read the signed result**: Read the signed result Z_i $(i \in \{0, 1, 2..., n\})$ from memory block DS_Z_MEM. The memory block stores Z in little-endian byte order.

10. **Exit the operation**: Write 1 to DS_SET_FINISH_REG, then poll DS_QUERY_BUSY_REG until it reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

20.4 Base Address

Users can access the DS peripheral with two base addresses, which can be seen in Table 119. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 119: Digital Signature Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43D000
PeriBUS2	0x6003D000

20.5 Memory Blocks

Both the starting address and ending address in the following table are relative to the DS peripheral base addresses provided in Section 20.4.

Table 120: Digital Signature Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
DS_C_MEM	Memory block C	1584	0x0000	0x062F	WO
DS_X_MEM	Memory block X	512	0x0800	0x09FF	WO
DS_Z_MEM	Memory block Z	512	0x0A00	0x0BFF	RO

20.6 Register Summary

The addresses in the following table are relative to the DS peripheral base addresses provided in Section 20.4.

Name	Description	Address	Access
Configuration Registers			
DS_IV_0_REG	IV block data	0x0630	WO
DS_IV_1_REG	IV block data	0x0634	WO
DS_IV_2_REG	IV block data	0x0638	WO
DS_IV_3_REG	IV block data	0x063C	WO
Status/Control Registers			
DS_SET_START_REG	Activates the DS peripheral	0x0E00	WO
DS_SET_ME_REG	Starts DS operation	0x0E04	WO
DS_SET_FINISH_REG	Ends DS operation	0x0E08	WO
DS_QUERY_BUSY_REG	Status of the DS	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	Checks the reason why DS_KEY is not ready	0x0E10	RO
DS_QUERY_CHECK_REG	Queries DS check result	0x0814	RO
Version Register			
DS_DATE_REG	Version control register	0x0820	W/R

20.7 Registers

Register 20.1: DS_IV_*m*_REG (*m*: 0-3) (0x0630+4*m)

31	0	
0x000000000		Reset

DS_IV_*m*_REG (*m*: 0-3) IV block data. (WO)

Register 20.2: DS_SET_START_REG (0x0E00)

(reserved)	DS_SET_START	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

DS_SET_START Write 1 to this register to activate the DS peripheral. (WO)

Register 20.3: DS_SET_ME_REG (0x0E04)

(reserved)	DS_SET_ME	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

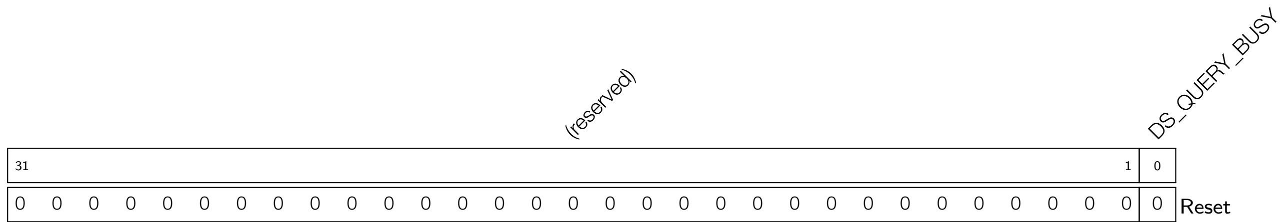
DS_SET_ME Write 1 to this register to start DS operation. (WO)

Register 20.4: DS_SET_FINISH_REG (0x0E08)

(reserved)	DS_SET_FINISH	
31 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

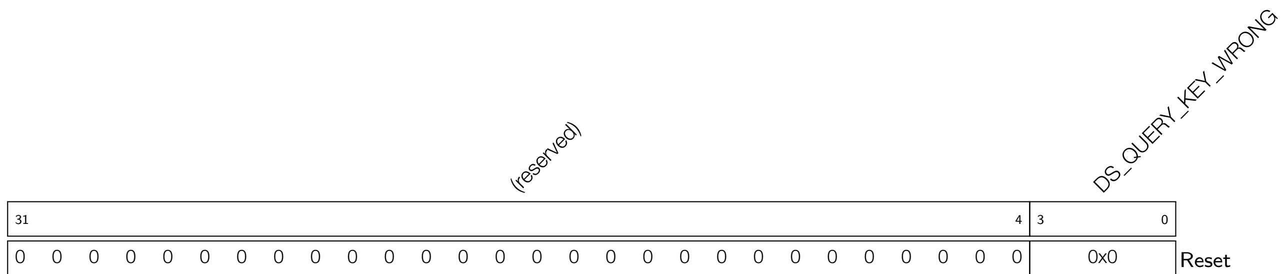
DS_SET_FINISH Write 1 to this register to end DS operation. (WO)

Register 20.5: DS_QUERY_BUSY_REG (0x0E0C)



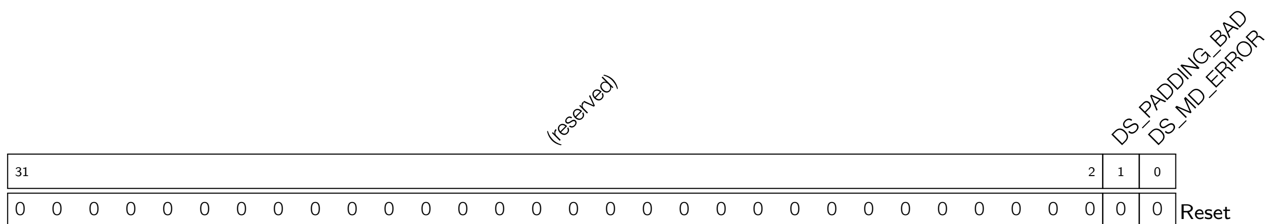
DS_QUERY_BUSY 1: The DS peripheral is busy; 0: The DS peripheral is idle. (RO)

Register 20.6: DS_QUERY_KEY_WRONG_REG (0x0E10)



DS_QUERY_KEY_WRONG 1-15: HMAC was activated, but the DS peripheral did not successfully receive the DS_KEY value from the HMAC peripheral. The biggest value is 15. 0: HMAC is not activated. (RO)

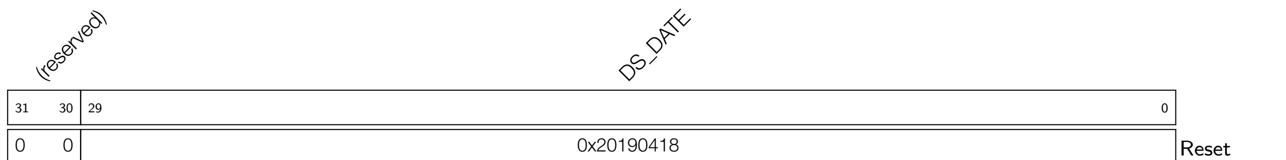
Register 20.7: DS_QUERY_CHECK_REG (0x0E14)



DS_PADDING_BAD 1: The padding check fails; 0: The padding check passes. (RO)

DS_MD_ERROR 1: The MD check fails; 0: The MD check passes. (RO)

Register 20.8: DS_DATE_REG (0x0E20)



DS_DATE Version control register. (R/W)

21. External Memory Encryption and Decryption (XTS_AES)

21.1 Overview

The ESP32-S2 SoC implements an External Memory Encryption and Decryption module that secures users' application code and data stored in the external memory (flash and external RAM). The encryption and decryption algorithm complies with the XTS-AES standard specified in IEEE Std 1619-2007. Users can store proprietary firmware and sensitive data (for example credentials for gaining access to a private network) to the external flash, and general data to the external RAM.

21.2 Features

- General XTS-AES algorithm, compliant with IEEE Std 1619-2007
- Software-based manual encryption
- High-speed hardware auto encryption
- High-speed hardware auto decryption
- Encryption and decryption functions jointly determined by register configuration, eFuse parameters, and boot mode

21.3 Functional Description

The External Memory Encryption and Decryption module consists of three blocks, namely the Manual Encryption block, Auto Encryption block, and Auto Decryption block. The module architecture is shown in Figure 21-1.

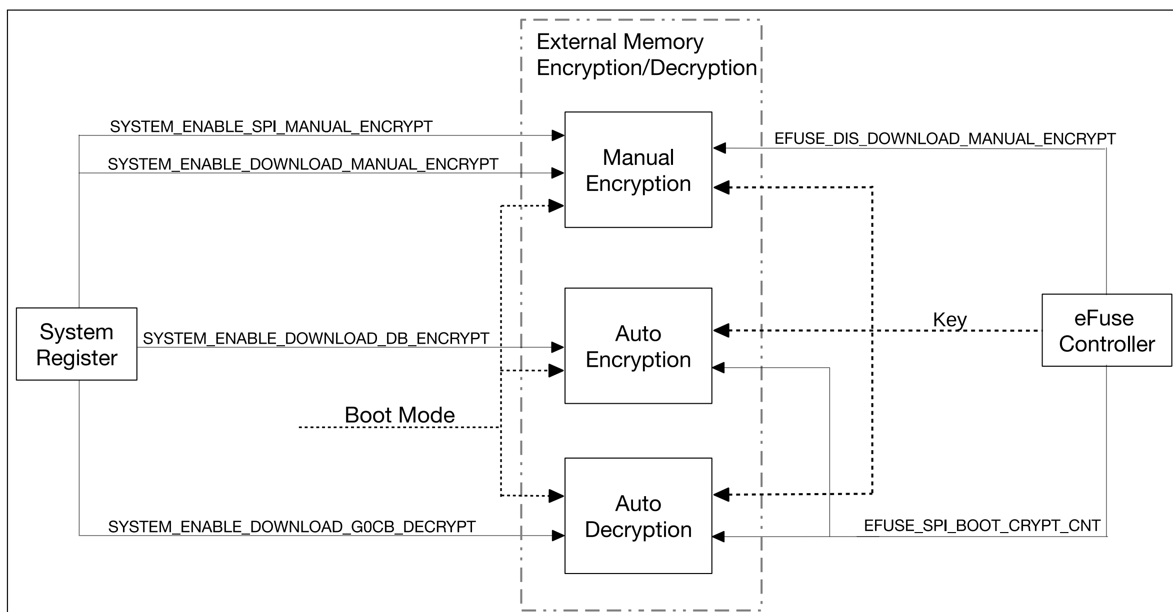


Figure 21-1. Architecture of the External Memory Encryption and Decryption Module

The Manual Encryption block can encrypt instructions and data which will then be written to the external flash as ciphertext through SPI1.

When the CPU writes the external RAM through cache, the Auto Encryption block automatically encrypts the

data first, and the data is written to the external RAM as ciphertext.

When the CPU reads the external flash or RAM through cache, the Auto Decryption block automatically decrypts the ciphertext to retrieve instructions and data.

In the peripheral System Register, four bits in the SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG register are relevant to external memory encryption and decryption:

- SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

The External Memory Encryption and Decryption module fetches two parameters from the peripheral eFuse Controller. These parameters are: EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT and EFUSE_SPI_BOOT_CRYPT_CNT.

21.3.1 XTS Algorithm

The manual encryption, auto encryption, and auto decryption operations use the same algorithm, i.e., XTS algorithm. In real-life implementation, the XTS algorithm is characterized by "data unit" of 1024 bits. The "data unit" is defined in the *XTS-AES Tweakable Block Cipher* standard, section *XTS-AES encryption procedure*. More information on the XTS-AES algorithm can be found in IEEE Std 1619-2007.

21.3.2 Key

The Manual Encryption block, Auto Encryption block, and Auto Decryption block share the same key to perform XTS algorithm. The key is provided by the eFuse hardware and protected from user access.

The key can be either 256 bits or 512 bits long. The key is determined by the content in one or two eFuse blocks from BLOCK4 ~ BLOCK9. For easy description, define:

- Block$_A$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1. Block$_A$ contains 256-bit Key_A.
- Block$_B$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2. Block$_B$ contains 256-bit Key_B.
- Block$_C$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_128_KEY. Block$_C$ contains 256-bit Key_C.

Table 122 shows how the Key is generated, depending on whether Block$_A$, Block$_B$, and Block$_C$ exists or not.

Table 122: Key

Block$_A$	Block$_B$	Block$_C$	Key	Key Length (bit)
Yes	Yes	Don't care	$Key_A \| \| Key_B$	512
Yes	No	Don't care	$Key_A \| \| 0^{256}$	512
No	Yes	Don't care	$0^{256} \| \| Key_B$	512
No	No	Yes	Key_C	256
No	No	No	0^{256}	256

"Yes" indicates that the block exists; "No" indicates that the block does not exist; "0^{256}" indicates a bit string that consists of 256-bit zeros; "||" is a bonding operator for joining one key string to another.

For more information on setting of key purposes, please refer to Chapter 4 *eFuse Controller (eFuse)*.

21.3.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory where the encrypted result is stored. The target memory space can be uniquely determined by three relevant parameters: $type$, $size$, and $base_addr$. They are defined as follows:

- $type$: the type of the external memory, either flash or external RAM. Value 0 indicates flash, 1 indicates external RAM.
- $size$: the size of the target memory space, in unit of bytes. One single encryption operation supports either 16, 32, or 64 bytes of data.
- $base_addr$: the base address of the target memory space. It is a physical address aligned to $size$, i.e., $base_addr \% size == 0$.

Assume encrypted 16 bytes written to address 0x130 ~ 0x13F in the external flash, then, the target memory space is 0x130 ~ 0x13F, $type$ is 0 (flash), $size$ is 16 (bytes), and $base_addr$ 0x130.

The encryption of any length (must be multiples of 16 bytes) of data can be completed separately in multiple operations. Each operation can have individual target memory space and the relevant parameters.

For auto encryption and auto decryption, these parameters are automatically defined by hardware. For manual encryption, these parameters should be configured manually by users.

21.3.4 Data Padding

For auto encryption and auto decryption, data padding is automatically completed by hardware. For manual encryption, data padding should be completed manually by users. The Manual Encryption block is equipped with 16 registers, i.e., XTS_AES_PLAIN_*n*_REG (*n*: 0-15), that are dedicated to data padding and can store up to 512 bits of plaintext at a time.

Actually, the Manual Encryption block does not care where the plaintext comes from, but only where the ciphertext is to be stored. Because of the strict correspondence between plaintext and ciphertext, in order to better describe how the plaintext is stored in the register heap, it is assumed that the plaintext is stored in the target memory space in the first place and replaced by ciphertext after encryption. Therefore, the following description no longer has the concept of "plaintext", but uses "target memory space" instead. However, users should note that the plaintext can come from anywhere, and that they should understand how the plaintext is stored in the register heap.

How mapping works between target memory space and registers:

Assume a word is stored in $address$, define $offset = address\%64$, $n = \frac{offset}{4}$, then the word will be stored in register XTS_AES_PLAIN_n_REG.

For example, if the $size$ of the target memory space is 64, then all the 16 registers will be used for data storage. The mapping between $offset$ and registers is shown in Table 123.

$offset$	Register	$offset$	Register

Table 123: Mapping Between Offsets and Registers

$offset$	Register	$offset$	Register
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

21.3.5 Manual Encryption Block

The Manual Encryption block is a peripheral module. It is equipped with registers that can be accessed by the CPU directly. Registers embedded in this block, System registers, eFuse parameters, and boot mode jointly configure and control this block. Please note that currently the Manual Encryption block can only encrypt flash.

The manual encryption requires software participation. The steps are as follows:

1. Configure XTS_AES:

   - Set XTS_AES_DESTINATION_REG register to $type$ = 0.
   - Set XTS_AES_PHYSICAL_ADDRESS_REG register to $base_addr$.
   - Set XTS_AES_LINESIZE_REG register to $\frac{size}{32}$.

   For definitions of $type$, $base_addr$, $size$, please refer to Section 21.3.3.

2. Fill registers XTS_AES_PLAIN_*n*_REG (*n*: 0-15) in with plaintext (refer to Section 21.3.4). Registers that are not used can be written into any value.

3. Poll XTS_AES_STATE_REG until it reads 0 that indicates the Manual Encryption block is idle.

4. Activate encryption by writing 1 to XTS_AES_TRIGGER_REG register.

5. Wait for the encryption to complete. Poll register XTS_AES_STATE_REG until it reads 2.
   Steps 1 ~ 5 complete the encryption operation, where Key is used.

6. Grant SPI1 access to the encrypted result by writing 1 to XTS_AES_RELEASE_REG register.
   XTS_AES_STATE_REG will read 3 afterwards.

7. Call SPI1 and write the encrypted result to the external flash.

8. Destroy the encrypted result by writing 1 to XTS_AES_DESTROY_REG. XTS_AES_STATE_REG register will read 0 afterwards.

Repeat the steps above to complete multiple encryption operations.

The Manual Encryption block is operational only with granted permission. The operating conditions are:

- In SPI Boot mode
  If bit SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Manual Encryption block is granted permission. Otherwise, it is not operational.
- In Download Boot mode
  If bit SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1 and the eFuse parameter EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT is 0, the Manual Encryption block is granted permission. Otherwise, it is not operational.

Note:

- Even though the CPU can skip cache and get the encrypted result directly by reading the external memory, software can by no means access Key.
- The Manual Encryption block needs to call the AES accelerator to perform encryption. Therefore, users cannot access AES accelerator during the process.

21.3.6 Auto Encryption Block

The Auto Encryption block is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access this block. The System Register, eFuse parameters, and boot mode jointly control this block.

The Auto Encryption block is operational only with granted permission. The operating conditions are:

- In SPI Boot mode
  If the 3-bit parameter SPI_BOOT_CRYPT_CNT has 1 or 3 bits set to 1, then the Auto Encryption block is granted permission. Otherwise, it is not operational.
- In Download Boot mode
  If bit SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Encryption block is granted permission. Otherwise, it is not operational.

Note:

- When the Auto Encryption block is operational, the CPU will read data from the external RAM via the cache. The Auto Encryption block automatically encrypts the data and write it to the external RAM. The entire encryption process does not need software participation and is transparent to the cache. Software cannot access the encryption Key.
- When the Auto Encryption block is not operational, it will ignore the CPU's request to access cache and do not process the data. Therefore, data will be written to the external RAM as plaintext.

21.3.7 Auto Decryption Block

The Auto Decryption block is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access this block. The System Register, eFuse parameters, and boot mode jointly control and configure this block.

The Auto Decryption block is operational only with granted permission. The operating conditions are:

- In SPI Boot mode
  If the 3-bit parameter SPI_BOOT_CRYPT_CNT has 1 or 3 bits set to 1, then the Auto Decryption block is granted permission. Otherwise, it is not operational.
- In Download Boot mode
  If bit SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Decryption block is granted permission. Otherwise, it is not operational.

Note:

- When the Auto Decryption block is operational, the CPU will read instructions and data from the external memory via cache. The Auto Decryption block automatically decrypts and retrieves the instructions and data. The entire decryption process does not need software participation and is transparent to the cache. Software cannot access the decryption Key.
- When the Auto Decryption block is not operational, it does not have any effect on the contents stored in the external memory, be they encrypted or unencrypted. What the CPU reads via cache is the original information stored in the external memory.

21.4 Base Address

Users can access the Manual Encryption block with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 124: Manual Encryption Block Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

21.5 Register Summary

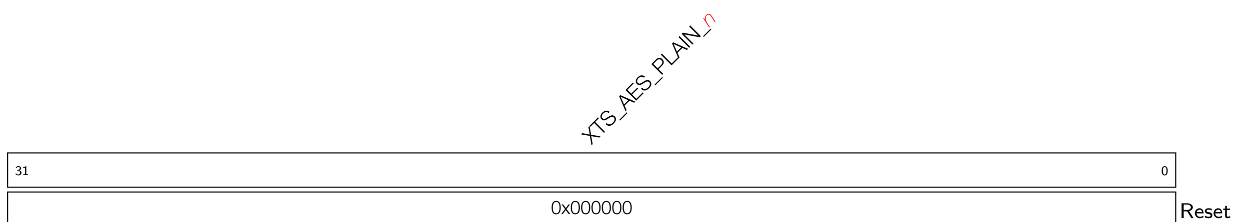
The addresses in the following table are relative to the Manual Encryption block's base addresses provided in Section 21.4.

Name	Description	Address	Access
Plaintext Register Heap			
XTS_AES_PLAIN_0_REG	Plaintext register 0	0x0100	R/W
XTS_AES_PLAIN_1_REG	Plaintext register 1	0x0104	R/W
XTS_AES_PLAIN_2_REG	Plaintext register 2	0x0108	R/W
XTS_AES_PLAIN_3_REG	Plaintext register 3	0x010C	R/W
XTS_AES_PLAIN_4_REG	Plaintext register 4	0x0110	R/W
XTS_AES_PLAIN_5_REG	Plaintext register 5	0x0114	R/W
XTS_AES_PLAIN_6_REG	Plaintext register 6	0x0118	R/W
XTS_AES_PLAIN_7_REG	Plaintext register 7	0x011C	R/W
XTS_AES_PLAIN_8_REG	Plaintext register 8	0x0120	R/W

Name	Description	Address	Access
XTS_AES_PLAIN_9_REG	Plaintext register 9	0x0124	R/W
XTS_AES_PLAIN_10_REG	Plaintext register 10	0x0128	R/W
XTS_AES_PLAIN_11_REG	Plaintext register 11	0x012C	R/W
XTS_AES_PLAIN_12_REG	Plaintext register 12	0x0130	R/W
XTS_AES_PLAIN_13_REG	Plaintext register 13	0x0134	R/W
XTS_AES_PLAIN_14_REG	Plaintext register 14	0x0138	R/W
XTS_AES_PLAIN_15_REG	Plaintext register 15	0x013C	R/W
Configuration Registers			
XTS_AES_LINESIZE_REG	Configures the size of target memory space	0x0140	R/W
XTS_AES_DESTINATION_REG	Configures the type of the external memory	0x0144	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	Physical address	0x0148	R/W
Control/Status Registers			
XTS_AES_TRIGGER_REG	Activates AES algorithm	0x014C	WO
XTS_AES_RELEASE_REG	Release control	0x0150	WO
XTS_AES_DESTROY_REG	Destroys control	0x0154	WO
XTS_AES_STATE_REG	Status register	0x0158	RO
Version Register			
XTS_AES_DATE_REG	Version control register	0x015C	RO

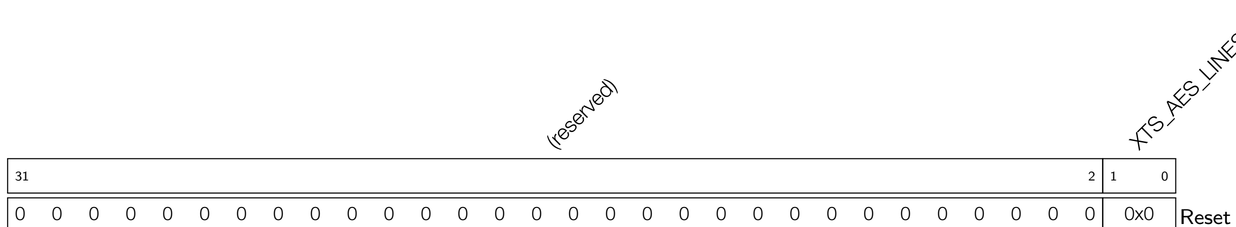
21.6 Registers

Register 21.1: XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0100+4n*)**



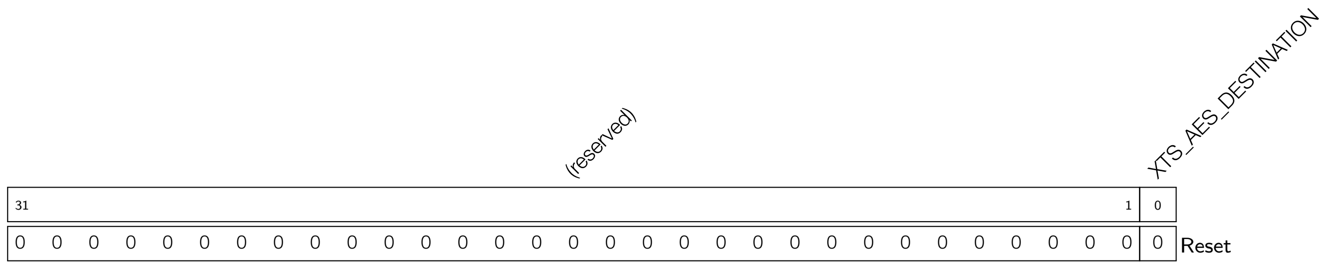
XTS_AES_PLAIN_*n* This register stores *n*th 32-bit piece of plaintext. (R/W)

Register 21.2: XTS_AES_LINESIZE_REG (0x0140)



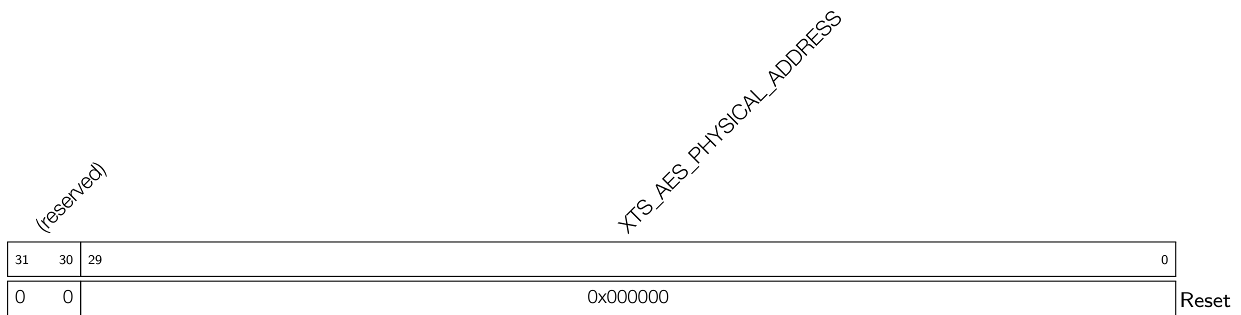
XTS_AES_LINESIZE Configures the data size of a single encryption. 0: 128 bits; 1: 256 bits; 2: 512 bits. (R/W)

Register 21.3: XTS_AES_DESTINATION_REG (0x0144)



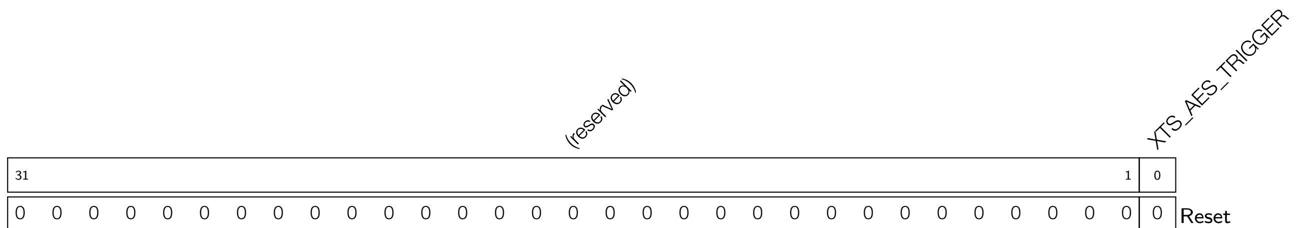
XTS_AES_DESTINATION Configures the type of the external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Errors may occur if users write 1. 0: flash; 1: external RAM. (R/W)

Register 21.4: XTS_AES_PHYSICAL_ADDRESS_REG (0x0148)



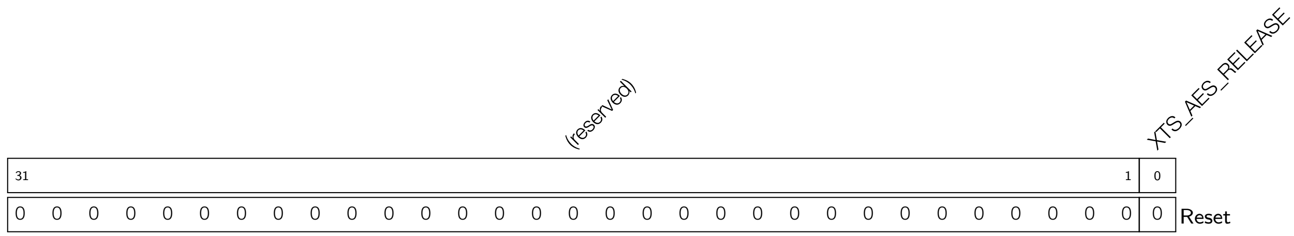
XTS_AES_PHYSICAL_ADDRESS Physical address. (R/W)

Register 21.5: XTS_AES_TRIGGER_REG (0x014C)



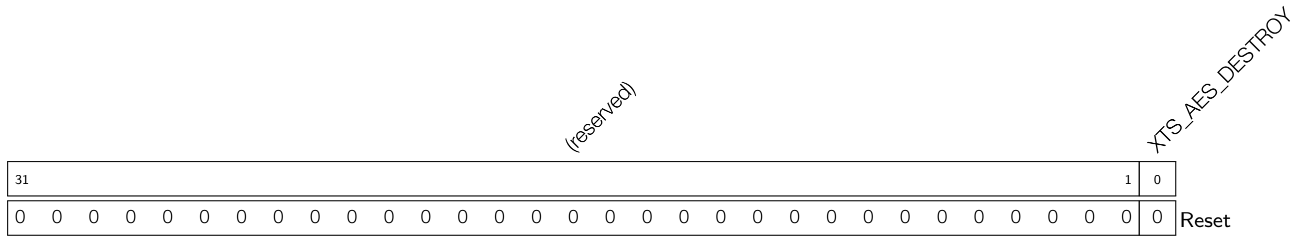
XTS_AES_TRIGGER Set to enable manual encryption. (WO)

Register 21.6: XTS_AES_RELEASE_REG (0x0150)



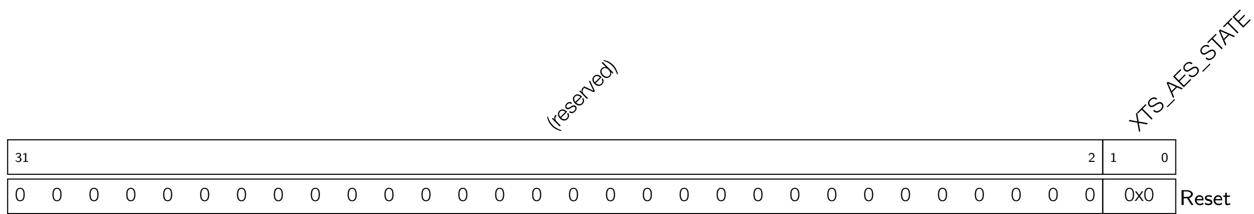
XTS_AES_RELEASE Set to grant SPI1 access to encrypted result. (WO)

Register 21.7: XTS_AES_DESTROY_REG (0x0154)



XTS_AES_DESTROY Set to destroy encrypted result. (WO)

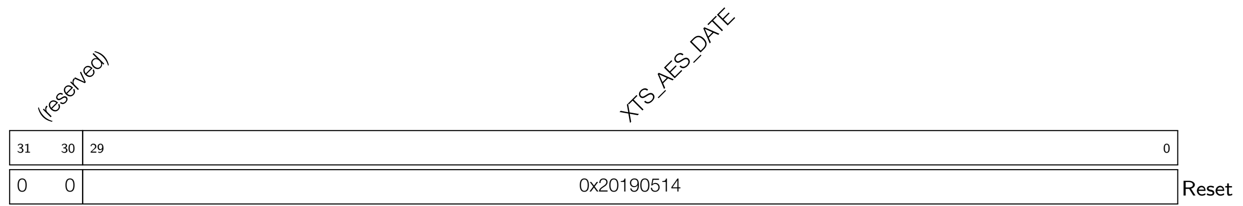
Register 21.8: XTS_AES_STATE_REG (0x0158)



XTS_AES_STATE Indicates the status of the Manual Encryption block. (RO)

- 0x0 (XTS_AES_IDLE): idle;
- 0x1 (XTS_AES_BUSY): busy with encryption;
- 0x2 (XTS_AES_DONE): encryption is completed, but the encrypted result is not accessible to SPI;
- 0x3 (XTS_AES_RELEASE): encrypted result is accessible to SPI.

Register 21.9: XTS_AES_DATE_REG (0x015C)



XTS_AES_DATE Version control register. (RO)

22. Random Number Generator (RNG)

22.1 Introduction

The ESP32-S2 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

22.2 Features

The random number generator generates true random numbers, which means random number generated from a physical process, rather than by means of an algorithm. No number generated within the specified range is more or less likely to appear than any other number.

22.3 Functional Description

Every 32-bit value that the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the thermal noise in the system and the asynchronous clock mismatch.

Thermal noise comes from the high-speed ADC or SAR ADC or both. Whenever the high-speed ADC or SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.

When the RTC8M_CLK clock is enabled for the digital core, the random number generator will also sample RTC8M_CLK (8 MHz) as a random bit seed. RTC8M_CLK is an asynchronous clock source and it increases the RNG entropy by introducing circuit metastability. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

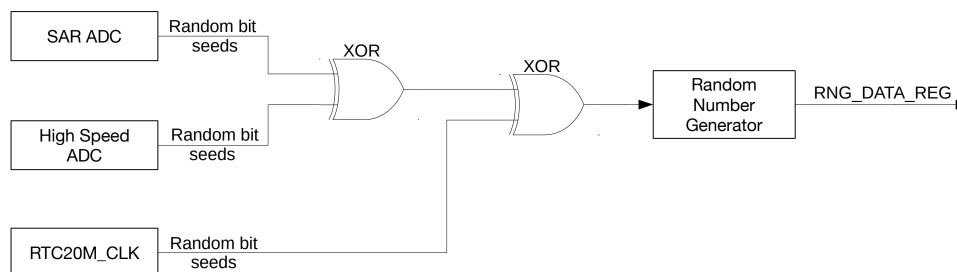


Figure 22-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle of RTC8M_CLK (8 MHz), which is generated from an internal RC oscillator (see Chapter 6 *Reset and Clock* for details). Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 500 kHz to obtain the maximum entropy.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1).

The sample passed all tests.

22.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC, high-speed ADC, or RTC8M_CLK is enabled. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 32 *On-Chip Sensor and Analog Signal Processing*.
- High-speed ADC is enabled automatically when the Wi-Fi module is enabled.
- RTC8M_CLK is enabled by setting the RTC_CNTL_DIG_CLK8M_EN bit in the RTC_CNTL_CLK_CONF_REG register.

Note:
Note that, when the Wi-Fi module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.

When using the random number generator, read the RNG_DATA_REG register multiple times until sufficient random numbers have been generated. Ensure the rate at which the register is read does not exceed the frequencies described in section 22.3 above.

22.5 Base Address

Users can access the random number generator with two base addresses, which can be seen in Table 126. For more information about accessing peripherals from different buses, please see Chapter 3 *System and Memory*.

Table 126: Random Number Generator Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F435000
PeriBUS2	0x60035000

22.6 Register Summary

The addresses in the following table are relative to the random number generator base addresses provided in Section 22.5.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x0110	RO

22.7 Register

The address in this section is relative to the random number generator base addresses provided in Section 22.5.

Register 22.1: RNG_DATA_REG (0x0110)

31	0	
0x00000000		Reset

RNG_DATA Random number source. (RO)

23. UART Controller (UART)

23.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full-duplex mode. ESP32-S2 has two UART controllers compatible with various UART devices. They support Infrared Data Association (IrDA) and RS485 transmission.

ESP32-S2 has two UART controllers. Each has a group of registers that function identically. In this chapter, the two UART controllers are referred to as UART*n*, in which *n* denotes 0 or 1.

23.2 Features

Each UART controller has the following features:

- Programmable baud rate
- 512 x 8-bit RAM shared by TX FIFOs and RX FIFOs of two UART controllers
- Full-duplex asynchronous communication
- Automatic baud rate detection
- Data bits ranging from 5 to 8
- Stop bits whose length can be 1, 1.5, 2 or 3 bits
- Parity bits
- Special character AT_CMD detection
- RS485 protocol
- IrDA protocol
- High-speed data communication using DMA
- UART as wake-up source
- Software and hardware flow control

23.3 Functional Description

23.3.1 UART Introduction

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit and parity bit.

A UART data packet usually begins with one start bit, followed by data bits, one parity bit (optional) and one or more stop bits. UART controllers on ESP32-S2 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as DMA for seamless high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

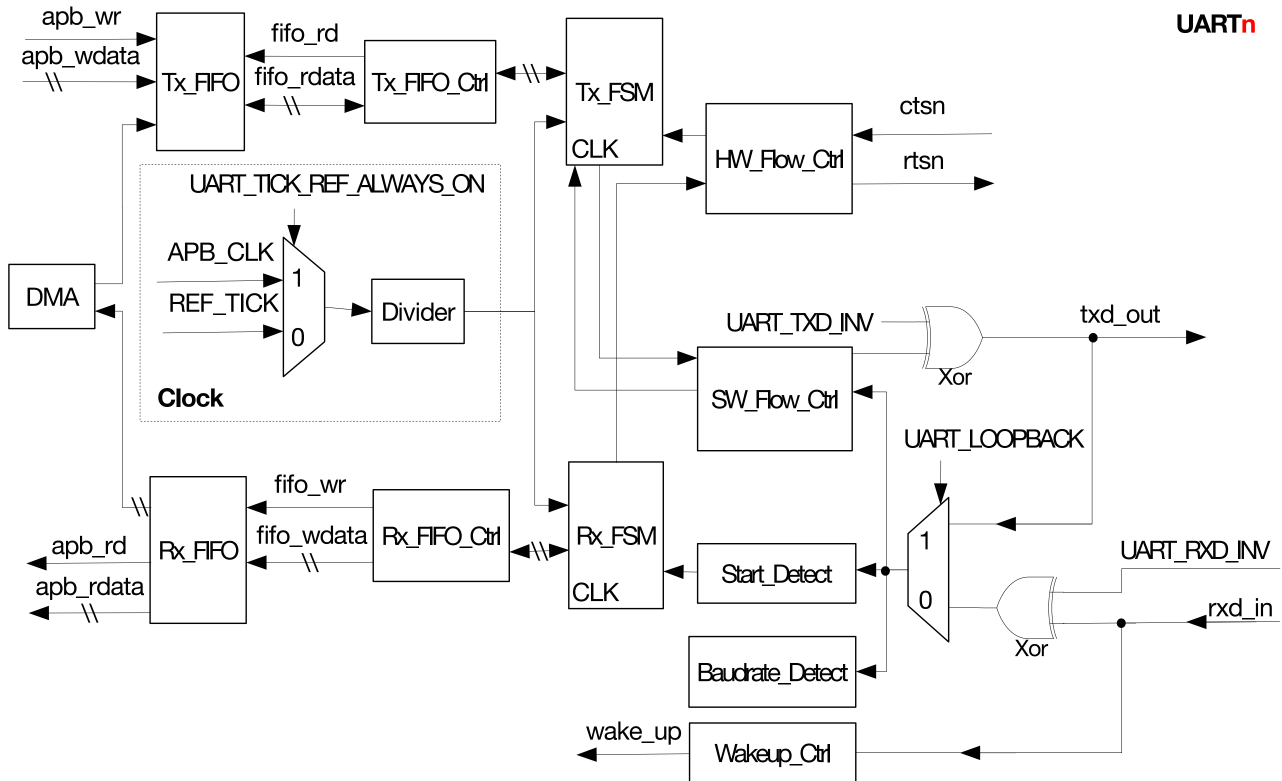


Figure 23-1. UART Structure

23.3.2 UART Structure

Figure 23-1 shows the basic structure of a UART controller. It has two possible clock sources: a 80 MHz APB_CLK and a reference clock REF_TICK (for details, please refer to Chapter 6 *Reset and Clock*), which are selected by configuring UART_TICK_REF_ALWAYS_ON. The selected clock source is divided by a divider to generate clock signals that drive the UART controller. The divisor is configured by UART_CLKDIV_REG: UART_CLKDIV for the integral part, and UART_CLKDIV_FRAG for the fractional part.

A UART controller is broken down into two parts according to functions: a transmitter and a receiver.

The transmitter contains a TX FIFO, which buffers data to be sent. Software can write data to Tx_FIFO via the APB bus, or move data to Tx_FIFO using DMA. Tx_FIFO_Ctrl controls writing and reading Tx_FIFO. When Tx_FIFO is not empty, Tx_FSM reads bytes via Tx_FIFO_Ctrl, and converts them into a bitstream. The levels of output signal txd_out can be inverted by configuring UART_TXD_INV register.

The receiver contains a RX FIFO, which buffers data to be processed. Software can read data from Rx_FIFO via the APB bus, or receive data using DMA. The levels of input signal rxd_in can be inverted by configuring UART_RXD_INV register, and the signal is then input to the Rx components of the UART Controller: Baudrate_Detect measures the baud rate of input signal rxd_in by detecting its minimum pulse width. Start_Detect detects the start bit in a data frame. If the start bit is detected, Rx_FSM stores data bits in the data frame into Rx_FIFO by Rx_FIFO_Ctrl.

HW_Flow_Ctrl controls rxd_in and txd_out data flows by standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls data flows by automatically adding special characters to outgoing data and detecting special characters in incoming data. When a UART controller is in Light-sleep mode (see Chapter 9: *Low-Power Management (RTC_CNTL)* for more details), Wakeup_Ctrl counts up rising edges of rxd_in. When the number reaches (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC,

which then wakes up the ESP32-S2 chip.

23.3.3 UART RAM

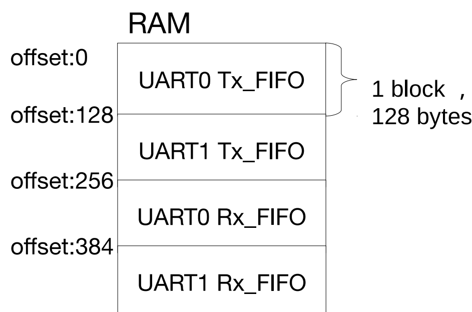


Figure 23-2. UART Controllers Sharing RAM

The two UART controllers on ESP32-S2 share 512 × 8 bits of FIFO RAM. As figure 23-2 illustrates, RAM is divided into 4 blocks, each has 128 × 8 bits. Figure 23-2 shows by default how many RAM blocks are allocated to TX FIFOs and RX FIFOs of the two UART controllers. UART*n* Tx_FIFO can be expanded by configuring UART_TX_SIZE, while UART*n* Rx_FIFO can be expanded by configuring UART_RX_SIZE. The size of UART0 Tx_FIFO can be increased to 4 blocks (the whole RAM), the size of UART1 Tx_FIFO can be increased to 3 blocks (from offset 128 to the end address), the size of UART0 Rx_FIFO can be increased to 2 blocks (from offset 256 to the end address), but the size of UART1 Rx_FIFO cannot be increased. Please note that expanding one FIFO may take up the default space of other FIFOs. For example, by setting UART_TX_SIZE of UART0 to 2, the size of UART0 Tx_FIFO is increased by 128 bytes (from offset 0 to offset 255). In this case, UART0 Tx_FIFO takes up the default space for UART1 Tx_FIFO, and UART1's transmitting function cannot be used as a result.

When neither of the two UART controllers is active, RAM could enter low-power mode by setting UART_MEM_FORCE_PD.

UART0 Tx_FIFO and UART1 Tx_FIFO are reset by setting UART_TXFIFO_RST. UART0 Rx_FIFO and UART1 Rx_FIFO are reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using DMA, read automatically and converted from a frame into a bitstream by hardware Tx_FSM; data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using DMA. The two UART controllers share one DMA controller.

The empty signal threshold for Tx_FIFO is configured by setting UART_TXFIFO_EMPTY_THRHD. When data stored in Tx_FIFO is less than UART_TXFIFO_EMPTY_THRHD, a UART_TXFIFO_EMPTY_INT interrupt is generated.

The full signal threshold for Rx_FIFO is configured by setting UART_RXFIFO_FULL_THRHD. When data stored in Rx_FIFO is greater than UART_RXFIFO_FULL_THRHD, a UART_RXFIFO_FULL_INT interrupt is generated. In addition, when Rx_FIFO receives more data than its capacity, a UART_RXFIFO_OVF_INT interrupt is generated.

UART*n* can access FIFO via register UART_FIFO_REG.

23.3.4 Baud Rate Generation and Detection

23.3.4.1 Baud Rate Generation

Before a UART controller sends or receives data, the baud rate should be configured by setting corresponding registers. A UART Controller baud rate generator functions by dividing the input clock source. It can divide the clock source by a fractional amount. The divisor is configured by UART_CLKDIV_REG: UART_CLKDIV for the integral part, and UART_CLKDIV_FRAG for the fractional part. When using an 80 MHz input clock, the UART controller supports a maximum baud rate of 5 Mbaud.

The divisor of the baud rate divider is equal to UART_CLKDIV + (UART_CLKDIV_FRAG / 16), meaning that the final baud rate is equal to INPUT_FREQ / (UART_CLKDIV + (UART_CLKDIV_FRAG / 16)). For example, if UART_CLKDIV = 694 and UART_CLKDIV_FRAG = 7 then the divisor value is (694 + 7/16) = 694.4375. If the input clock frequency is 80MHz APB_CLK, the baud rate will be (80MHz / 69.4375) = 115201.

When UART_CLKDIV_FRAG is zero, the baud rate generator is an integer clock divider where an output pulse is generated every UART_CLKDIV input pulses.

When UART_CLKDIV_FRAG is not zero, the divider is fractional and the output baud rate clock pulses are not strictly uniform. As shown in figure 23-3, for every 16 output pulses, the generator divides either (UART_CLKDIV + 1) input pulses or UART_CLKDIV input pulses per output pulse. A total of UART_CLKDIV_FRAG output pulses are generated by dividing (UART_CLKDIV + 1) input pulses, and the remaining (16 - UART_CLKDIV_FRAG) output pulses are generated by dividing UART_CLKDIV input pulses.

The output pulses are interleaved as shown in figure 23-3 below, to make the output timing more uniform:

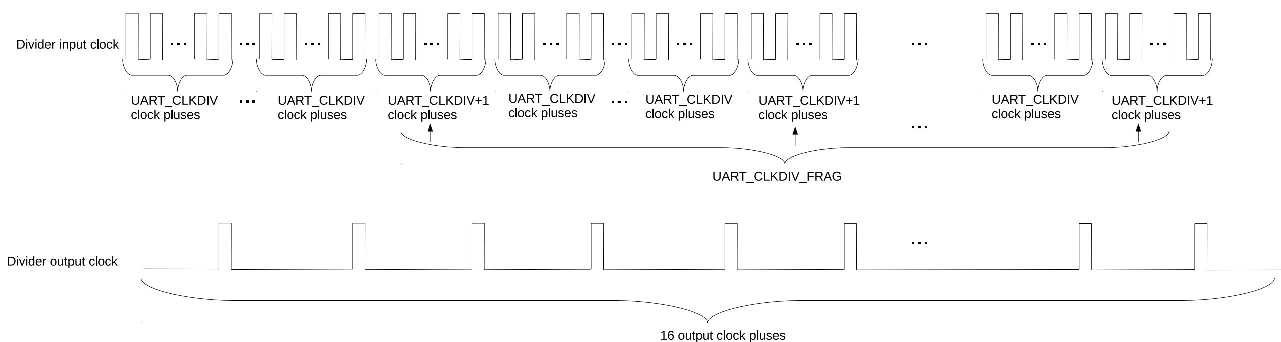


Figure 23-3. UART Controllers Division

To support IrDA (see Section 23.3.7 *IrDA* for details), the fractional clock divider for IrDA data transmission generates clock signals divided by 16× UART_CLKDIV_REG. This divider works similarly as the one elaborated above: it takes UART_CLKDIV/16 as the integer value and the lowest four bits of UART_CLKDIV as the fractional value.

23.3.4.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in figure 23-1 will measure pulse widths while filtering any noise whose pulse width is shorter than UART_GLITCH_FILT.

Before communication starts, the transmitter could send random data to the receiver for baud rate detection. UART_LOWPULSE_MIN_CNT stores the minimum low pulse width, UART_HIGHPULSE_MIN_CNT stores the minimum high pulse width, UART_POSEDGE_MIN_CNT stores the minimum pulse width between two rising edges, and UART_NEGEDGE_MIN_CNT stores the minimum pulse width between two falling edges. These four

registers are read by software to determine the transmitter's baud rate.

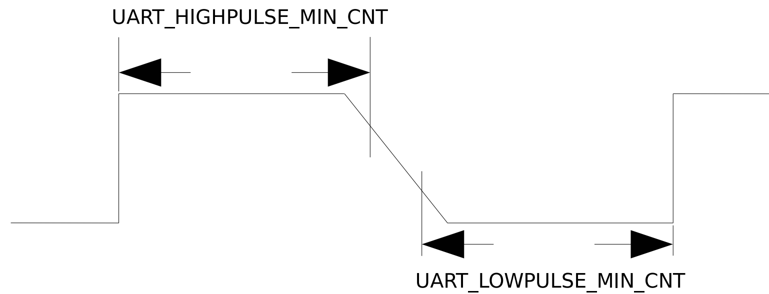


Figure 23-4. The Timing Diagram of Weak UART Signals Along Falling Edges

Baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or falling edges in semi-stable state, which results in inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a weighted average of these two values to eliminate errors. In this case, baud rate is calculated as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_LOWPULSE_MIN_CNT} + \text{UART_HIGHPULSE_MIN_CNT} + 2)/2}$$

2. If UART signals are weak along falling edges as shown in figure 23-4, which leads to inaccurate average of UART_LOWPULSE_MIN_CNT and UART_HIGHPULSE_MIN_CNT, use UART_POSEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_POSEDGE_MIN_CNT} + 1)/2}$$

3. If UART signals are weak along rising edges, use UART_NEGEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_NEGEDGE_MIN_CNT} + 1)/2}$$

23.3.5 UART Data Frame

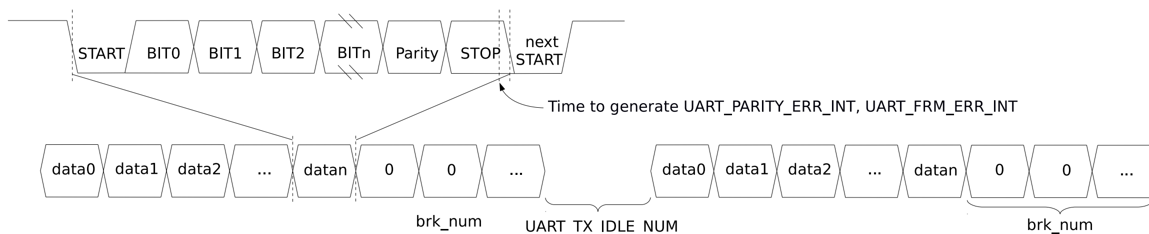


Figure 23-5. Structure of UART Data Frame

Figure 23-5 shows the basic structure of a data frame. A frame starts with one START bit, and ends with STOP bits which can be 1, 1.5, 2 or 3 bits long, configured by UART_STOP_BIT_NUM, UART_DL1_EN and UART_DL0_EN. The START bit is logical low, whereas STOP bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in data received, a UART_PARITY_ERR_INT interrupt is

generated and data received is still stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set then the transmitter will send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK
_IDLE_DONE_INT interrupt is generated.

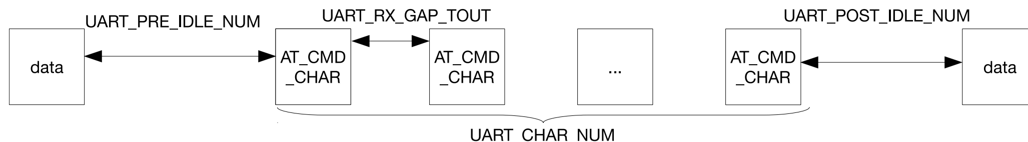


Figure 23-6. AT_CMD Character Structure

Figure 23-6 is the structure of a special character AT_CMD. If the receiver constantly receives AT_CMD_CHAR and the following conditions are met, a UART_AT_CMD_CHAR_DET_INT interrupt is generated.

- The interval between the first AT_CMD_CHAR and the last non AT_CMD_CHAR character is at least UART _PRE_IDLE_NUM single-bit cycles.
- The interval between two AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT single-bit cycles.
- The number of AT_CMD_CHAR characters is equal to or greater than UART_CHAR_NUM.
- The interval between the last AT_CMD_CHAR character and next non AT_CMD_CHAR character is at least UART_POST_IDLE_NUM single-bit cycles.

23.3.6 RS485

The two UART controllers support RS485 standard. This standard uses differential signals to transmit data, so it can communicate over longer distances at higher bit rates than RS232. RS485 has two-wire half-duplex mode and four-wire full-duplex mode. UART controllers support two-wire half-duplex transmission and bus snooping. In a two-wire RS485 multidrop network, there can be 32 slaves at most.

23.3.6.1 Driver Control

As shown in figure 23-7, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion. A RS485 transceiver contains a driver and a receiver. When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver. When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

The UART receiver converts differential signals to single-ended signals via an external receiver. RE is the enable control signal for the receiver. When RE is 0, the receiver is enabled; when RE is 1, the receiver is disabled. If RE is configured as 0, the UART controller is allowed to snoop data on the bus, including data sent by itself.

DE can be controlled by either software or hardware. To reduce cost of software, in our design DE is controlled by hardware. As shown in figure 23-7, DE is connected to dtrn_out of UART (please refer to Section23.3.9.1 *Hardware Flow Control* for more details).

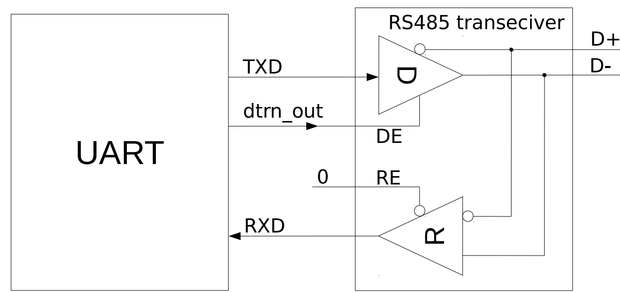


Figure 23-7. Driver Control Diagram in RS485 Mode

23.3.6.2 Turnaround Delay

By default, the two UART controllers work in receiver mode. When a UART controller is switched from transmitter mode to receiver mode, the RS485 protocol requires a turnaround delay of at least one cycle after the stop bit. The transmitter supports turnaround delay of two cycles added after the stop bit. When UART_DL1_EN is set, turnaround delay of one single-bit cycle is added; when UART_DL0_EN is set, turnaround delay of a second cycle is added.

23.3.6.3 Bus Snooping

By default, an RS485 device is not allowed to transmit and receive data simultaneously. However, the UART controller peripheral supports snooping this bus by receiving while transmitting. If UART_RS485TX_RX_EN is set and the external RS485 transceiver is configured as in figure 23-7, a UART controller may receive data in transmitter mode and snoop the bus. If UART_RS485RXBY_TX_EN is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop data sent by themselves. In transmitter mode, when a UART controller monitors a collision between data sent and data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitor a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a polarity error, a UART_RS485_PARITY_ERR_INT is generated.

23.3.7 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol and the link management protocol. The two UART controllers implement IrDA physical layer. In IrDA encoding, a UART controller supports data rates up to 115.2 kbit/s (SIR, or serial infrared mode). As shown in figure 23-8, the IrDA encoder converts a NRZ (non-return to zero code) signal to a RZI (return to zero code) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits to indicate logic “0”, and low levels to indicate logic “1”. The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the encoder output polarity is the opposite of the decoder input polarity. If a low pulse is detected, it indicates that a start bit has been received.

When IrDA function is enabled, one bit is divided into 16 clock cycles. If the bit to be sent is zero, then the 9th, 10th and 11th clock cycle is high.

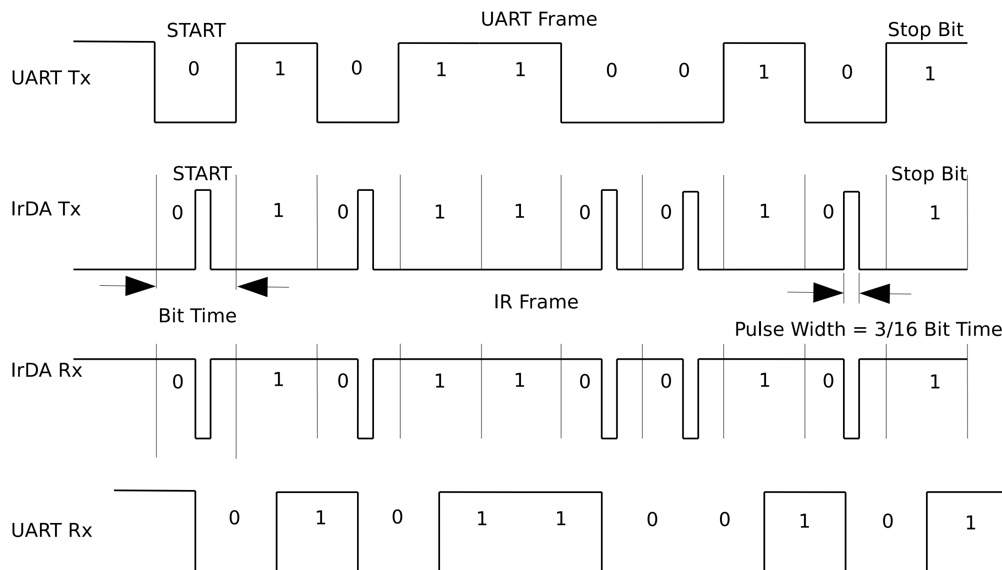


Figure 23-8. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in figure 23-9, IrDA function is enabled by setting UART_IRDA_EN. When UART_IRDA_TX_EN is set (high), the IrDA transceiver is enabled to send data and not allowed to receive data; when UART_IRDA_TX_EN is reset (low), the IrDA transceiver is enabled to receive data and not allowed to send data.

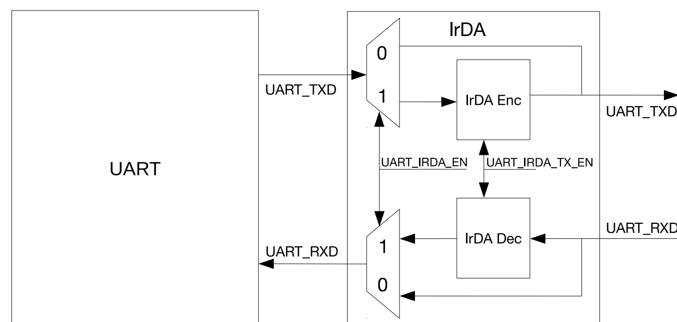


Figure 23-9. IrDA Encoding and Decoding Diagram

23.3.8 Wake-up

UART0 and UART1 can be set as wake-up source. When a UART controller is in Light-sleep mode, Wakeup_Ctrl counts up the rising edges of rxd_in. When the number of rising edges is greater than (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC, which then wakes up ESP32-S2.

23.3.9 Flow Control

UART controllers have two ways to control data flow, namely hardware flow control and software flow control. Hardware flow control is achieved using output signal rtsn_out and input signal dsrn_in. Software flow control is achieved by inserting special characters in data flow sent and detecting special characters in data flow received.

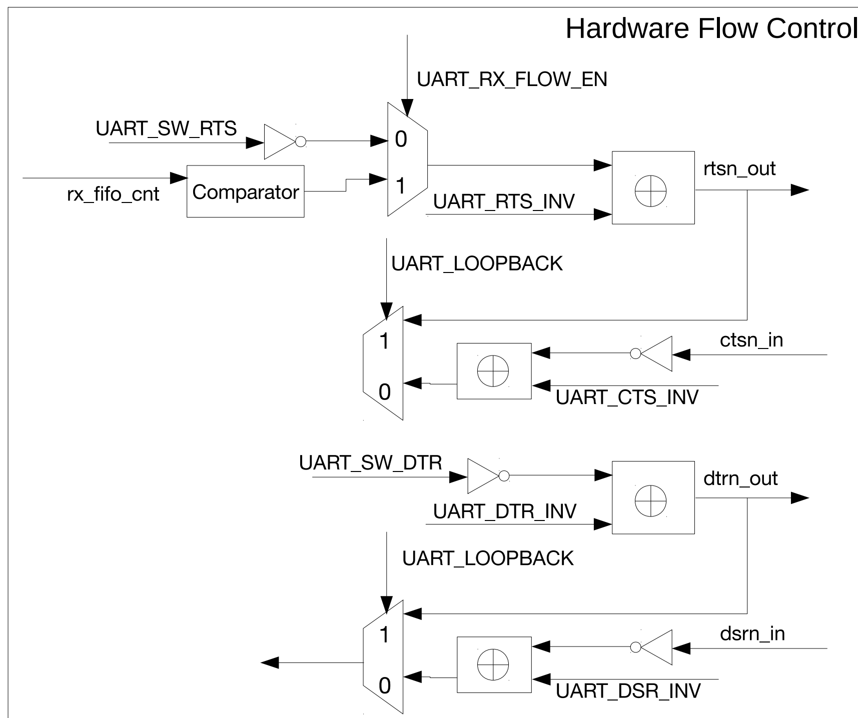


Figure 23-10. Hardware Flow Control Diagram

23.3.9.1 Hardware Flow Control

Figure 23-10 shows hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 23-11 illustrates how these signals are connected between ESP32-S2 UART (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

When rtsn_out of IU0 is low, EU0 is allowed to send data; when rtsn_out of IU0 is high, EU0 is notified to stop sending data until rtsn_out of IU0 returns to low. Output signal rtsn_out can be controlled in two ways.

- Software control: Enter this mode by setting UART_RX_FLOW_EN to 0. In this mode, the level of rtsn_out is changed by configuring UART_SW_RTS.
- Hardware control: Enter this mode by setting UART_RX_FLOW_EN to 1. In this mode, rtsn_out is pulled high when data in Rx_FIFO exceeds UART_RX_FLOW_THRHD.

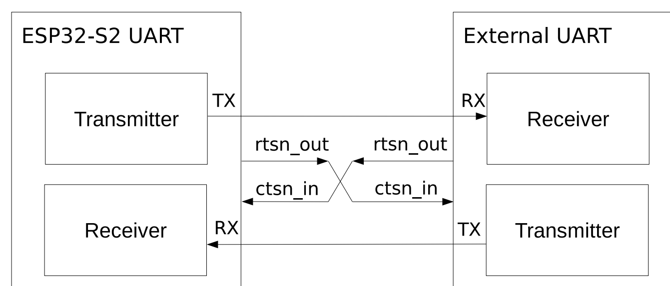


Figure 23-11. Connection between Hardware Flow Control Signals

When ctsn_in of IU0 is low, IU0 is allowed to send data; when ctsn_in is high, IU0 is not allowed to send data. When IU0 detects an edge change of ctsn_in, a UART_CTS_CHG_INT interrupt is generated.

If dtrn_out of IU0 is high, it indicates that IU0 is ready to transmit data. dtrn_out is generated by configuring register UART_SW_DTR. When the IU0 transmitter detects a edge change of dsrn_in, a UART_DSR_CHG_INT

interrupt is generated. After this interrupt is detected, software can obtain the level of input signal dsrn_in by reading UART_DSRN. If dsrn_in is high, it indicates that EU0 is ready to transmit data.

In a two-wire RS485 multidrop network enabled by setting UART_RS485_EN, dtrn_out is used for transmit/receive turnaround. In this case, dtrn_out is generated by hardware. When data transmission starts, dtrn_out is pulled high and the external driver is enabled; when data transmission completes, dtrn_out is pulled low and the external driver is disabled. Please note that when there is turnaround delay of one cycle added after the stop bit, dtrn_out is pulled low after the delay.

UART loopback test is enabled by setting UART_LOOPBACK. In the test, UART output signal txd_out is connected to its input signal rxd_in, rtsn_out is connected to ctsn_in, and dtrn_out is connected to dsrn_out. If data sent matches data received, it indicates that UART controllers are working properly.

23.3.9.2 Software Flow Control

Instead of CTS/RTS lines, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting UART_SW_FLOW_CON_EN to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in data flow received, and generate a UART_SW_XOFF_INT or a UART_SW_XON_INT interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting UART_FORCE_XOFF, or to start sending data by setting UART_FORCE_XON.

Software determines whether to insert flow control characters according to the remaining room in RX FIFO. When UART_SEND_XOFF is set, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission; when UART_SEND_XON is set, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores more data than UART_XOFF_THRESHOLD, UART_SEND_XOFF is set by hardware. As a result, the transmitter sends an XOFF character after the current byte in transmission. If the RX FIFO of a UART controller stores less data than UART_XON_THRESHOLD, UART_SEND_XON is set by hardware. As a result, the transmitter sends an XON character after the current byte in transmission.

23.3.10 UDMA

The two UART controllers on ESP32-S2 share one UDMA (UART DMA), which supports the decoding and encoding of HCI data packets. For more information, please refer to Chapter 2: *DMA Controller (DMA)*.

23.3.11 UART Interrupts

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.

- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter sends a NULL character after all data in TX FIFO has been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.
- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than UART_RX_TOUT_THRHD to receive one byte.
- UART_BRK_DET_INT: Triggered when the receiver detects a NULL character after stop bits.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of CTSn signals.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of DSRn signals.
- UART_RXFIFO_OVF_INT: Triggered when the receiver receives more data than the capacity of RX FIFO.
- UART_FRM_ERR_INT: Triggered when the receiver detects a data frame error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

23.3.12 UHCI Interrupts

- UHCI_DMA_INFIFO_FULL_WM_INT: Triggered when the counter value of DMA RX FIFO exceeds UHCI_DMA_INFIFO_FULL_THRS.
- UHCI_SEND_A_REG_Q_INT: Triggered when DMA has sent a series of short packets using always_send.
- UHCI_SEND_S_REG_Q_INT: Triggered when DMA has sent a series of short packets using single_send.
- UHCI_OUT_TOTAL_EOF_INT: Triggered when all data has been sent.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_IN_DSCR_EMPTY_INT: Triggered when there are not enough receive descriptors for DMA.
- UHCI_OUT_DSCR_ERR_INT: Triggered when an error is detected in a transmit descriptor.
- UHCI_IN_DSCR_ERR_INT: Triggered when an error is detected in an receive descriptor.
- UHCI_OUT_EOF_INT: Triggered when the EOF bit in a descriptor is 1.
- UHCI_OUT_DONE_INT: Triggered when a transmit descriptor is completed.
- UHCI_IN_ERR_EOF_INT: Triggered when an EOF error is detected in an receive descriptor.

- UHCI_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- UHCI_IN_DONE_INT: Triggered when an receive descriptor is completed.
- UHCI_TX_HUNG_INT: Triggered when DMA spends too much time on reading RAM.
- UHCI_RX_HUNG_INT: Triggered when DMA spends too much time on receiving data.
- UHCI_TX_START_INT: Triggered when DMA detects a separator character.
- UHCI_RX_START_INT: Triggered when a separator character has been sent.

23.4 Base Address

Users can access UART0, UART1 and UHCI0 respectively with two register base addresses shown in the following table. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 128: UART0, UART1 and UHCI0 Base Address

Module	Bus to Access Peripheral	Base Address
UART0	PeriBUS1	0x3F400000
	PeriBUS2	0x60000000
UART1	PeriBUS1	0x3F410000
	PeriBUS2	0x60010000
UHCI0	PeriBUS1	0x3F414000
	PeriBUS2	0x60014000

23.5 Register Summary

The addresses in the following table are relative to the UART base addresses provided in Section 23.4.

Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	R/W
UART_MEM_CONF_REG	UART threshold and allocation configuration	0x005C	R/W
Interrupt registers			
UART_INT_RAW_REG	Raw interrupt status	0x0004	RO
Interrupt Register			
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WO
Configuration Register			
UART_CLKDIV_REG	Clock divider configuration	0x0014	R/W
UART_CONF0_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_FLOW_CONF_REG	Software flow control configuration	0x0034	R/W
UART_SLEEP_CONF_REG	Sleeping mode configuration	0x0038	R/W
UART_SWFC_CONF0_REG	Software flow control character configuration	0x003C	R/W
UART_SWFC_CONF1_REG	Software flow-control character configuration	0x0040	R/W

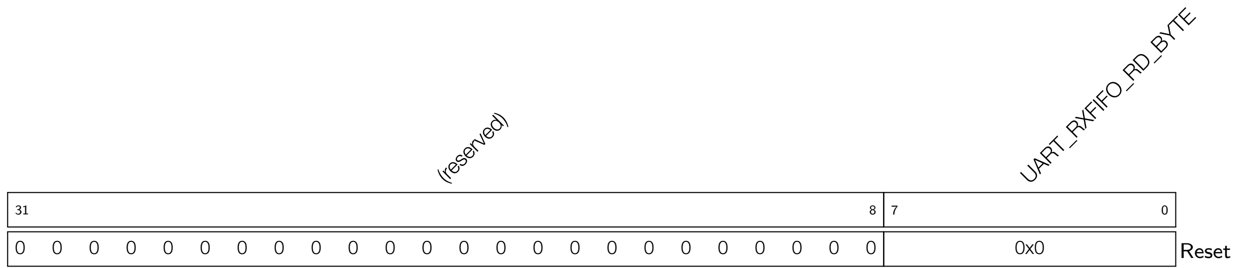
Name	Description	Address	Access
UART_IDLE_CONF_REG	Frame-end idle configuration	0x0044	R/W
UART_RS485_CONF_REG	RS485 mode configuration	0x0048	R/W
Autobaud Register			
UART_AUTOBAUD_REG	Autobaud configuration register	0x0018	R/W
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x0028	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x002C	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0030	RO
UART_POSPULSE_REG	Autobaud high pulse register	0x006C	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0070	RO
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0060	RO
UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x0064	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0068	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_REG	Pre-sequence timing configuration	0x004C	R/W
UART_AT_CMD_POSTCNT_REG	Post-sequence timing configuration	0x0050	R/W
UART_AT_CMD_GAPTOUT_REG	Timeout configuration	0x0054	R/W
UART_AT_CMD_CHAR_REG	AT Escape Sequence Selection Configuration	0x0058	R/W
Version Register			
UART_DATE_REG	UART version control register	0x0074	R/W

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x002C	R/W
UHCI_AHB_TEST_REG	AHB test register	0x0048	R/W
UHCI_ESCAPE_CONF_REG	Escape characters configuration	0x0064	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0068	R/W
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0074	R/W
UHCI_Q0_WORD0_REG	Q0_WORD0 quick_sent register	0x0078	R/W
UHCI_Q0_WORD1_REG	Q0_WORD1 quick_sent register	0x007C	R/W
UHCI_Q1_WORD0_REG	Q1_WORD0 quick_sent register	0x0080	R/W
UHCI_Q1_WORD1_REG	Q1_WORD1 quick_sent register	0x0084	R/W
UHCI_Q2_WORD0_REG	Q2_WORD0 quick_sent register	0x0088	R/W
UHCI_Q2_WORD1_REG	Q2_WORD1 quick_sent register	0x008C	R/W
UHCI_Q3_WORD0_REG	Q3_WORD0 quick_sent register	0x0090	R/W
UHCI_Q3_WORD1_REG	Q3_WORD1 quick_sent register	0x0094	R/W
UHCI_Q4_WORD0_REG	Q4_WORD0 quick_sent register	0x0098	R/W
UHCI_Q4_WORD1_REG	Q4_WORD1 quick_sent register	0x009C	R/W
UHCI_Q5_WORD0_REG	Q5_WORD0 quick_sent register	0x00A0	R/W
UHCI_Q5_WORD1_REG	Q5_WORD1 quick_sent register	0x00A4	R/W
UHCI_Q6_WORD0_REG	Q6_WORD0 quick_sent register	0x00A8	R/W

Name	Description	Address	Access
UHCI_Q6_WORD1_REG	Q6_WORD1 quick_sent register	0x00AC	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x00B0	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x00B4	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x00B8	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x00BC	R/W
UHCI_PKT_THRES_REG	Configure register for packet length	0x00C0	R/W
Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	RO
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WO
DMA Status			
UHCI_DMA_OUT_STATUS_REG	DMA data-output status register	0x0014	RO
UHCI_DMA_IN_STATUS_REG	UHCI data-input status register	0x001C	RO
UHCI_STATE0_REG	UHCI decoder status register	0x0030	RO
UHCI_STATE1_REG	UHCI encoder status register	0x0034	RO
UHCI_DMA_OUT_EOF_DES_ADDR_REG	Outlink descriptor address when EOF occurs	0x0038	RO
UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG	Inlink descriptor address when EOF occurs	0x003C	RO
UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG	Inlink descriptor address when errors occur	0x0040	RO
UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG	Outlink descriptor address before the last transmit descriptor	0x0044	RO
UHCI_DMA_IN_DSCR_REG	The third word of the next receive descriptor	0x004C	RO
UHCI_DMA_IN_DSCR_BF0_REG	The third word of current receive descriptor	0x0050	RO
UHCI_DMA_OUT_DSCR_REG	The third word of the next transmit descriptor	0x0058	RO
UHCI_DMA_OUT_DSCR_BF0_REG	The third word of current transmit descriptor	0x005C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x0070	RO
DMA Configuration			
UHCI_DMA_OUT_PUSH_REG	Push control register of data-output FIFO	0x0018	R/W
UHCI_DMA_IN_POP_REG	Pop control register of data-input FIFO	0x0020	varies
UHCI_DMA_OUT_LINK_REG	Link descriptor address and control	0x0024	varies
UHCI_DMA_IN_LINK_REG	Link descriptor address and control	0x0028	varies
Version Register			
UHCI_DATE_REG	UHCI version control register	0x00FC	R/W

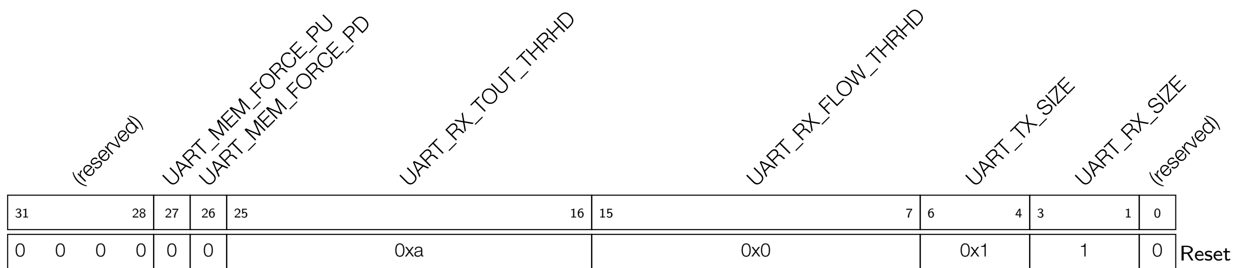
23.6 Registers

Register 23.1: UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE UART *n* accesses FIFO via this register. (R/W)

Register 23.2: UART_MEM_CONF_REG (0x005C)



UART_RX_SIZE This register is used to configure the amount of mem allocated for RX FIFO. The default number is 128 bytes. (R/W)

UART_TX_SIZE This register is used to configure the amount of mem allocated for TX FIFO. The default number is 128 bytes. (R/W)

UART_RX_FLOW_THRHD This register is used to configure the maximum amount of data that can be received when hardware flow control works. (R/W)

UART_RX_TOUT_THRHD This register is used to configure the threshold time that receiver takes to receive one byte. The UART_RXFIFO_TOUT_INT interrupt will be triggered when the receiver takes more time to receive one byte with UART RX_TOUT_EN set to 1. (R/W)

UART_MEM_FORCE_PD Set this bit to force power down UART memory. (R/W)

UART_MEM_FORCE_PU Set this bit to force power up UART memory. (R/W)

Register 23.3: UART_INT_RAW_REG (0x0004)

(reserved)	UART_WAKEUP_INT_RAW	UART_AT_CMD_CHAR_DET_INT_RAW	UART_RS485_CLASH_INT_RAW	UART_RS485_FRM_ERR_INT_RAW	UART_RS485_PARITY_ERR_INT_RAW	UART_TX_DONE_INT_RAW	UART_TX_BRK_IDLE_DONE_INT_RAW	UART_TX_BRK_DONE_INT_RAW	UART_GLITCH_DET_INT_RAW	UART_SW_XOFF_INT_RAW	UART_SW_XON_INT_RAW	UART_RXFIFO_TOUT_INT_RAW	UART_BRK_DET_INT_RAW	UART_CTS_CHG_INT_RAW	UART_DSR_CHG_INT_RAW	UART_RXFIFO_OVF_INT_RAW	UART_FRM_ERR_INT_RAW	UART_PARITY_ERR_INT_RAW	UART_TXFIFO_EMPTY_INT_RAW	UART_RXFIFO_FULL_INT_RAW	
31 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UART_RXFIFO_FULL_INT_RAW This interrupt raw bit turns to high level when receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies. (RO)

UART_TXFIFO_EMPTY_INT_RAW This interrupt raw bit turns to high level when the amount of data in TX FIFO is less than what UART_TXFIFO_EMPTY_THRHD specifies. (RO)

UART_PARITY_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a parity error in the data. (RO)

UART_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a data frame error. (RO)

UART_RXFIFO_OVF_INT_RAW This interrupt raw bit turns to high level when receiver receives more data than the FIFO can store. (RO)

UART_DSR_CHG_INT_RAW This interrupt raw bit turns to high level when receiver detects the edge change of DSRn signal. (RO)

UART_CTS_CHG_INT_RAW This interrupt raw bit turns to high level when receiver detects the edge change of CTSn signal. (RO)

UART_BRK_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects a 0 after the stop bit. (RO)

UART_RXFIFO_TOUT_INT_RAW This interrupt raw bit turns to high level when receiver takes more time than UART_RX_TOUT_THRHD to receive a byte. (RO)

UART_SW_XON_INT_RAW This interrupt raw bit turns to high level when receiver receives XON character when UART_SW_FLOW_CON_EN is set to 1. (RO)

UART_SW_XOFF_INT_RAW This interrupt raw bit turns to high level when receiver receives XOFF character when UART_SW_FLOW_CON_EN is set to 1. (RO)

UART_GLITCH_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects a glitch in the middle of a start bit. (RO)

UART_TX_BRK_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter completes sending NULL characters, after all data in TX FIFO are sent. (RO)

UART_TX_BRK_IDLE_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter has kept the shortest duration after sending the last data. (RO)

Continued on the next page...

Register 23.3: UART_INT_RAW_REG (0x0004)

Continued from the previous page...

UART_TX_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter has sent out all data in FIFO. (RO)

UART_RS485_PARITY_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a parity error from the echo of transmitter in RS485 mode. (RO)

UART_RS485_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a data frame error from the echo of transmitter in RS485 mode. (RO)

UART_RS485_CLASH_INT_RAW This interrupt raw bit turns to high level when detects a clash between transmitter and receiver in RS485 mode. (RO)

UART_AT_CMD_CHAR_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects the configured UART_AT_CMD CHAR. (RO)

UART_WAKEUP_INT_RAW This interrupt raw bit turns to high level when input rxd edge changes more times than what UART_ACTIVE_THRESHOLD specifies in Light-sleep mode. (RO)

Register 23.4: UART_INT_ST_REG (0x0008)

Bits	31–20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field	(reserved)	UART_WAKEUP_INT_ST	UART_AT_CMD_CHAR_DET_INT_ST	UART_RS485_CLASH_INT_ST	UART_RS485_FRM_ERR_INT_ST	UART_RS485_PARITY_ERR_INT_ST	UART_TX_DONE_INT_ST	UART_TX_BRK_IDLE_DONE_INT_ST	UART_TX_BRK_DONE_INT_ST	UART_GLITCH_DET_INT_ST	UART_SW_XOFF_INT_ST	UART_SW_XON_INT_ST	UART_RXFIFO_TOUT_INT_ST	UART_BRK_DET_INT_ST	UART_CTS_CHG_INT_ST	UART_DSR_CHG_INT_ST	UART_RXFIFO_OVF_INT_ST	UART_FRM_ERR_INT_ST	UART_PARITY_ERR_INT_ST	UART_TXFIFO_EMPTY_INT_ST	UART_RXFIFO_FULL_INT_ST	
Value	0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UART_RXFIFO_FULL_INT_ST This is the status bit for UART_RXFIFO_FULL_INT_RAW when UART_RXFIFO_FULL_INT_ENA is set to 1. (RO)

UART_TXFIFO_EMPTY_INT_ST This is the status bit for UART_TXFIFO_EMPTY_INT_RAW when UART_TXFIFO_EMPTY_INT_ENA is set to 1. (RO)

UART_PARITY_ERR_INT_ST This is the status bit for UART_PARITY_ERR_INT_RAW when UART_PARITY_ERR_INT_ENA is set to 1. (RO)

UART_FRM_ERR_INT_ST This is the status bit for UART_FRM_ERR_INT_RAW when UART_FRM_ERR_INT_ENA is set to 1. (RO)

UART_RXFIFO_OVF_INT_ST This is the status bit for UART_RXFIFO_OVF_INT_RAW when UART_RXFIFO_OVF_INT_ENA is set to 1. (RO)

UART_DSR_CHG_INT_ST This is the status bit for UART_DSR_CHG_INT_RAW when UART_DSR_CHG_INT_ENA is set to 1. (RO)

UART_CTS_CHG_INT_ST This is the status bit for UART_CTS_CHG_INT_RAW when UART_CTS_CHG_INT_ENA is set to 1. (RO)

UART_BRK_DET_INT_ST This is the status bit for UART_BRK_DET_INT_RAW when UART_BRK_DET_INT_ENA is set to 1. (RO)

UART_RXFIFO_TOUT_INT_ST This is the status bit for UART_RXFIFO_TOUT_INT_RAW when UART_RXFIFO_TOUT_INT_ENA is set to 1. (RO)

UART_SW_XON_INT_ST This is the status bit for UART_SW_XON_INT_RAW when UART_SW_XON_INT_ENA is set to 1. (RO)

UART_SW_XOFF_INT_ST This is the status bit for UART_SW_XOFF_INT_RAW when UART_SW_XOFF_INT_ENA is set to 1. (RO)

UART_GLITCH_DET_INT_ST This is the status bit for UART_GLITCH_DET_INT_RAW when UART_GLITCH_DET_INT_ENA is set to 1. (RO)

UART_TX_BRK_DONE_INT_ST This is the status bit for UART_TX_BRK_DONE_INT_RAW when UART_TX_BRK_DONE_INT_ENA is set to 1. (RO)

UART_TX_BRK_IDLE_DONE_INT_ST This is the status bit for UART_TX_BRK_IDLE_DONE_INT_RAW when UART_TX_BRK_IDLE_DONE_INT_ENA is set to 1. (RO)

Continued on the next page...

Register 23.4: UART_INT_ST_REG (0x0008)

Continued from the previous page...

UART_TX_DONE_INT_ST This is the status bit for UART_TX_DONE_INT_RAW when UART_TX_DONE_INT_ENA is set to 1. (RO)

UART_RS485_PARITY_ERR_INT_ST This is the status bit for UART_RS485_PARITY_ERR_INT_RAW when UART_RS485_PARITY_INT_ENA is set to 1. (RO)

UART_RS485_FRM_ERR_INT_ST This is the status bit for UART_RS485_FRM_ERR_INT_RAW when UART_RS485_FM_ERR_INT_ENA is set to 1. (RO)

UART_RS485_CLASH_INT_ST This is the status bit for UART_RS485_CLASH_INT_RAW when UART_RS485_CLASH_INT_ENA is set to 1. (RO)

UART_AT_CMD_CHAR_DET_INT_ST This is the status bit for UART_AT_CMD_DET_INT_RAW when UART_AT_CMD_CHAR_DET_INT_ENA is set to 1. (RO)

UART_WAKEUP_INT_ST This is the status bit for UART_WAKEUP_INT_RAW when UART_WAKEUP_INT_ENA is set to 1. (RO)

Register 23.5: UART_INT_ENA_REG (0x000C)

Bit(s)	Field	Reset
31:20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	UART_WAKEUP_INT_ENA	0
18	UART_AT_CMD_CHAR_DET_INT_ENA	0
17	UART_RS485_CLASH_INT_ENA	0
16	UART_RS485_FRM_ERR_INT_ENA	0
15	UART_RS485_PARITY_ERR_INT_ENA	0
14	UART_TX_DONE_INT_ENA	0
13	UART_TX_BRK_IDLE_DONE_INT_ENA	0
12	UART_TX_BRK_DONE_INT_ENA	0
11	UART_GLITCH_DET_INT_ENA	0
10	UART_SW_XOFF_INT_ENA	0
9	UART_SW_XON_INT_ENA	0
8	UART_RXFIFO_TOUT_INT_ENA	0
7	UART_BRK_DET_INT_ENA	0
6	UART_CTS_CHG_INT_ENA	0
5	UART_DSR_CHG_INT_ENA	0
4	UART_RXFIFO_OVF_INT_ENA	0
3	UART_FRM_ERR_INT_ENA	0
2	UART_PARITY_ERR_INT_ENA	0
1	UART_TXFIFO_EMPTY_INT_ENA	0
0	UART_RXFIFO_FULL_INT_ENA	0

UART_RXFIFO_FULL_INT_ENA This is the enable bit for UART_RXFIFO_FULL_INT_ST register. (R/W)

UART_TXFIFO_EMPTY_INT_ENA This is the enable bit for UART_TXFIFO_EMPTY_INT_ST register. (R/W)

UART_PARITY_ERR_INT_ENA This is the enable bit for UART_PARITY_ERR_INT_ST register. (R/W)

UART_FRM_ERR_INT_ENA This is the enable bit for UART_FRM_ERR_INT_ST register. (R/W)

UART_RXFIFO_OVF_INT_ENA This is the enable bit for UART_RXFIFO_OVF_INT_ST register. (R/W)

UART_DSR_CHG_INT_ENA This is the enable bit for UART_DSR_CHG_INT_ST register. (R/W)

UART_CTS_CHG_INT_ENA This is the enable bit for UART_CTS_CHG_INT_ST register. (R/W)

UART_BRK_DET_INT_ENA This is the enable bit for UART_BRK_DET_INT_ST register. (R/W)

UART_RXFIFO_TOUT_INT_ENA This is the enable bit for UART_RXFIFO_TOUT_INT_ST register. (R/W)

UART_SW_XON_INT_ENA This is the enable bit for UART_SW_XON_INT_ST register. (R/W)

UART_SW_XOFF_INT_ENA This is the enable bit for UART_SW_XOFF_INT_ST register. (R/W)

UART_GLITCH_DET_INT_ENA This is the enable bit for UART_GLITCH_DET_INT_ST register. (R/W)

UART_TX_BRK_DONE_INT_ENA This is the enable bit for UART_TX_BRK_DONE_INT_ST register. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA This is the enable bit for UART_TX_BRK_IDLE_DONE_INT_ST register. (R/W)

UART_TX_DONE_INT_ENA This is the enable bit for UART_TX_DONE_INT_ST register. (R/W)

UART_RS485_PARITY_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

UART_RS485_FRM_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

Continued on the next page...

Register 23.5: UART_INT_ENA_REG (0x000C)

Continued from the previous page...

UART_RS485_CLASH_INT_ENA This is the enable bit for UART_RS485_CLASH_INT_ST register. (R/W)

UART_AT_CMD_CHAR_DET_INT_ENA This is the enable bit for UART_AT_CMD_CHAR_DET_INT_ST register. (R/W)

UART_WAKEUP_INT_ENA This is the enable bit for UART_WAKEUP_INT_ST register. (R/W)

Register 23.6: UART_INT_CLR_REG (0x0010)

Bits	Field	Reset
31–20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	UART_WAKEUP_INT_CLR	0
18	UART_AT_CMD_CHAR_DET_INT_CLR	0
17	UART_RS485_CLASH_INT_CLR	0
16	UART_RS485_FRM_ERR_INT_CLR	0
15	UART_RS485_PARITY_ERR_INT_CLR	0
14	UART_TX_DONE_INT_CLR	0
13	UART_TX_BRK_IDLE_DONE_INT_CLR	0
12	UART_TX_BRK_DONE_INT_CLR	0
11	UART_GLITCH_DET_INT_CLR	0
10	UART_SW_XOFF_INT_CLR	0
9	UART_SW_XON_INT_CLR	0
8	UART_RXFIFO_TOUT_INT_CLR	0
7	UART_BRK_DET_INT_CLR	0
6	UART_CTS_CHG_INT_CLR	0
5	UART_DSR_CHG_INT_CLR	0
4	UART_RXFIFO_OVF_INT_CLR	0
3	UART_FRM_ERR_INT_CLR	0
2	UART_PARITY_ERR_INT_CLR	0
1	UART_TXFIFO_EMPTY_INT_CLR	0
0	UART_RXFIFO_FULL_INT_CLR	0

UART_RXFIFO_FULL_INT_CLR Set this bit to clear UART_THE RXFIFO_FULL_INT_RAW interrupt. (WO)

UART_TXFIFO_EMPTY_INT_CLR Set this bit to clear UART_TXFIFO_EMPTY_INT_RAW interrupt. (WO)

UART_PARITY_ERR_INT_CLR Set this bit to clear UART_PARITY_ERR_INT_RAW interrupt. (WO)

UART_FRM_ERR_INT_CLR Set this bit to clear UART_FRM_ERR_INT_RAW interrupt. (WO)

UART_RXFIFO_OVF_INT_CLR Set this bit to clear UART_UART_RXFIFO_OVF_INT_RAW interrupt. (WO)

UART_DSR_CHG_INT_CLR Set this bit to clear UART_DSR_CHG_INT_RAW interrupt. (WO)

UART_CTS_CHG_INT_CLR Set this bit to clear UART_CTS_CHG_INT_RAW interrupt. (WO)

UART_BRK_DET_INT_CLR Set this bit to clear UART_BRK_DET_INT_RAW interrupt. (WO)

UART_RXFIFO_TOUT_INT_CLR Set this bit to clear UART_RXFIFO_TOUT_INT_RAW interrupt. (WO)

UART_SW_XON_INT_CLR Set this bit to clear UART_SW_XON_INT_RAW interrupt. (WO)

UART_SW_XOFF_INT_CLR Set this bit to clear UART_SW_XOFF_INT_RAW interrupt. (WO)

UART_GLITCH_DET_INT_CLR Set this bit to clear UART_GLITCH_DET_INT_RAW interrupt. (WO)

UART_TX_BRK_DONE_INT_CLR Set this bit to clear UART_TX_BRK_DONE_INT_RAW interrupt. (WO)

UART_TX_BRK_IDLE_DONE_INT_CLR Set this bit to clear UART_TX_BRK_IDLE_DONE_INT_RAW interrupt. (WO)

UART_TX_DONE_INT_CLR Set this bit to clear UART_TX_DONE_INT_RAW interrupt. (WO)

UART_RS485_PARITY_ERR_INT_CLR Set this bit to clear UART_RS485_PARITY_ERR_INT_RAW interrupt. (WO)

UART_RS485_FRM_ERR_INT_CLR Set this bit to clear UART_RS485_FRM_ERR_INT_RAW interrupt. (WO)

Continued on the next page...

Register 23.6: UART_INT_CLR_REG (0x0010)

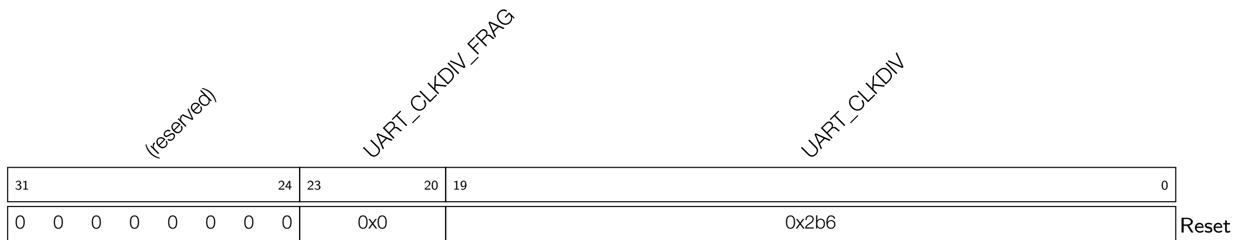
Continued from the previous page...

UART_RS485_CLASH_INT_CLR Set this bit to clear UART_RS485_CLASH_INT_RAW interrupt. (WO)

UART_AT_CMD_CHAR_DET_INT_CLR Set this bit to clear UART_AT_CMD_CHAR_DET_INT_RAW interrupt. (WO)

UART_WAKEUP_INT_CLR Set this bit to clear UART_WAKEUP_INT_RAW interrupt. (WO)

Register 23.7: UART_CLKDIV_REG (0x0014)



UART_CLKDIV The integral part of the frequency divisor. (R/W)

UART_CLKDIV_FRAG The decimal part of the frequency divisor. (R/W)

Register 23.8: UART_CONF0_REG (0x0020)

Bits	Field	Reset
31:29	(reserved)	0 0 0
28	UART_MEM_CLK_EN	1
27	UART_TICK_REF_ALWAYS_ON	1
26:25	(reserved)	0 0
24	UART_DTR_INV	0
23	UART_RTS_INV	0
22	UART_TXD_INV	0
21	UART_DSR_INV	0
20	UART_CTS_INV	0
19	UART_RXD_INV	0
18	UART_TXFIFO_RST	0
17	UART_RXFIFO_RST	0
16	UART_IRDA_EN	0
15	UART_TX_FLOW_EN	0
14	UART_LOOPBACK	0
13	UART_IRDA_RX_INV	0
12	UART_IRDA_TX_INV	0
11	UART_IRDA_WCTL	0
10	UART_IRDA_TX_EN	0
9	UART_IRDA_DPLX	0
8	UART_TXD_BRK	0
7	UART_SW_DTR	0
6	UART_SW_RTS	0
5:4	UART_STOP_BIT_NUM	1
3:2	UART_BIT_NUM	3
1	UART_PARITY_EN	0
0	UART_PARITY	0

UART_PARITY This register is used to configure the parity check mode. 1'h0: even. 1'h1: odd. (R/W)

UART_PARITY_EN Set this bit to enable UART parity check. (R/W)

UART_BIT_NUM This register is used to set the length of data. 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits. (R/W)

UART_STOP_BIT_NUM This register is used to set the length of stop bit. 1: 1 bit 2: 1.5 bits 3: 2 bits. (R/W)

UART_SW_RTS This register is used to configure the software RTS signal which is used in software flow control. (R/W)

UART_SW_DTR This register is used to configure the software DTR signal which is used in software flow control. (R/W)

UART_TXD_BRK Set this bit to enable transmitter to send NULL when the process of sending data is done. (R/W)

UART_IRDA_DPLX Set this bit to enable IrDA loopback mode. (R/W)

UART_IRDA_TX_EN This is the start enable bit for IrDA transmitter. (R/W)

UART_IRDA_WCTL 1'h1: The IrDA transmitter's 11th bit is the same as 10th bit. 1'h0: Set IrDA transmitter's 11th bit to 0. (R/W)

UART_IRDA_TX_INV Set this bit to invert the level of IrDA transmitter. (R/W)

UART_IRDA_RX_INV Set this bit to invert the level of IrDA receiver. (R/W)

UART_LOOPBACK Set this bit to enable UART loopback test mode. (R/W)

UART_TX_FLOW_EN Set this bit to enable flow control function for transmitter. (R/W)

UART_IRDA_EN Set this bit to enable IrDA protocol. (R/W)

UART_RXFIFO_RST Set this bit to reset the UART RX FIFO. (R/W)

UART_TXFIFO_RST Set this bit to reset the UART TX FIFO. (R/W)

UART_RXD_INV Set this bit to inverse the level value of UART RXD signal. (R/W)

UART_CTS_INV Set this bit to inverse the level value of UART CTS signal. (R/W)

Continued on the next page...

Register 23.8: UART_CONF0_REG (0x0020)

Continued from the previous page...

UART_DSR_INV Set this bit to inverse the level value of UART DSR signal. (R/W)

UART_TXD_INV Set this bit to inverse the level value of UART TXD signal. (R/W)

UART_RTS_INV Set this bit to inverse the level value of UART RTS signal. (R/W)

UART_DTR_INV Set this bit to inverse the level value of UART DTR signal. (R/W)

UART_TICK_REF_ALWAYS_ON This register is used to select the clock. 1'h1: APB_CLK. 1'h0: REF_TICK. (R/W)

UART_MEM_CLK_EN UART memory clock gate enable signal. 1: UART memory powers on, the data of which can be read and written. 0: UART memory powers down. (R/W)

Register 23.9: UART_CONF1_REG (0x0024)

UART_RX_TOUT_EN	UART_RX_FLOW_EN	UART_RX_TOUT_FLOW_DIS	(reserved)	UART_TXFIFO_EMPTY_THRHD	UART_RXFIFO_FULL_THRHD	
31	30	29	28 ... 18	17 ... 9	8 ... 0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0	0x60	0x60	Reset

UART_RXFIFO_FULL_THRHD It will produce UART_RXFIFO_FULL_INT interrupt when receiver receives more data than this register value. (R/W)

UART_TXFIFO_EMPTY_THRHD It will produce UART_TXFIFO_EMPTY_INT interrupt when the data amount in TX FIFO is less than this register value. (R/W)

UART_RX_TOUT_FLOW_DIS Set this bit to stop accumulating idle_cnt when hardware flow control works. (R/W)

UART_RX_FLOW_EN This is the flow enable bit for UART receiver. 1'h1: Choose software flow control with configuring sw_rts signal. 1'h0: Disable software flow control. (R/W)

UART_RX_TOUT_EN This is the enable bit for UART receiver's timeout function. (R/W)

Register 23.10: UART_FLOW_CONF_REG (0x0034)

31 ... 6 (reserved)	5 UART_SEND_XOFF	4 UART_SEND_XON	3 UART_FORCE_XOFF	2 UART_FORCE_XON	1 UART_XONOFF_DEL	0 UART_SW_FLOW_CON_EN	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

UART_SW_FLOW_CON_EN Set this bit to enable software flow control. It is used with register SW_XON or SW_XOFF. (R/W)

UART_XONOFF_DEL Set this bit to remove flow control character from the received data. (R/W)

UART_FORCE_XON Set this bit to enable the transmitter to go on sending data. (R/W)

UART_FORCE_XOFF Set this bit to stop the transmitter from sending data. (R/W)

UART_SEND_XON Set this bit to send XON character. It is cleared by hardware automatically. (R/W)

UART_SEND_XOFF Set this bit to send XOFF character. It is cleared by hardware automatically. (R/W)

Register 23.11: UART_SLEEP_CONF_REG (0x0038)

31 ... 10 (reserved)	9 ... 0 UART_ACTIVE_THRESHOLD	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0xf0	Reset

UART_ACTIVE_THRESHOLD The UART is activated from light sleeping mode when the input rxd edge changes more times than this register value. (R/W)

Register 23.12: UART_SWFC_CONF0_REG (0x003C)

(reserved) 31–17	UART_XOFF_CHAR 16–9	UART_XOFF_THRESHOLD 8–0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x13	0xe0	Reset

UART_XOFF_THRESHOLD When the data amount in RX FIFO is more than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XOFF character. (R/W)

UART_XOFF_CHAR This register stores the XOFF flow control character. (R/W)

Register 23.13: UART_SWFC_CONF1_REG (0x0040)

(reserved) 31–17	UART_XON_CHAR 16–9	UART_XON_THRESHOLD 8–0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x11	0x0	Reset

UART_XON_THRESHOLD When the data amount in RX FIFO is less than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XON character. (R/W)

UART_XON_CHAR This register stores the XON flow control character. (R/W)

Register 23.14: UART_IDLE_CONF_REG (0x0044)

(reserved) 31–28	UART_TX_BRK_NUM 27–20	UART_TX_IDLE_NUM 19–10	UART_RX_IDLE_THRHD 9–0	
0 0 0 0	0xa	0x100	0x100	Reset

UART_RX_IDLE_THRHD It will produce frame end signal when receiver takes more time to receive one byte data than this register value. (R/W)

UART_TX_IDLE_NUM This register is used to configure the duration time between transfers. (R/W)

UART_TX_BRK_NUM This register is used to configure the number of 0 to be sent after the process of sending data is done. It is active when UART_TXD_BRK is set to 1. (R/W)

Register 23.15: UART_RS485_CONF_REG (0x0048)

(reserved)	UART_RS485_TX_DLY_NUM	UART_RS485_RX_DLY_NUM	UART_RS485RXBY_TX_EN	UART_RS485TX_RX_EN	UART_DL1_EN	UART_DL0_EN	UART_RS485_EN	
31 ... 10	9 ... 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0	0	0	0	0	0	Reset

UART_RS485_EN Set this bit to choose the RS485 mode. (R/W)

UART_DL0_EN Set this bit to delay the stop bit by 1 bit. (R/W)

UART_DL1_EN Set this bit to delay the stop bit by 1 bit. (R/W)

UART_RS485TX_RX_EN Set this bit to enable receiver could receive data when the transmitter is transmitting data in RS485 mode. (R/W)

UART_RS485RXBY_TX_EN 1'h1: enable RS485 transmitter to send data when RS485 receiver line is busy. 1'h0: RS485 transmitter should not send data when its receiver is busy. (R/W)

UART_RS485_RX_DLY_NUM This register is used to delay the receiver's internal data signal. (R/W)

UART_RS485_TX_DLY_NUM This register is used to delay the transmitter's internal data signal. (R/W)

Register 23.16: UART_AUTOBAUD_REG (0x0018)

(reserved)	UART_GLITCH_FILT	(reserved)	UART_AUTOBAUD_EN	
31 ... 16	15 ... 8	7 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x10	0 0 0 0 0 0 0	0	Reset

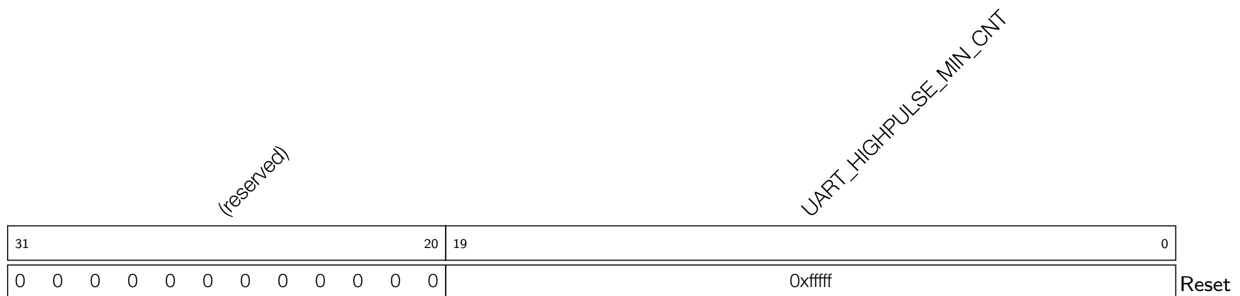
UART_AUTOBAUD_EN This is the enable bit for detecting baudrate. (R/W)

UART_GLITCH_FILT when input pulse width is lower than this value, the pulse is ignored. This register is used in autobaud detect process. (R/W)

Register 23.17: UART_LOWPULSE_REG (0x0028)

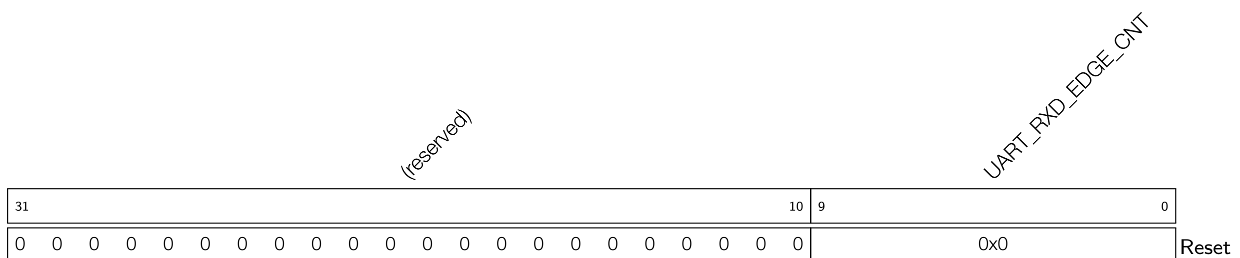
UART_LOWPULSE_MIN_CNT This register stores the value of the minimum duration time of the low level pulse. It is used in baud rate detection. (RO)

Register 23.18: UART_HIGHPULSE_REG (0x002C)



UART_HIGHPULSE_MIN_CNT This register stores the value of the maximum duration time for the high level pulse. It is used in baud rate detection. (RO)

Register 23.19: UART_RXD_CNT_REG (0x0030)



UART_RXD_EDGE_CNT This register stores the count of RXD edge change. It is used in baud rate detection. As baud rate registers UART_REG_LOWPULSE_MIN_CNT, UART_REG_HIGHPULSE_MIN_CNT, UART_REG_POSEDGE_MIN_CNT, and UART_REG_NEGEDGE_MIN_CNT always record the minimal value, UART_REG_RXD_EDGE_CNT indicates the statistic number of rxd edge to find out the minimal value for these baud rate registers. (RO)

Register 23.20: UART_POSPULSE_REG (0x006C)

UART_POSEDGE_MIN_CNT This register stores the minimal input clock count between two positive edges. It is used in baud rate detection. (RO)

Register 23.21: UART_NEGPULSE_REG (0x0070)

UART_NEGEDGE_MIN_CNT This register stores the minimal input clock count between two negative edges. It is used in baud rate detection. (RO)

Register 23.22: UART_STATUS_REG (0x001C)

UART_TXD	UART_RTSN	UART_DTRN	(reserved)	UART_TXFIFO_CNT	UART_RXD	UART_CTSN	UART_DSRN	(reserved)	UART_RXFIFO_CNT	
31	30	29	28 26	25 16	15	14	13	12 10	9 0	
0x0	0	0	0 0 0	0x0	0	0	0	0 0 0	0x0	Reset

UART_RXFIFO_CNT Stores the byte number of valid data in RX FIFO. (RO)

UART_DSRN The register represent the level value of the internal UART DSR signal. (RO)

UART_CTSN This register represent the level value of the internal UART CTS signal. (RO)

UART_RXD This register represent the level value of the internal UART RXD signal. (RO)

UART_TXFIFO_CNT Stores the byte number of data in TX FIFO. (RO)

UART_DTRN This bit represents the level of the internal UART DTR signal. (RO)

UART_RTSN This bit represents the level of the internal UART RTS signal. (RO)

UART_TXD This bit represents the level of the internal UART TXD signal. (RO)

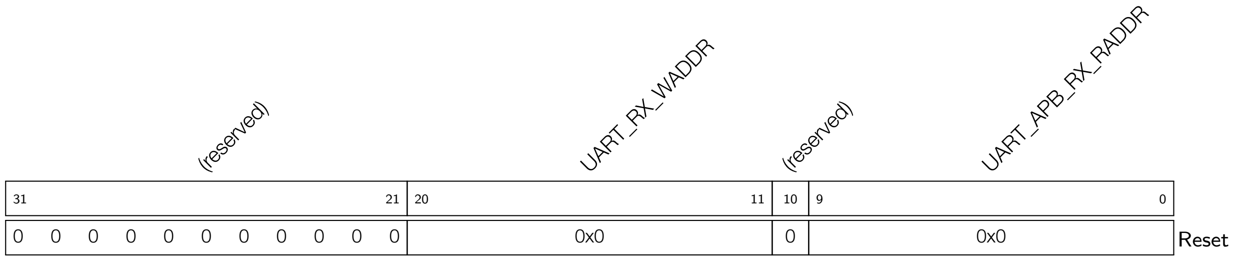
Register 23.23: UART_MEM_TX_STATUS_REG (0x0060)

(reserved)	UART_TX_RADDR	(reserved)	UART_APB_TX_WADDR	
31 21	20 11	10	9 0	
0 0 0 0 0 0 0 0 0 0 0	0x0	0	0x0	Reset

UART_APB_TX_WADDR This register stores the offset address in TX FIFO when software writes TX FIFO via APB. (RO)

UART_TX_RADDR This register stores the offset address in TX FIFO when TX FSM reads data via Tx_FIFO_Ctrl. (RO)

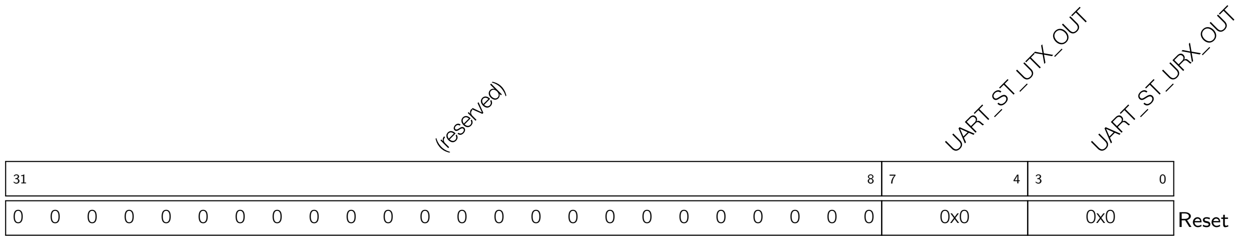
Register 23.24: UART_MEM_RX_STATUS_REG (0x0064)



UART_APB_RX_RADDR This register stores the offset address in RX_FIFO when software reads data from RX FIFO via APB. (RO)

UART_RX_WADDR This register stores the offset address in RX FIFO when Rx_FIFO_Ctrl writes RX FIFO. (RO)

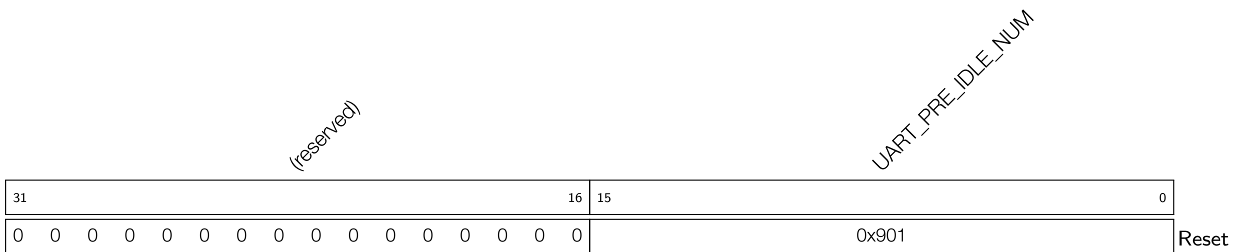
Register 23.25: UART_FSM_STATUS_REG (0x0068)



UART_ST_URX_OUT This is the status register of receiver. (RO)

UART_ST_UTX_OUT This is the status register of transmitter. (RO)

Register 23.26: UART_AT_CMD_PRECNT_REG (0x004C)



UART_PRE_IDLE_NUM This register is used to configure the idle duration time before the first AT_CMD is received by receiver. It will not take the next data received as AT_CMD character when the duration is less than this register value. (R/W)

Register 23.27: UART_AT_CMD_POSTCNT_REG (0x0050)

(reserved)		UART_POST_IDLE_NUM	
31	16	15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0x901	Reset

UART_POST_IDLE_NUM This register is used to configure the duration time between the last AT_CMD and the next data. It will not take the previous data as AT_CMD character when the duration is less than this register value. (R/W)

Register 23.28: UART_AT_CMD_GAPTOUT_REG (0x0054)

(reserved)		UART_RX_GAP_TOUT	
31	16	15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		11	Reset

UART_RX_GAP_TOUT This register is used to configure the duration time between the AT_CMD chars. It will not take the data as continuous AT_CMD chars when the duration time is less than this register value. (R/W)

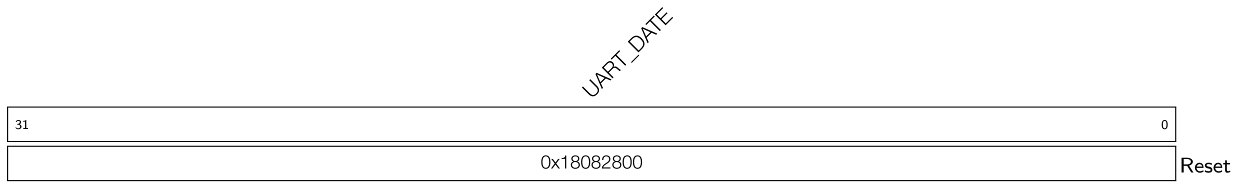
Register 23.29: UART_AT_CMD_CHAR_REG (0x0058)

(reserved)		UART_CHAR_NUM		UART_AT_CMD_CHAR	
31	16	15	8	7	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0x3		0x2b	Reset

UART_AT_CMD_CHAR This register is used to configure the content of AT_CMD character. (R/W)

UART_CHAR_NUM This register is used to configure the number of continuous AT_CMD chars received by receiver. (R/W)

Register 23.30: UART_DATE_REG (0x0074)



UART_DATE This is the version control register. (R/W)

Register 23.31: UHCI_CONF0_REG (0x0000)

Bit(s)	Field	Reset
31:24	(reserved)	0 0 0 0 0 0 0 0
23	UHCI_UART_RX_BRK_EOF_EN	0
22	UHCI_CLK_EN	0
21	UHCI_ENCODE_CRC_EN	1
20	UHCI_LEN_EOF_EN	1
19	UHCI_UART_IDLE_EOF_EN	0
18	UHCI_CRC_REC_EN	1
17	UHCI_HEAD_EN	1
16	UHCI_SEPER_EN	1
15	UHCI_MEM_TRANS_EN	0
14	(reserved)	0
13	UHCI_INDSCR_BURST_EN	0
12	UHCI_OUTDSCR_BURST_EN	0
11	(reserved)	0
10	UHCI_UART1_CE	0
9	UHCI_UART0_CE	0
8	UHCI_OUT_EOF_MODE	1
7	UHCI_OUT_NO_RESTART_CLR	0
6	UHCI_OUT_AUTO_WRBACK	0
5	UHCI_OUT_LOOP_TEST	0
4	UHCI_IN_LOOP_TEST	0
3	UHCI_AHBM_RST	0
2	UHCI_AHBM_FIFO_RST	0
1	UHCI_OUT_RST	0
0	UHCI_IN_RST	0

UHCI_IN_RST Set this bit to reset in DMA FSM. (R/W)

UHCI_OUT_RST Set this bit to reset out DMA FSM. (R/W)

UHCI_AHBM_FIFO_RST Set this bit to reset AHB interface cmdFIFO of DMA. (R/W)

UHCI_AHBM_RST Set this bit to reset AHB interface of DMA. (R/W)

UHCI_IN_LOOP_TEST Reserved. (R/W)

UHCI_OUT_LOOP_TEST Reserved. (R/W)

UHCI_OUT_AUTO_WRBACK Set this bit to enable automatic outlink-writeback when all the data in TX Buffer has been transmitted. (R/W)

UHCI_OUT_NO_RESTART_CLR Reserved. (R/W)

UHCI_OUT_EOF_MODE This register is used to specify the generation mode of UHCI_OUT_EOF_INT interrupt. 1: When DMA has popped all data from FIFO. 0: When AHB has pushed all data to FIFO. (R/W)

UHCI_UART0_CE Set this bit to link up HCI and UART0. (R/W)

UHCI_UART1_CE Set this bit to link up HCI and UART1. (R/W)

UHCI_OUTDSCR_BURST_EN This register is used to specify DMA transmit descriptor transfer mode. 1: burst mode. 0: byte mode. (R/W)

UHCI_INDSCR_BURST_EN This register is used to specify DMA receive descriptor transfer mode. 1: burst mode. 0: byte mode. (R/W)

UHCI_MEM_TRANS_EN 1: UHCI transmitted data would be write back into DMA INFIFO. (R/W)

UHCI_SEPER_EN Set this bit to separate the data frame using a special char. (R/W)

UHCI_HEAD_EN Set this bit to encode the data packet with a formatting header. (R/W)

UHCI_CRC_REC_EN Set this bit to enable UHCI to receive the 16 bit CRC. (R/W)

UHCI_UART_IDLE_EOF_EN If this bit is set to 1, UHCI will end the payload receiving process when UART has been in idle state. (R/W)

Continued on the next page...

Register 23.31: UHCI_CONF0_REG (0x0000)

Continued from the previous page...

UHCI_LEN_EOF_EN If this bit is set to 1, UHCI decoder receiving payload data is end when the receiving byte count has reached the specified value. The value is payload length indicated by UCHI packet header when UHCI_HEAD_EN is 1 or the value is a configuration value when UHCI_HEAD_EN is 0. If this bit is set to 0, UHCI decoder receiving payload data is end when 0xc0 is received. (R/W)

UHCI_ENCODE_CRC_EN Set this bit to enable data integrity checking by appending a 16 bit CCITT-CRC to the end of the payload. (R/W)

UHCI_CLK_EN 1’b1: Force clock on for register. 1’b0: Support clock only when application writes registers. (R/W)

UHCI_UART_RX_BRK_EOF_EN if this bit is set to 1, UHCI will end payload_rec process when NULL frame is received by UART. (R/W)

Register 23.32: UHCI_CONF1_REG (0x002C)

31..21	20..9	8	7	6	5	4	3	2	1	0	
(reserved)	UHCI_DMA_INFIFO_FULL_THRS	UHCI_SW_START	UHCI_WAIT_SW_START	UHCI_CHECK_OWNER	UHCI_TX_ACK_NUM_RE	UHCI_TX_CHECK_SUM_RE	UHCI_SAVE_HEAD	UHCI_CRC_DISABLE	UHCI_CHECK_SEQ_EN	UHCI_CHECK_SUM_EN	
0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	1	1	0	0	1	1	Reset

UHCI_CHECK_SUM_EN This is the enable bit to check header checksum when UHCI receives a data packet. (R/W)

UHCI_CHECK_SEQ_EN This is the enable bit to check sequence number when UHCI receives a data packet. (R/W)

UHCI_CRC_DISABLE Set this bit to support CRC calculation. Data Integrity Check Present bit in UHCI packet frame should be 1. (R/W)

UHCI_SAVE_HEAD Set this bit to save the packet header when HCI receives a data packet. (R/W)

UHCI_TX_CHECK_SUM_RE Set this bit to encode the data packet with a checksum. (R/W)

UHCI_TX_ACK_NUM_RE Set this bit to encode the data packet with an acknowledgement when a reliable packet is to be transmit. (R/W)

UHCI_CHECK_OWNER 1: Check the link list descriptor when link list owner is DMA controller; 0: Always check link list descriptor. (R/W)

UHCI_WAIT_SW_START The uhci-encoder will jump to ST_SW_WAIT status if this register is set to 1. (R/W)

UHCI_SW_START If current UHCI_ENCODE_STATE is ST_SW_WAIT, the UHCI will start to send data packet out when this bit is set to 1. (R/W)

UHCI_DMA_INFIFO_FULL_THRS This field is used to generate the UHCI_DMA_INFIFO_FULL_WM_INT interrupt when the counter value of DMA RX FIFO exceeds the value of the register. (R/W)

Register 23.33: UHCI_AHB_TEST_REG (0x0048)

(reserved)	UHCI_AHB_TESTADDR	(reserved)	UHCI_AHB_TESTMODE	
31 ... 6	5 ... 4	3	2 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	Reset

UHCI_AHB_TESTMODE Reserved. (R/W)

UHCI_AHB_TESTADDR Reserved. (R/W)

Register 23.34: UHCI_ESCAPE_CONF_REG (0x0064)

(reserved)	UHCI_RX_13_ESC_EN	UHCI_RX_11_ESC_EN	UHCI_RX_DB_ESC_EN	UHCI_RX_C0_ESC_EN	UHCI_TX_13_ESC_EN	UHCI_TX_11_ESC_EN	UHCI_TX_DB_ESC_EN	UHCI_TX_C0_ESC_EN	
31 ... 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	1	1	0	0	1	1	Reset

UHCI_TX_C0_ESC_EN Set this bit to enable decoding char 0xc0 when DMA receives data. (R/W)

UHCI_TX_DB_ESC_EN Set this bit to enable decoding char 0xdb when DMA receives data. (R/W)

UHCI_TX_11_ESC_EN Set this bit to enable decoding flow control char 0x11 when DMA receives data. (R/W)

UHCI_TX_13_ESC_EN Set this bit to enable decoding flow control char 0x13 when DMA receives data. (R/W)

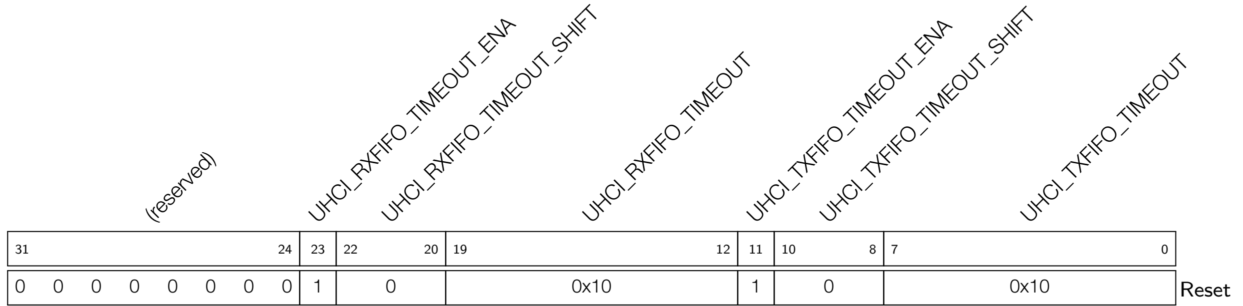
UHCI_RX_C0_ESC_EN Set this bit to enable replacing 0xc0 by special char when DMA sends data. (R/W)

UHCI_RX_DB_ESC_EN Set this bit to enable replacing 0xdb by special char when DMA sends data. (R/W)

UHCI_RX_11_ESC_EN Set this bit to enable replacing flow control char 0x11 by special char when DMA sends data. (R/W)

UHCI_RX_13_ESC_EN Set this bit to enable replacing flow control char 0x13 by special char when DMA sends data. (R/W)

Register 23.35: UHCI_HUNG_CONF_REG (0x0068)



UHCI_TXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_TX_HUNG_INT interrupt when DMA takes more time to receive data. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

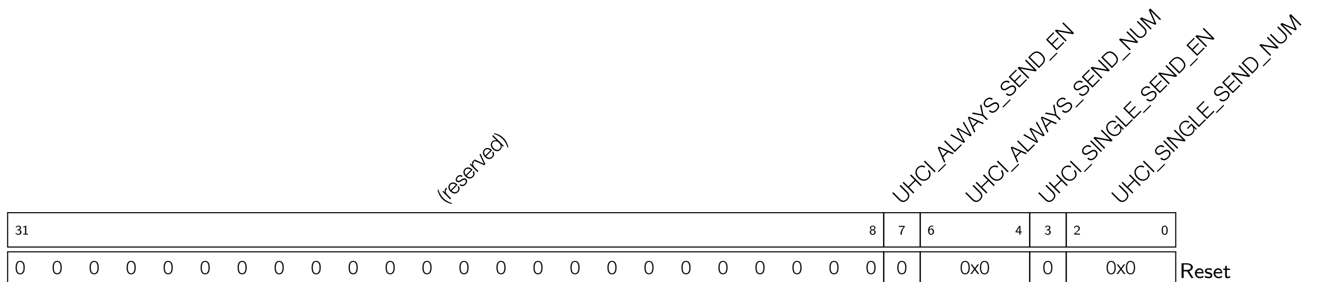
UHCI_TXFIFO_TIMEOUT_ENA This is the enable bit for Tx-FIFO receive-data timeout. (R/W)

UHCI_RXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_RX_HUNG_INT interrupt when DMA takes more time to read data from RAM. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA This is the enable bit for DMA send-data timeout. (R/W)

Register 23.36: UHCI_QUICK_SENT_REG (0x0074)



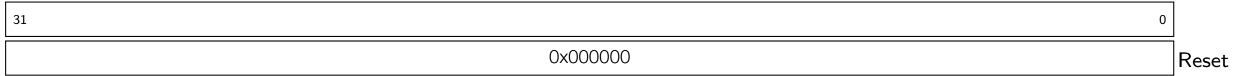
UHCI_SINGLE_SEND_NUM This register is used to specify the single_send register. (R/W)

UHCI_SINGLE_SEND_EN Set this bit to enable single_send mode to send short packet. (R/W)

UHCI_ALWAYS_SEND_NUM This register is used to specify the always_send register. (R/W)

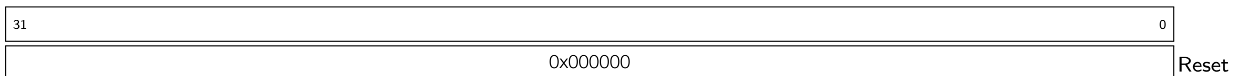
UHCI_ALWAYS_SEND_EN Set this bit to enable always_send mode to send short packet. (R/W)

Register 23.37: UHCI_Q0_WORD0_REG (0x0078)



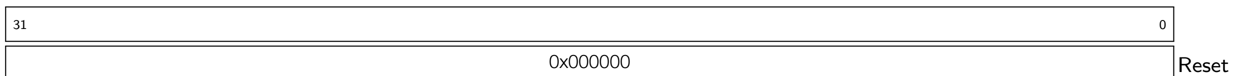
UHCI_SEND_Q0_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.38: UHCI_Q0_WORD1_REG (0x007C)



UHCI_SEND_Q0_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

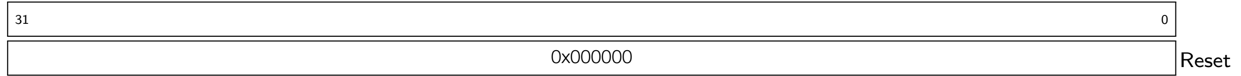
Register 23.39: UHCI_Q1_WORD0_REG (0x0080)



UHCI_SEND_Q1_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.40: UHCI_Q1_WORD1_REG (0x0084)

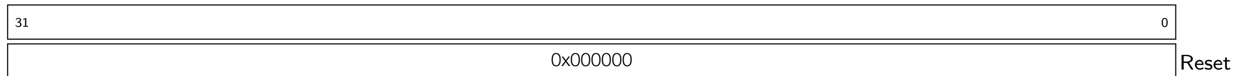
UHCI_SEND_Q1_WORD1



UHCI_SEND_Q1_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.41: UHCI_Q2_WORD0_REG (0x0088)

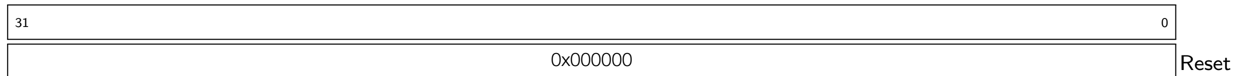
UHCI_SEND_Q2_WORD0



UHCI_SEND_Q2_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

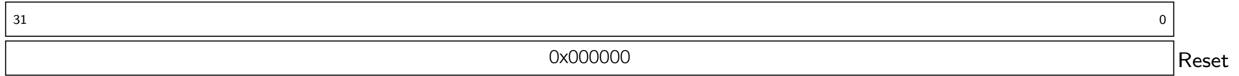
Register 23.42: UHCI_Q2_WORD1_REG (0x008C)

UHCI_SEND_Q2_WORD1



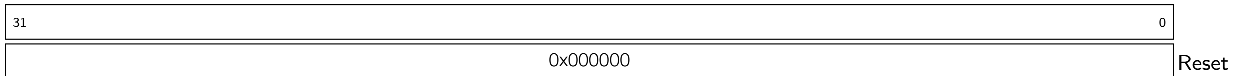
UHCI_SEND_Q2_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.43: UHCI_Q3_WORD0_REG (0x0090)



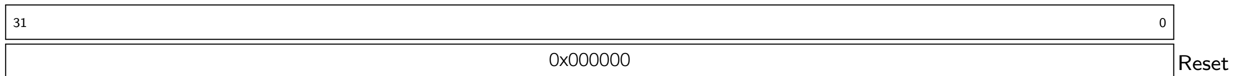
UHCI_SEND_Q3_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.44: UHCI_Q3_WORD1_REG (0x0094)



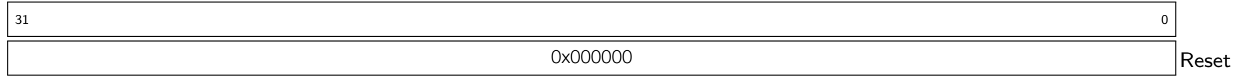
UHCI_SEND_Q3_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.45: UHCI_Q4_WORD0_REG (0x0098)



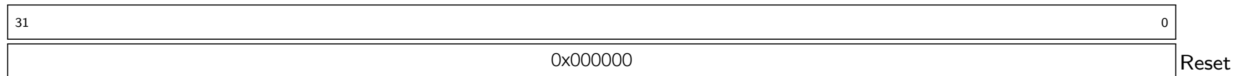
UHCI_SEND_Q4_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.46: UHCI_Q4_WORD1_REG (0x009C)



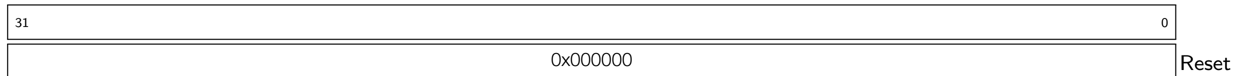
UHCI_SEND_Q4_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.47: UHCI_Q5_WORD0_REG (0x00A0)



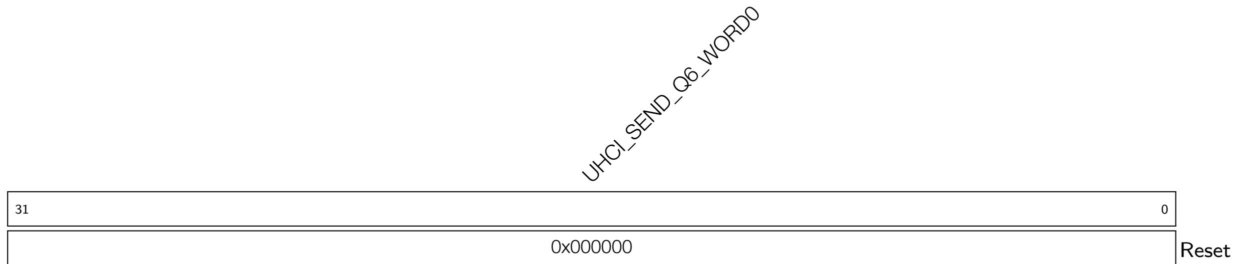
UHCI_SEND_Q5_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.48: UHCI_Q5_WORD1_REG (0x00A4)



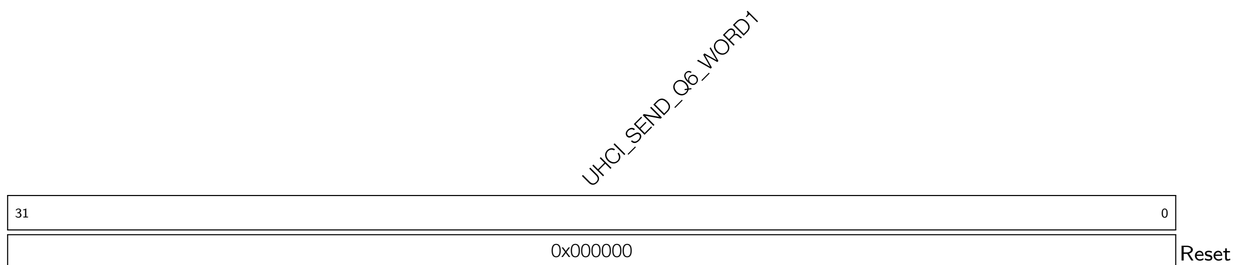
UHCI_SEND_Q5_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.49: UHCI_Q6_WORD0_REG (0x00A8)



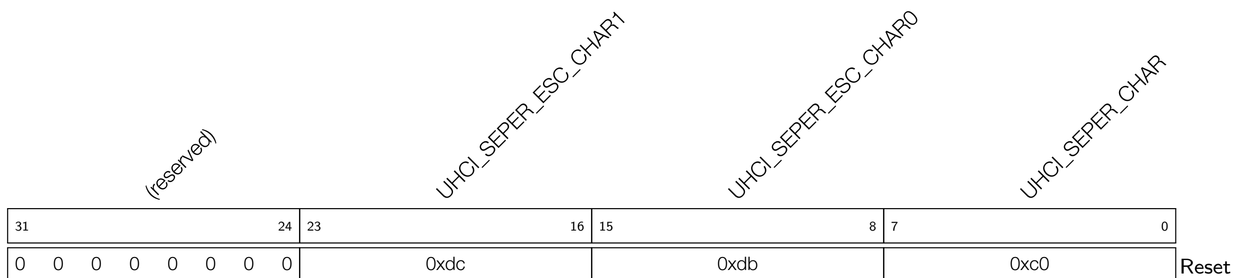
UHCI_SEND_Q6_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.50: UHCI_Q6_WORD1_REG (0x00AC)



UHCI_SEND_Q6_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 23.51: UHCI_ESC_CONF0_REG (0x00B0)



UHCI_SEPER_CHAR This register is used to define the separate char that need to be encoded, default is 0xc0. (R/W)

UHCI_SEPER_ESC_CHAR0 This register is used to define the first char of slip escape sequence when encoding the separate char, default is 0xdb. (R/W)

UHCI_SEPER_ESC_CHAR1 This register is used to define the second char of slip escape sequence when encoding the separate char, default is 0xdc. (R/W)

Register 23.52: UHCI_ESC_CONF1_REG (0x00B4)

(reserved)	UHCI_ESC_SEQ0_CHAR1	UHCI_ESC_SEQ0_CHAR0	UHCI_ESC_SEQ0	
31 ... 24	23 ... 16	15 ... 8	7 ... 0	
0 0 0 0 0 0 0 0	0xdd	0xdb	0xdb	Reset

UHCI_ESC_SEQ0 This register is used to define a char that need to be encoded, default is 0xdb that used as the first char of slip escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdb. (R/W)

UHCI_ESC_SEQ0_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdd. (R/W)

Register 23.53: UHCI_ESC_CONF2_REG (0x00B8)

(reserved)	UHCI_ESC_SEQ1_CHAR1	UHCI_ESC_SEQ1_CHAR0	UHCI_ESC_SEQ1	
31 ... 24	23 ... 16	15 ... 8	7 ... 0	
0 0 0 0 0 0 0 0	0xde	0xdb	0x11	Reset

UHCI_ESC_SEQ1 This register is used to define a char that need to be encoded, default is 0x11 that used as flow control char. (R/W)

UHCI_ESC_SEQ1_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xdb. (R/W)

UHCI_ESC_SEQ1_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xde. (R/W)

Register 23.54: UHCI_ESC_CONF3_REG (0x00BC)

(reserved)	UHCI_ESC_SEQ2_CHAR1	UHCI_ESC_SEQ2_CHAR0	UHCI_ESC_SEQ2	
31 ... 24	23 ... 16	15 ... 8	7 ... 0	
0 0 0 0 0 0 0 0	0xdf	0xdb	0x13	Reset

UHCI_ESC_SEQ2 This register is used to define a char that need to be decoded, default is 0x13 that used as flow control char. (R/W)

UHCI_ESC_SEQ2_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdb. (R/W)

UHCI_ESC_SEQ2_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdf. (R/W)

Register 23.55: UHCI_PKT_THRES_REG (0x00C0)

(reserved)	UHCI_PKT_THRS	
31 ... 13	12 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x80	Reset

UHCI_PKT_THRS This register is used to configure the maximum value of the packet length when UHCI_HEAD_EN is 0. (R/W)

Register 23.56: UHCI_INT_RAW_REG (0x0004)

31 ... 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	UHCI_DMA_INFIFO_FULL_WM_INT_RAW	UHCI_SEND_A_REG_Q_INT_RAW	UHCI_SEND_S_REG_Q_INT_RAW	UHCI_OUT_TOTAL_EOF_INT_RAW	UHCI_OUTLINK_EOF_ERR_INT_RAW	UHCI_IN_DSCR_EMPTY_INT_RAW	UHCI_OUT_DSCR_ERR_INT_RAW	UHCI_IN_DSCR_ERR_INT_RAW	UHCI_OUT_EOF_INT_RAW	UHCI_OUT_DONE_INT_RAW	UHCI_IN_ERR_EOF_INT_RAW	UHCI_IN_SUC_EOF_INT_RAW	UHCI_IN_DONE_INT_RAW	UHCI_TX_HUNG_INT_RAW	UHCI_RX_HUNG_INT_RAW	UHCI_TX_START_INT_RAW	UHCI_RX_START_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UHCI_RX_START_INT_RAW This is the interrupt raw bit. Triggered when a separator char has been sent. (RO)

UHCI_TX_START_INT_RAW This is the interrupt raw bit. Triggered when DMA detects a separator char. (RO)

UHCI_RX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when DMA takes more time to receive data than the configure value. (RO)

UHCI_TX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when DMA takes more time to read data from RAM than the configured value. (RO)

UHCI_IN_DONE_INT_RAW This is the interrupt raw bit. Triggered when an receive descriptor is completed. (RO)

UHCI_IN_SUC_EOF_INT_RAW This is the interrupt raw bit. Triggered when a data packet has been received successfully. (RO)

UHCI_IN_ERR_EOF_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in EOF in the receive descriptor. (RO)

UHCI_OUT_DONE_INT_RAW This is the interrupt raw bit. Triggered when an transmit descriptor is completed. (RO)

UHCI_OUT_EOF_INT_RAW This is the interrupt raw bit. Triggered when the current descriptor's EOF bit is 1. (RO)

UHCI_IN_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in the receive descriptor. (RO)

UHCI_OUT_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in the transmit descriptor. (RO)

UHCI_IN_DSCR_EMPTY_INT_RAW This is the interrupt raw bit. Triggered when there are not enough inlinks for DMA. (RO)

UHCI_OUTLINK_EOF_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in EOF in the transmit descriptor. (RO)

Continued on the next page...

Register 23.56: UHCI_INT_RAW_REG (0x0004)

Continued from the previous page...

UHCI_OUT_TOTAL_EOF_INT_RAW This is the interrupt raw bit. Triggered when all data in the last buffer address has been sent out. (RO)

UHCI_SEND_S_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when DMA has sent out a short packet using single_send registers. (RO)

UHCI_SEND_A_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when DMA has sent out a short packet using always_send registers. (RO)

UHCI_DMA_INFIFO_FULL_WM_INT_RAW This is the interrupt raw bit. Triggered when the DMA INFIFO count has reached the configured threshold value. (RO)

Register 23.57: UHCI_INT_ST_REG (0x0008)

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	UHCI_DMA_INFIFO_FULL_WM_INT_ST	UHCI_SEND_A_REG_Q_INT_ST	UHCI_SEND_S_REG_Q_INT_ST	UHCI_OUT_TOTAL_EOF_INT_ST	UHCI_OUTLINK_EOF_ERR_INT_ST	UHCI_IN_DSCR_EMPTY_INT_ST	UHCI_OUT_DSCR_ERR_INT_ST	UHCI_IN_DSCR_ERR_INT_ST	UHCI_OUT_EOF_INT_ST	UHCI_OUT_DONE_INT_ST	UHCI_IN_ERR_EOF_INT_ST	UHCI_IN_SUC_EOF_INT_ST	UHCI_IN_DONE_INT_ST	UHCI_TX_HUNG_INT_ST	UHCI_RX_HUNG_INT_ST	UHCI_TX_START_INT_ST	UHCI_RX_START_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UHCI_RX_START_INT_ST This is the masked interrupt bit for UHCI_RX_START_INT interrupt when UHCI_RX_START_INT_ENA is set to 1. (RO)

UHCI_TX_START_INT_ST This is the masked interrupt bit for UHCI_TX_START_INT interrupt when UHCI_TX_START_INT_ENA is set to 1. (RO)

UHCI_RX_HUNG_INT_ST This is the masked interrupt bit for UHCI_RX_HUNG_INT interrupt when UHCI_RX_HUNG_INT_ENA is set to 1. (RO)

UHCI_TX_HUNG_INT_ST This is the masked interrupt bit for UHCI_TX_HUNG_INT interrupt when UHCI_TX_HUNG_INT_ENA is set to 1. (RO)

UHCI_IN_DONE_INT_ST This is the masked interrupt bit for UHCI_IN_DONE_INT interrupt when UHCI_IN_DONE_INT_ENA is set to 1. (RO)

UHCI_IN_SUC_EOF_INT_ST This is the masked interrupt bit for UHCI_IN_SUC_EOF_INT interrupt when UHCI_IN_SUC_EOF_INT_ENA is set to 1. (RO)

UHCI_IN_ERR_EOF_INT_ST This is the masked interrupt bit for UHCI_IN_ERR_EOF_INT interrupt when UHCI_IN_ERR_EOF_INT_ENA is set to 1. (RO)

UHCI_OUT_DONE_INT_ST This is the masked interrupt bit for UHCI_OUT_DONE_INT interrupt when UHCI_OUT_DONE_INT_ENA is set to 1. (RO)

UHCI_OUT_EOF_INT_ST This is the masked interrupt bit for UHCI_OUT_EOF_INT interrupt when UHCI_OUT_EOF_INT_ENA is set to 1. (RO)

UHCI_IN_DSCR_ERR_INT_ST This is the masked interrupt bit for UHCI_IN_DSCR_ERR_INT interrupt when UHCI_IN_DSCR_ERR_INT is set to 1. (RO)

UHCI_OUT_DSCR_ERR_INT_ST This is the masked interrupt bit for UHCI_OUT_DSCR_ERR_INT interrupt when UHCI_OUT_DSCR_ERR_INT_ENA is set to 1. (RO)

UHCI_IN_DSCR_EMPTY_INT_ST This is the masked interrupt bit for UHCI_IN_DSCR_EMPTY_INT interrupt when UHCI_IN_DSCR_EMPTY_INT_ENA is set to 1. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST This is the masked interrupt bit for UHCI_OUTLINK_EOF_ERR_INT interrupt when UHCI_OUTLINK_EOF_ERR_INT_ENA is set to 1. (RO)

Continued on the next page...

Register 23.57: UHCI_INT_ST_REG (0x0008)

Continued from the previous page...

UHCI_OUT_TOTAL_EOF_INT_ST This is the masked interrupt bit for UHCI_OUT_TOTAL_EOF_INT interrupt when UHCI_OUT_TOTAL_EOF_INT_ENA is set to 1. (RO)

UHCI_SEND_S_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_S_REG_Q_INT interrupt when UHCI_SEND_S_REG_Q_INT_ENA is set to 1. (RO)

UHCI_SEND_A_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_A_REG_Q_INT interrupt when UHCI_SEND_A_REG_Q_INT_ENA is set to 1. (RO)

UHCI_DMA_INFIFO_FULL_WM_INT_ST This is the masked interrupt bit for UHCI_DMA_INFIFO_FULL_WM_INT INTERRUPT when UHCI_DMA_INFIFO_FULL_WM_INT_ENA is set to 1. (RO)

Register 23.58: UHCI_INT_ENA_REG (0x000C)

Bits	31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	(reserved)	UHCI_DMA_INFIFO_FULL_WM_INT_ENA	UHCI_SEND_A_REG_Q_INT_ENA	UHCI_SEND_S_REG_Q_INT_ENA	UHCI_OUT_TOTAL_EOF_INT_ENA	UHCI_OUTLINK_EOF_ERR_INT_ENA	UHCI_IN_DSCR_EMPTY_INT_ENA	UHCI_OUT_DSCR_ERR_INT_ENA	UHCI_IN_DSCR_ERR_INT_ENA	UHCI_OUT_EOF_INT_ENA	UHCI_OUT_DONE_INT_ENA	UHCI_IN_ERR_EOF_INT_ENA	UHCI_IN_SUC_EOF_INT_ENA	UHCI_IN_DONE_INT_ENA	UHCI_TX_HUNG_INT_ENA	UHCI_RX_HUNG_INT_ENA	UHCI_TX_START_INT_ENA	UHCI_RX_START_INT_ENA
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

UHCI_RX_START_INT_ENA This is the interrupt enable bit for UHCI_RX_START_INT interrupt. (R/W)

UHCI_TX_START_INT_ENA This is the interrupt enable bit for UHCI_TX_START_INT interrupt. (R/W)

UHCI_RX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_RX_HUNG_INT interrupt. (R/W)

UHCI_TX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_TX_HUNG_INT interrupt. (R/W)

UHCI_IN_DONE_INT_ENA This is the interrupt enable bit for UHCI_IN_DONE_INT interrupt. (R/W)

UHCI_IN_SUC_EOF_INT_ENA This is the interrupt enable bit for UHCI_IN_SUC_EOF_INT interrupt. (R/W)

UHCI_IN_ERR_EOF_INT_ENA This is the interrupt enable bit for UHCI_IN_ERR_EOF_INT interrupt. (R/W)

UHCI_OUT_DONE_INT_ENA This is the interrupt enable bit for UHCI_OUT_DONE_INT interrupt. (R/W)

UHCI_OUT_EOF_INT_ENA This is the interrupt enable bit for UHCI_OUT_EOF_INT interrupt. (R/W)

UHCI_IN_DSCR_ERR_INT_ENA This is the interrupt enable bit for UHCI_IN_DSCR_ERR_INT interrupt. (R/W)

UHCI_OUT_DSCR_ERR_INT_ENA This is the interrupt enable bit for UHCI_OUT_DSCR_ERR_INT interrupt. (R/W)

UHCI_IN_DSCR_EMPTY_INT_ENA This is the interrupt enable bit for UHCI_IN_DSCR_EMPTY_INT interrupt. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA This is the interrupt enable bit for UHCI_OUTLINK_EOF_ERR_INT interrupt. (R/W)

UHCI_OUT_TOTAL_EOF_INT_ENA This is the interrupt enable bit for UHCI_OUT_TOTAL_EOF_INT interrupt. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_S_REG_Q_INT interrupt. (R/W)

Continued on the next page...

Register 23.58: UHCI_INT_ENA_REG (0x000C)

Continued from the previous page...

UHCI_SEND_A_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_A_REG_Q_INT interrupt. (R/W)

UHCI_DMA_INFIFO_FULL_WM_INT_ENA This is the interrupt enable bit for UHCI_DMA_INFIFO_FULL_WM_INT interrupt. (R/W)

Register 23.59: UHCI_INT_CLR_REG (0x0010)

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	UHCI_DMA_INFIFO_FULL_WM_INT_CLR	UHCI_SEND_A_REG_Q_INT_CLR	UHCI_SEND_S_REG_Q_INT_CLR	UHCI_OUT_TOTAL_EOF_INT_CLR	UHCI_OUTLINK_EOF_ERR_INT_CLR	UHCI_IN_DSCR_EMPTY_INT_CLR	UHCI_OUT_DSCR_ERR_INT_CLR	UHCI_IN_DSCR_ERR_INT_CLR	UHCI_OUT_EOF_INT_CLR	UHCI_OUT_DONE_INT_CLR	UHCI_IN_ERR_EOF_INT_CLR	UHCI_IN_SUC_EOF_INT_CLR	UHCI_IN_DONE_INT_CLR	UHCI_TX_HUNG_INT_CLR	UHCI_RX_HUNG_INT_CLR	UHCI_TX_START_INT_CLR	UHCI_RX_START_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UHCI_RX_START_INT_CLR Set this bit to clear UHCI_RX_START_INT interrupt. (WO)

UHCI_TX_START_INT_CLR Set this bit to clear UHCI_TX_START_INT interrupt. (WO)

UHCI_RX_HUNG_INT_CLR Set this bit to clear UHCI_RX_HUNG_INT interrupt. (WO)

UHCI_TX_HUNG_INT_CLR Set this bit to clear UHCI_TX_HUNG_INT interrupt. (WO)

UHCI_IN_DONE_INT_CLR Set this bit to clear UHCI_IN_DONE_INT interrupt. (WO)

UHCI_IN_SUC_EOF_INT_CLR Set this bit to clear UHCI_IN_SUC_EOF_INT interrupt. (WO)

UHCI_IN_ERR_EOF_INT_CLR Set this bit to clear UHCI_IN_ERR_EOF_INT interrupt. (WO)

UHCI_OUT_DONE_INT_CLR Set this bit to clear UHCI_OUT_DONE_INT interrupt. (WO)

UHCI_OUT_EOF_INT_CLR Set this bit to clear UHCI_OUT_EOF_INT interrupt. (WO)

UHCI_IN_DSCR_ERR_INT_CLR Set this bit to clear UHCI_IN_DSCR_ERR_INT interrupt. (WO)

UHCI_OUT_DSCR_ERR_INT_CLR Set this bit to clear UHCI_OUT_DSCR_ERR_INT interrupt. (WO)

UHCI_IN_DSCR_EMPTY_INT_CLR Set this bit to clear UHCI_IN_DSCR_EMPTY_INT interrupt. (WO)

UHCI_OUTLINK_EOF_ERR_INT_CLR Set this bit to clear UHCI_OUTLINK_EOF_ERR_INT interrupt. (WO)

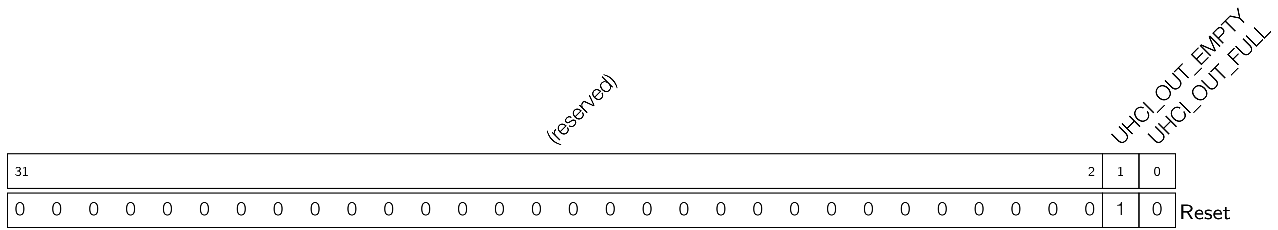
UHCI_OUT_TOTAL_EOF_INT_CLR Set this bit to clear UHCI_OUT_TOTAL_EOF_INT interrupt. (WO)

UHCI_SEND_S_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_S_UHCI_Q_INT interrupt. (WO)

UHCI_SEND_A_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_A_UHCI_Q_INT interrupt. (WO)

UHCI_DMA_INFIFO_FULL_WM_INT_CLR Set this bit to clear UHCI_DMA_INFIFO_FULL_WM_INT interrupt. (WO)

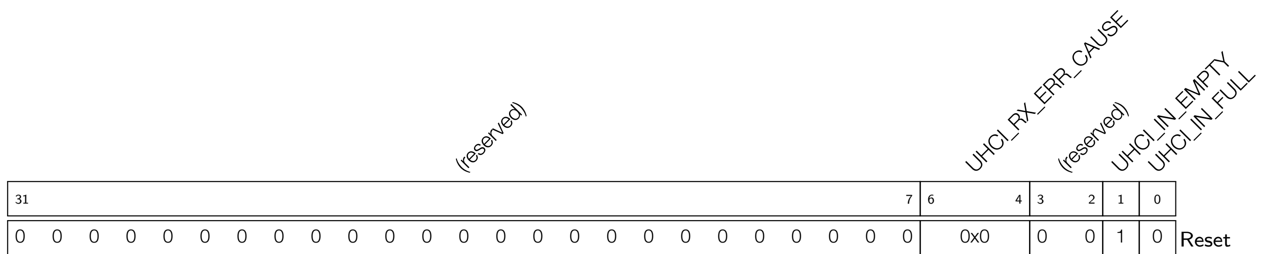
Register 23.60: UHCI_DMA_OUT_STATUS_REG (0x0014)



UHCI_OUT_FULL 1: DMA data-output FIFO is full. (RO)

UHCI_OUT_EMPTY 1: DMA data-output FIFO is empty. (RO)

Register 23.61: UHCI_DMA_IN_STATUS_REG (0x001C)



UHCI_IN_FULL Data-input FIFO full signal. (RO)

UHCI_IN_EMPTY Data-input FIFO empty signal. (RO)

UHCI_RX_ERR_CAUSE This register indicates the error type when DMA has received a packet with error. 3'b001: Checksum error in HCI packet; 3'b010: Sequence number error in HCI packet; 3'b011: CRC bit error in HCI packet; 3'b100: 0xc0 is found but received HCI packet is not end; 3'b101: 0xc0 is not found when receiving HCI packet is end; 3'b110: CRC check error. (RO)

Register 23.62: UHCI_STATE0_REG (0x0030)

(reserved)	UHCI_DECODE_STATE	UHCI_INFIFO_CNT_DEBUG	UHCI_IN_STATE	UHCI_IN_DSCR_STATE	UHCI_INLINK_DSCR_ADDR	
31	30 28	27 23	22 20	19 18	17 0	
0	0	0	0	0	0	Reset

UHCI_INLINK_DSCR_ADDR This register stores the current receive descriptor's address. (RO)

UHCI_IN_DSCR_STATE Reserved. (RO)

UHCI_IN_STATE Reserved. (RO)

UHCI_INFIFO_CNT_DEBUG This register stores the byte number of the data in the receive descriptor's FIFO. (RO)

UHCI_DECODE_STATE UHCI decoder status. (RO)

Register 23.63: UHCI_STATE1_REG (0x0034)

(reserved)	UHCI_ENCODE_STATE	UHCI_OUTFIFO_CNT	UHCI_OUT_STATE	UHCI_OUT_DSCR_STATE	UHCI_OUTLINK_DSCR_ADDR	
31	30 28	27 23	22 20	19 18	17 0	
0	0	0	0	0	0	Reset

UHCI_OUTLINK_DSCR_ADDR This register stores the current transmit descriptor's address. (RO)

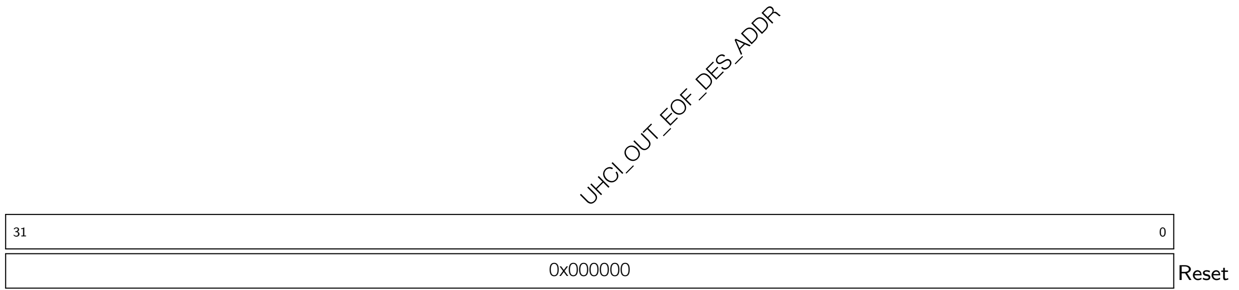
UHCI_OUT_DSCR_STATE Reserved. (RO)

UHCI_OUT_STATE Reserved. (RO)

UHCI_OUTFIFO_CNT This register stores the byte number of the data in the transmit descriptor's FIFO. (RO)

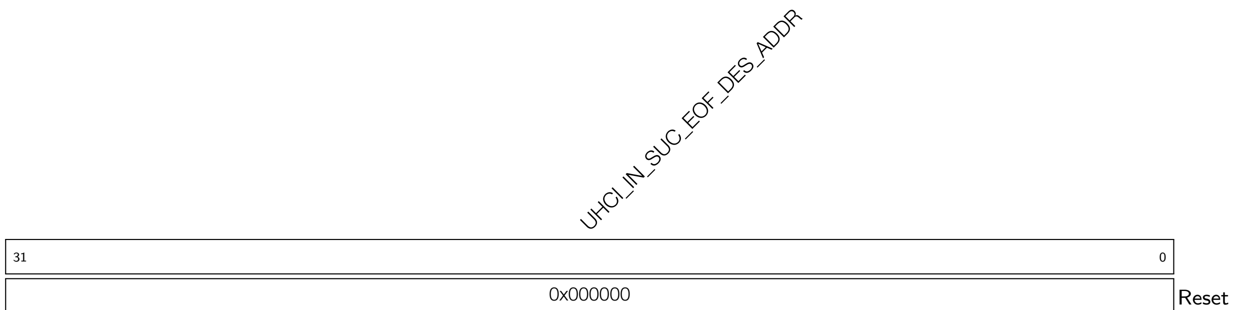
UHCI_ENCODE_STATE UHCI encoder status. (RO)

Register 23.64: UHCI_DMA_OUT_EOF_DES_ADDR_REG (0x0038)



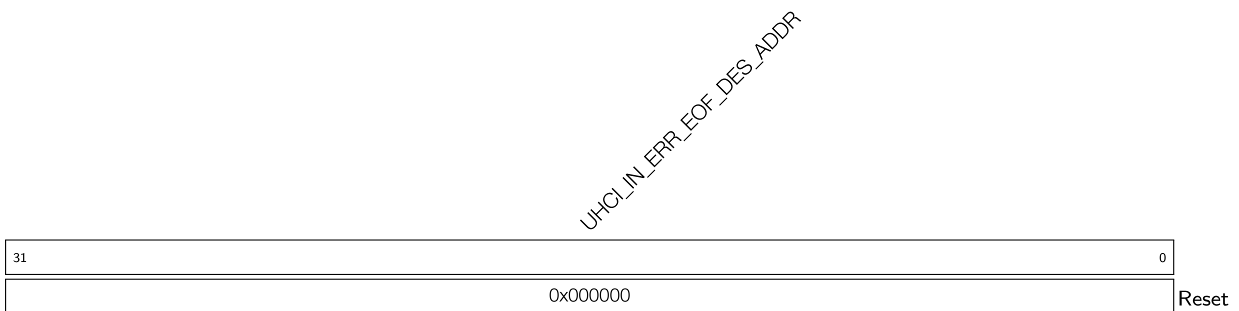
UHCI_OUT_EOF_DES_ADDR This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 23.65: UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG (0x003C)



UHCI_IN_SUC_EOF_DES_ADDR This register stores the address of the receive descriptor when received successful EOF. (RO)

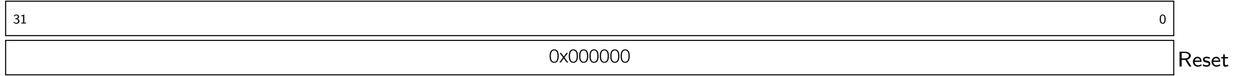
Register 23.66: UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG (0x0040)



UHCI_IN_ERR_EOF_DES_ADDR This register stores the address of the receive descriptor when there are some errors in this descriptor. (RO)

Register 23.67: UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x0044)

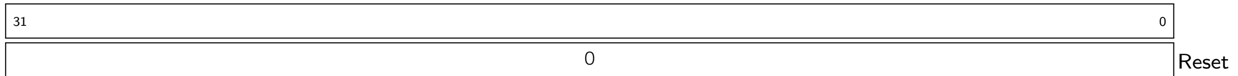
UHCI_OUT_EOF_BFR_DES_ADDR



UHCI_OUT_EOF_BFR_DES_ADDR This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 23.68: UHCI_DMA_IN_DSCR_REG (0x004C)

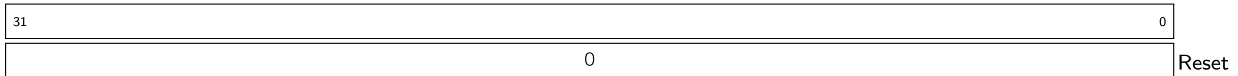
UHCI_INLINK_DSCR



UHCI_INLINK_DSCR This register stores the third word of the next receive descriptor. (RO)

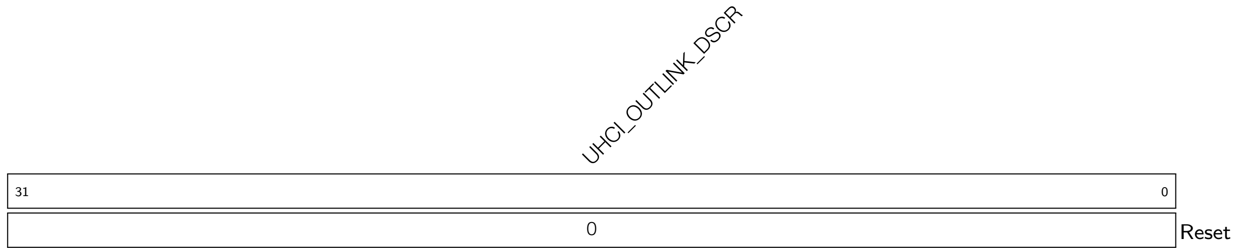
Register 23.69: UHCI_DMA_IN_DSCR_BF0_REG (0x0050)

UHCI_INLINK_DSCR_BF0



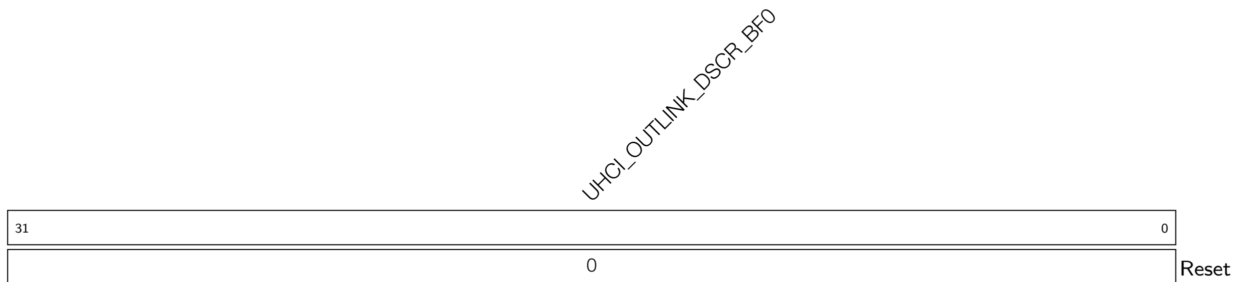
UHCI_INLINK_DSCR_BF0 This register stores the third word of the current receive descriptor. (RO)

Register 23.70: UHCI_DMA_OUT_DSCR_REG (0x0058)



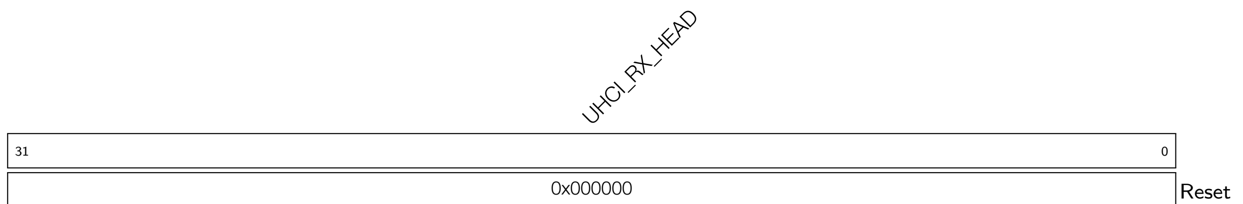
UHCI_OUTLINK_DSCR This register stores the third word of the next transmit descriptor. (RO)

Register 23.71: UHCI_DMA_OUT_DSCR_BF0_REG (0x005C)



UHCI_OUTLINK_DSCR_BF0 This register stores the third word of the current transmit descriptor. (RO)

Register 23.72: UHCI_RX_HEAD_REG (0x0070)



UHCI_RX_HEAD This register stores the header of the current received packet. (RO)

Register 23.73: UHCI_DMA_OUT_PUSH_REG (0x0018)

(reserved)	UHCI_OUTFIFO_PUSH	(reserved)	UHCI_OUTFIFO_WDATA	
31 … 17	16	15 … 9	8 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0	0x0	Reset

UHCI_OUTFIFO_WDATA This is the data that need to be pushed into data-output FIFO. (R/W)

UHCI_OUTFIFO_PUSH Set this bit to push data into the data-output FIFO. (R/W)

Register 23.74: UHCI_DMA_IN_POP_REG (0x0020)

(reserved)	UHCI_INFIFO_POP	(reserved)	UHCI_INFIFO_RDATA	
31 … 17	16	15 … 12	11 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0	0x0	Reset

UHCI_INFIFO_RDATA This register stores the data popping from data-input FIFO. (RO)

UHCI_INFIFO_POP Set this bit to pop data from data-input FIFO. (R/W)

Register 23.75: UHCI_DMA_OUT_LINK_REG (0x0024)

UHCI_OUTLINK_PARK	UHCI_OUTLINK_RESTART	UHCI_OUTLINK_START	UHCI_OUTLINK_STOP	(reserved)	UHCI_OUTLINK_ADDR	
31	30	29	28	27 … 20	19 … 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000	Reset

UHCI_OUTLINK_ADDR This register is used to specify the least significant 20 bits of the first transmit descriptor's address. (R/W)

UHCI_OUTLINK_STOP Set this bit to stop dealing with the transmit descriptor. (R/W)

UHCI_OUTLINK_START Set this bit to start a new transmit descriptor. (R/W)

UHCI_OUTLINK_RESTART Set this bit to restart the transmit descriptor from the last address. (R/W)

UHCI_OUTLINK_PARK 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

Register 23.76: UHCI_DMA_IN_LINK_REG (0x0028)

UHCI_INLINK_PARK	UHCI_INLINK_RESTART	UHCI_INLINK_START	UHCI_INLINK_STOP	(reserved)	UHCI_INLINK_AUTO_RET	UHCI_INLINK_ADDR	
31	30	29	28	27 21	20	19 0	
0	0	0	0	0 0 0 0 0 0 0	1	0x000	Reset

UHCI_INLINK_ADDR This register is used to specify the least significant 20 bits of the first receive descriptor's address. (R/W)

UHCI_INLINK_AUTO_RET This is the enable bit to return to current receive descriptor's address, when there are some errors in current packet. (R/W)

UHCI_INLINK_STOP Set this bit to stop dealing with the receive descriptors. (R/W)

UHCI_INLINK_START Set this bit to start dealing with the receive descriptors. (R/W)

UHCI_INLINK_RESTART Set this bit to restart new receive descriptors. (R/W)

UHCI_INLINK_PARK 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

Register 23.77: UHCI_DATE_REG (0x00FC)

UHCI_DATE	
31 0	
0x18073001	Reset

UHCI_DATE This is the version control register. (R/W)

24. SPI Controller (SPI)

24.1 Overview

The ESP32-S2 chip integrates four SPI controllers SPI0, SPI1, General Purpose SPI2 (GP-SPI2), and GP-SPI3 as shown in Figure 24-1.

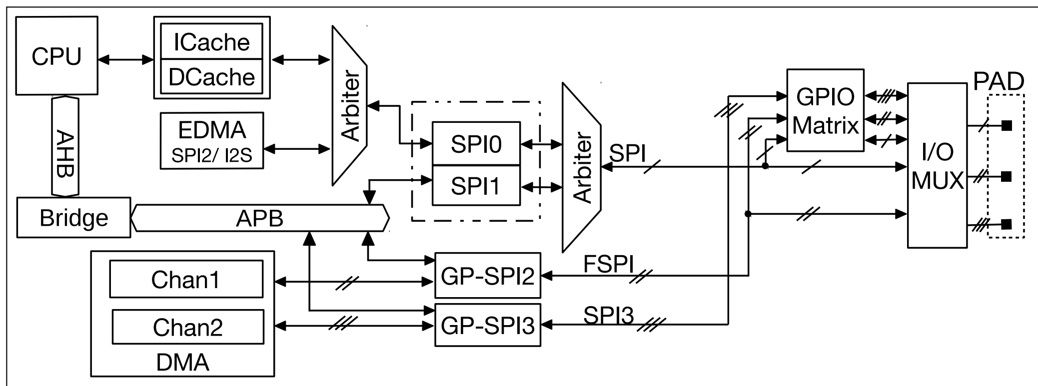


Figure 24-1. SPI Block Diagram

All the four SPI controllers can be used to communicate with external SPI devices:

- SPI0 comes with two chip select (CS) lines: CS0 and CS1, CS0 for flash and CS1 for external RAM. SPI0 is only used by ESP32-S2 cache or EDMA to:
  - Read data from an external RAM.
  - Write data to an external RAM.
  - Read data from an external flash, only applicable for cache.
- SPI1 also comes with two CS lines, CS0 and CS1. SPI1 can be used by the CPU to access various external flash.
- GP-SPI2 is a general purpose SPI controller with its own DMA channel. GP-SPI2 works as either a master or a slave. As a master, GP-SPI2 provides six CS lines, CS0 ~ CS5.
- GP-SPI3 is also a general purpose SPI controller, but shares a DMA channel with ADC and DAC modules. GP-SPI3 works as either a master or a slave. As a master, GP-SPI3 provides three CS lines, CS0 ~ CS2.

SPI0 and SPI1 are used internally and share the same SPI signal bus via an arbiter. The name of signals shared by SPI0 and SPI1 is prefixed with ”SPI”: SPICLK, SPICS0 ~ SPICS1, SPID, SPIQ, SPIWP, SPIHD, SPIIO4 ~ SPIIO7, and SPIDQS. The signal buses for GP-SPI2 and GP-SPI3 are prefixed with ”FSPI” and ”SPI3”, respectively. The prefix FSPI (Fast SPI) indicates that the recommended usage of GP-SPI2 is to communicate with external high-speed SPI devices.

The I/O lines of the SPI/FSPI/SPI3 buses can be mapped to physical GPIO pads by following ways:

- SPI bus signals are routed to GPIO pads via either GPIO Matrix or IO MUX.
- FSPI bus signals are routed to GPIO pads via either GPIO Matrix or IO MUX.
- SPI3 bus signals are routed to GPIO pads via GPIO Matrix only.

For more information, please see Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*.

24.2 Features

SPI0 and SPI1 controllers are primarily reserved for internal use. This section and the following sections mainly focus on GP-SPI2 and GP-SPI3 controllers. For mapping between SPI signals and ESP32-S2 signals, see Table 142.

24.2.1 GP-SPI2 Features

24.2.1.1 Functioning as a Master

- The bits used in CMD, ADDR and DATA phases are SW-configurable.
- Support accessing various SPI devices (such as flash, external RAM, LCD driver) working in different data modes:
  - 1-bit SPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID and/or FSPIQ. In full-duplex communication, 1-bit SPI is supported. In half-duplex communication, 3-line half-duplex SPI is supported, and only one data line FSPID is used; 4-line half-duplex SPI is also supported, and one of the data lines FSPID and FSPIQ is used.
  - 2-bit Dual SPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID, and FSPIQ. Note: in this mode, two data lines, i.e. FSPID and FSPIQ, are used in parallel.
  - 4-bit Quad SPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID, FSPIQ, FSPIWP, and FSPIHD. Note: in this mode, 4 data lines are used in parallel, i.e. FSPID, FSPIQ, FSPIWP, and FSPIHD.
  - QPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID, FSPIQ, FSPIWP, and FSPIHD.
  - 8-bit Octal SPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID, FSPIQ, FSPIWP, FSPIHD, and FSPIIO4 ~ FSPIIO7. Note: in this mode, 8 data lines are used in parallel, i.e. FSPID, FSPIQ, FSPIWP, FSPIHD, and FSPIIO4~FSPIIO7.
  - OPI mode, with signals FSPICLK, FSPICS0 ~ FSPICS5, FSPID, FSPIQ, FSPIWP, FSPIHD, and FSPIIO4 ~ FSPIIO7.
- Support Moto6800/I8080/Parallel RGB interface 8-bit LCD driver.
- Support the frequency of SPI clock: 1 ~ n division of APB clock, see the division formula.
- The level of SPI_CD pin can be set in CMD and ADDR phases.
- Provide six SPI_CS pins for connection with six independent SPI slaves. The master can communicate with these devices with different configurations (data modes, full/half-duplex, command, address, dummy, etc.) successively.
- Support configurable CS setup time and hold time.
- Support single transfer and DMA segmented-configure-transfer (SCT). Have separated interrupts for both of them.

24.2.1.2 Functioning as a Slave

- Support the following data modes. For the bits used in CMD, ADDR and DATA phases in different data modes, see Table 131.
  - 1-bit SPI, with signals FSPICLK, FSPICS0, FSPID and/or FSPIQ. In full-duplex communication, 1-bit SPI is supported. In half-duplex communication, 3-line half-duplex SPI is supported, and uses only

    one data line: FSPID; 4-line half-duplex SPI is also supported, and uses one of the data lines FSPID and FSPIQ.
  - 2-bit Dual SPI, with signals FSPICLK, FSPICS0, FSPID and FSPIQ.
  - 4-bit Qual SPI, with signals FSPICLK, FSPICS0, FSPID, FSPIQ, FSPIWP, and FSPIHD.
  - QPI, with signals FSPICLK, FSPICS0, FSPID, FSPIQ, FSPIWP, and FSPIHD.
- Support the frequency of SPI clock up to 40 MHz.
- Support the communication formats described in Section 24.5.1.
- Support single transfer and segmented-transfer. Have separated interrupts for both of them.

24.2.1.3 Functioning as a Master or a Slave

- Data transmission length configurable. 1 ~ 64 bytes for CPU controlled mode, 1 ~ n (unlimited) bytes for DMA controlled mode.
- Support configurable read and write data bit order: most-significant bit (MSB) first, or least-significant bit (LSB) first.
- Support full-duplex and half-duplex communications.
- Support data exchange between external SPI devices and:
  - GP-SPI2 data buffer (access by CPU)
  - internal RAM (access by DMA)
  - external RAM (access by EDMA)
- Support configurable clock frequency, polarity, and phase.
- Support SPI clock mode 0 ~ 3.

24.2.2 GP-SPI3 Features

24.2.2.1 Functioning as a Master

- Support the frequency of SPI clock: 1 ~ n division of APB clock, see the division formula.
- Support 1-bit LCD driver.
- The level of SPI_CD pin can be set in CMD and ADDR phases.
- Provide three SPI_CS pins for connection of three SPI slaves. The master can communicate with these devices with different configurations (data modes, full/half-duplex, command, address, dummy, etc.) successively.
- Support configurable CS setup time and hold time.
- Support single transfer and DMA segmented-configure-transfer (SCT). Have separated interrupts for both of them.

24.2.2.2 Functioning as a Slave

- Support frequency of SPI clock up to 40 MHz.

- Support the communication formats described in Section 24.5.1.
- Support single transfer and segmented-transfer. Have separated interrupts for both of them.

24.2.2.3 Functioning as a Master or a Slave

- Support 1-bit SPI. In full-duplex communication, 1-bit SPI is supported. In half-duplex communication, 3-line half-duplex SPI is supported, and uses only one data line SPI3_D; 4-line half-duplex SPI is also supported, and uses one of the data lines SPI3_D and SPI3_Q.
- Data transmission length configurable. 1 ~ 64 bytes for CPU controlled mode, 1 ~ n(unlimited) bytes for DMA controlled mode.
- Support configurable read and write data bit order: most-significant bit (MSB) first, or least-significant bit (LSB) first.
- Support full-duplex and half-duplex communications
- Support data exchange between external SPI devices and:
  - GP-SPI3 data buffer (access by CPU)
  - internal RAM (access by DMA)
- Support configurable clock frequency, polarity, and phase.
- Support SPI clock mode 0 ~ 3.

24.2.3 SPI Interrupt Features

- Support SPI interrupts, see Section 24.11.
- Support SPI DMA interrupts, see Section 24.11.

24.3 GP-SPI Interfaces

GP-SPI2 and GP-SPI3 are general purpose interfaces which can be configured as either a master or a slave to communicate with other SPI devices, see Figure 24-2. The supported data modes of GP-SPI2 and GP-SPI3 are shown in Table 131.

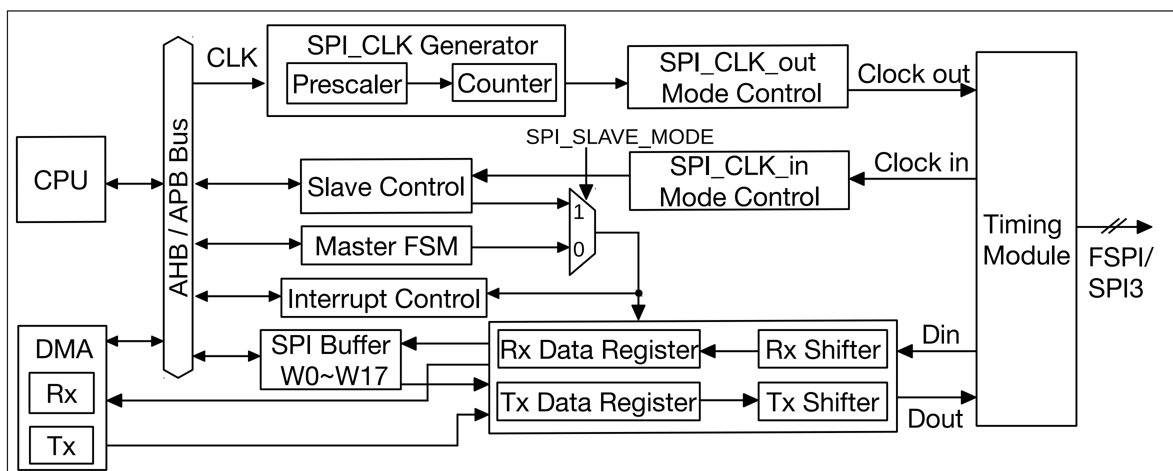


Figure 24-2. GP-SPI2/GP-SPI3 Block Diagram

Table 131: Data Modes Supported by GP-SPI2 and GP-SPI3

Supported Mode		CMD Phase	Address Phase	Data Phase	GP-SPI2	GP-SPI3
1-bit SPI		1-bit	1-bit	1-bit	Y	Y
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit	Y	_
	Dual I/O Read	1-bit	2-bit	2-bit	Y	_
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit	Y	_
	Quad I/O Read	1-bit	4-bit	4-bit	Y	_
Octal SPI	Octal Output Read	1-bit	1-bit	8-bit	Y	_
	Octal I/O Read	1-bit	8-bit	8-bit	Y	_
QPI		4-bit	4-bit	4-bit	Y	_
OPI		8-bit	8-bit	8-bit	Y	_

The functionalities of GP-SPI3 are nearly the same as those of GP-SPI2. GP-SPI2's functionalities are depicted in Section 24.4 and Section 24.5. The differences between GP-SPI2 and GP-SPI3 are described in Section 24.6. GP-SPI2 supports the following settings:

- Choose between full-duplex and half-duplex communications, depending on the bit SPI_DOUTDIN in register SPI_USER_REG:
  - 0: choose half-duplex.
  - 1: choose full-duplex.
- Choose between master and slave modes, depending on the bit SPI_SLAVE_MODE in register SPI_SLAVE_REG:
  - 0: choose master mode.
  - 1: choose slave mode.
- Set read/write data bit order by configuring SPI_RD_BIT_ORDER and SPI_WR_BIT_ORDER in register SPI_CTRL_REG, respectively:
  - 0: LSB first.
  - 1: MSB first.

When working as master, GP-SPI2 supports single transfer and DMA segmented-configure-transfer. Every single transfer needs to be triggered by ESP32-S2 CPU, after its related registers are configured. The segmented-configure-transfer consist of several single SPI transfers but requires only one triggering from ESP32-S2 CPU. The detailed description of segmented-configure-transfer can be seen in Subsection 24.4.7.

When working as slave, GP-SPI2 supports single transfer and segmented-transfer. GP-SPI2 should be configured to slave segmented-transfer mode before the SPI master starts the transfer in segments. Segmented-transfer will end when En_SEG_TRANS command is received correctly. The detailed description can be seen in Subsection 24.5.4.

When GP-SPI2 is working as slave and in the half-duplex mode, the SPI master should communicate with the slave by supported commands in Table 137 and Table 138 according to their format.

24.4 GP-SPI2 Works as a Master

Many features are supported in GP-SPI2 master mode. And all the features are controlled by GP-SPI2's state machine and registers. For more information, see the following subsections:

- Subsection 24.4.1: state machine workflow
- Subsection 24.4.2: register configuration rules
- Subsection 24.4.3 to Subsection 24.4.8: the functions of GP-SPI2 and its typical applications
- Subsection 24.4.9: programmable CS setup time and hold time
- Subsection 24.9: configuration of FSPICLK frequency, polarity, and phase

24.4.1 State Machine

GP-SPI2 is mainly used to access 1/2/4/8-bit SPI devices, such as flash, external RAM, and LCD, thus the naming of GP-SPI2 states keeps consistent with the sequence phase naming of flash, external RAM, and LCD. Figure 24-3 shows GP-SPI2 state flow, which is helpful to use all the functions of GP-SPI2 in master mode.

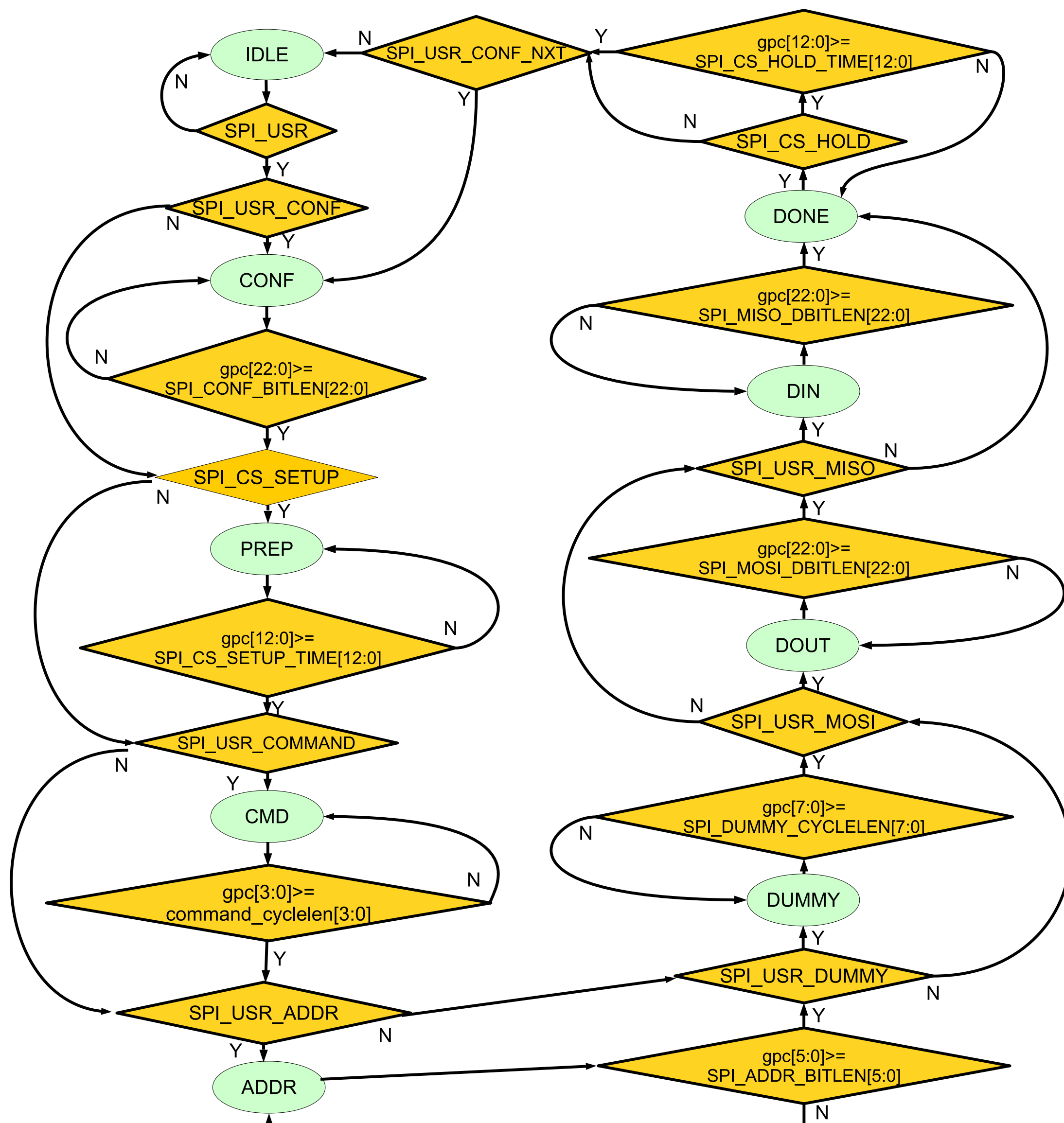


Figure 24-3. GP-SPI2 State Flow in Master Mode

As shown in Figure 24-3, GP-SPI2 has the following states:

1. IDLE: GP-SPI2 is not active or is in slave mode.
2. CONF: only used for segmented-configure-transfer function. Set the bits SPI_USR and SPI_USR_CONF to enable this state. If this state is not enabled, it means this transfer now is in single transfer mode.
3. PREP: prepare an SPI transmission and control SPI CS setup time. Set the bits SPI_USR and SPI_CS_SETUP to enable this state.
4. CMD: send command sequence. Set the bits SPI_USR and SPI_USR_COMMAND to enable this state.
5. ADDR: send address sequence. Set the bits SPI_USR and SPI_USR_ADDR to enable this state.
6. DUMMY (wait cycle): send dummy sequence. Set the bits SPI_USR and SPI_USR_DUMMY to enable this state.
7. DATA: transfer data.
   - DOUT: send data sequence. Set the bits SPI_USR and SPI_USR_MOSI to enable this state.
   - DIN: receive data sequence. Set the bits SPI_USR and SPI_USR_MISO to enable this state.
8. DONE: control SPI CS hold time. Set SPI_USR to enable this state.

As shown in Figure 24-3, a counter (gpc[22:0]) is used in state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently, of which the cycle lengths can also be configured independently. With the combination of the enabled states and their cycle lengths, the behavior of GP-SPI2's state machine can be controlled flexibly.

24.4.2 Register Configuration Rules for State Control

The behavior of GP-SPI2 in each state is controlled by GP-SPI2 registers. The register configuration rules, related to GP-SPI2 state control, are shown in Table 132 and Table 133. Users can enable OPI mode or QPI mode for GP-SPI2 by setting the bits SPI_OPI_MODE and SPI_QPI_MODE in register SPI_USER_REG, respectively. Note that the two bits can not be set at the same time.

Table 132: Register Configuration Rules for State Control in 1/2-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_USR_MISO_DBITLEN	SPI_USR_MISO SPI_USR_MISO_DBITLEN SPI_FREAD_DUAL

Table 132: Register Configuration Rules for State Control in 1/2-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus
DOUT	SPI_USR_MOSI SPI_USR_MOSI_DBITLEN	SPI_USR_MOSI SPI_USR_MOSI_DBITLEN SPI_FWRITE_DUAL

Table 133: Register Configuration Rules for State Control in 4/8-bit Modes

State	Control Registers for 4-bit Mode FSPI Bus	Control Registers for 8-bit Mode FSPI Bus
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_OCT SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_OCT
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_USR_MISO_DBITLEN SPI_FREAD_QUAD	SPI_USR_MISO SPI_USR_MISO_DBITLEN SPI_FREAD_OCT
DOUT	SPI_USR_MOSI SPI_USR_MOSI_DBITLEN SPI_FWRITE_QUAD	SPI_USR_MOSI SPI_USR_MOSI_DBITLEN SPI_FWRITE_OCT

As shown in Table 132 and Table 133, the registers in each cell should be configured to set FSPI bus to corresponding bit mode, i.e. the mode shown in table header, at a specific state phase (corresponding to the first column).

For instance, when GP-SPI2 reads data, and

- CMD is 4-bit mode
- ADDR is 2-bit mode
- DUMMY is n clock cycles
- DIN is 8-bit mode

The register configuration can be as follows:

1. Configure CMD state related registers.
    - Configure the required command value in SPI_USR_COMMAND_VALUE.
    - Configure command bit length in SPI_USR_COMMAND_BITLEN. SPI_USR_COMMAND_BITLEN = bit length expected - 1.

   - Set SPI_FCMD_QUAD and SPI_USR_COMMAND.
   - Clear SPI_FCMD_DUAL and SPI_FCMD_OCT.

2. Configure ADDR state related registers.
   - Configure the required address value in SPI_USR_ADDR_VALUE.
   - Configure address bit length in SPI_USR_ADDR_BITLEN. SPI_USR_ADDR_BITLEN = bit length expected - 1.
   - Set SPI_USR_ADDR and SPI_FADDR_DUAL.
   - Clear SPI_FADDR_QUAD and SPI_FADDR_OCT.
3. Configure DUMMY state related registers.
   - Configure DUMMY cycles in SPI_USR_DUMMY_CYCLELEN. SPI_USR_DUMMY_CYCLELEN = DUMMY cycles (*n*) expected - 1.
   - Set SPI_USR_DUMMY.
4. Configure DIN state related registers.
   - Configure read data bit length in SPI_USR_MISO_DBITLEN. SPI_USR_MISO_DBITLEN = bit length expected - 1.
   - Set SPI_FREAD_OCT and SPI_USR_MISO.
   - Clear SPI_FREAD_DUAL and SPI_FREAD_QUAD.
   - Configure GP-SPI2 DMA in DMA controlled mode. In CPU controlled mode, no action is needed.
5. Clear SPI_USR_MOSI.
6. Set SPI_USR to start GP-SPI2 transmission.

When write data (DOUT state), SPI_USR_MOSI and SPI_USR_MOSI_DBITLEN should be configured instead, while SPI_USR_MISO should be cleared. The output data bit length is the value of SPI_USR_MOSI_DBITLEN plus 1. Output data should be configured in GP-SPI2 data buffer (SPI_W0_REG ~ SPI_W17_REG) in CPU controlled mode, or DMA TX buffer in DMA controlled mode. The data byte order is incremented from LSB (byte 0) to MSB.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and to address value in SPI_USR_ADDR_VALUE.

The configuration of command value is as follows:

- If SPI_USR_COMMAND_BITLEN < 8, the command value is written to SPI_USR_COMMAND_VALUE[7:0]. Command value is sent as follows.
  - If SPI_WR_BIT_ORDER is cleared, the lower part of SPI_USR_COMMAND_VALUE[7:0] (SPI_USR_COMMAND_VALUE[SPI_USR_COMMAND_BITLEN:0]) is sent first.
  - If SPI_WR_BIT_ORDER is set, the higher part of SPI_USR_COMMAND_VALUE[7:0] (SPI_USR_COMMAND_VALUE[7: 7-SPI_USR_COMMAND_BITLEN]) is sent first.
- If 7 < SPI_USR_COMMAND_BITLEN < 16, the command value is written to SPI_USR_COMMAND_VALUE[15:0]. Command value is sent as follows.

  - If SPI_WR_BIT_ORDER is cleared, SPI_USR_COMMAND_VALUE[7:0] is sent first, and then the lower part of SPI_USR_COMMAND_VALUE[15:8] (SPI_USR_COMMAND_VALUE[SPI_USR_COMMAND_BITLEN:8]) is sent.
  - If SPI_WR_BIT_ORDER is set, SPI_USR_COMMAND_VALUE[7:0] is sent first, and then the higher part of SPI_USR_COMMAND_VALUE[15:8] (SPI_USR_COMMAND_VALUE[15: 15-SPI_USR_COMMAND_BITLEN]) is sent.

The configuration of address value is as follows:

- If SPI_USR_ADDR_BITLEN < 8, the address value is written to SPI_USR_ADDR_VALUE[7:0]. Address value is sent as follows.
  - If SPI_WR_BIT_ORDER is cleared, the lower part of SPI_USR_ADDR_VALUE[7:0] (SPI_USR_ADDR_VALUE[SPI_USR_ADDR_BITLEN:0]) is sent first.
  - If SPI_WR_BIT_ORDER is set, the higher part of SPI_USR_ADDR_VALUE[7:0] (SPI_USR_ADDR_VALUE[7: 7-SPI_USR_ADDR_BITLEN]) is sent first.
- If 7< SPI_USR_ADDR_BITLEN < 16, the ADDR value is written to SPI_USR_ADDR_VALUE[15:0]. Address value is sent as follows.
  - If SPI_WR_BIT_ORDER is cleared, SPI_USR_ADDR_VALUE[7:0] is sent first, and then the low part of SPI_USR_ADDR_VALUE[15:8] (SPI_USR_ADDR_VALUE[SPI_USR_ADDR_BITLEN:8]) is sent.
  - If SPI_WR_BIT_ORDER is set, SPI_USR_ADDR_VALUE[7:0] is sent first, and then the higher part of SPI_USR_ADDR_VALUE[15:8] (SPI_USR_ADDR_VALUE[15: 15-SPI_USR_ADDR_BITLEN]) is sent.

In summary, each state of GP-SPI2 can be configured to 1/2/4/8-bit mode independently. The register configuration for other applications can be found in Table 132, Table 133, and the instance above.

24.4.3 Full-Duplex Communication (1-bit Mode Only)

GP-SPI2 supports SPI full-duplex communication, one of the most widely used modes in electronic systems. In this mode, SPI master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode by MOSI (FSPID, sending) and MISO (FSPIQ, receiving) at the same time. Figure 24-4 shows the block diagram of full-duplex communication.

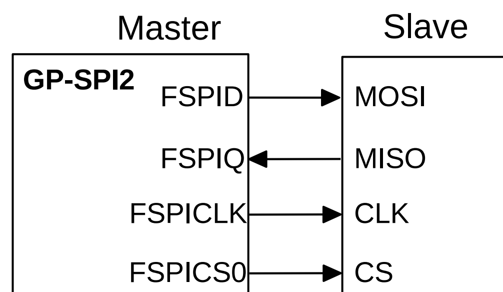


Figure 24-4. Full-Duplex Communication Between GP-SPI2 Master and a Slave

In full-duplex communication, the behavior in states CMD, ADDR, DUMMY, DOUT and DIN can be configured. Usually, the states of CMD, ADDR, DUMMY are not used in this communication. The bit length of transferred data is configured in SPI_USR_MOSI_DBITLEN and in SPI_USR_MISO_DBITLEN. The actual bit length used in communication = the value of SPI_USR_MOSI_DBITLEN or SPI_USR_MISO_DBITLEN + 1.

To start data transfer, follow the steps below:

- Set the bit SPI_DOUTDIN and clear the bit SPI_SLAVE_MODE, see Section 24.3.
- Set SPI_USR in register SPI_CMD_REG to start the transfer.

The read/write data byte order in CPU-controlled transfer can be configured in SPI_RD_BYTE_ORDER and SPI_WR_BYTE_ORDER in SPI_USER_REG. The register configuration for DMA-controlled transfer is described in Subsection 24.8.

24.4.4 Half-Duplex Communication (1/2/4/8-bit Mode)

SPI half-duplex mode is also very common in SPI communication. In this mode, SPI master provides CLK and CS signals. Only one side (SPI master or slave) can send data at a time, while the other side receives the data. The standard format of SPI half-duplex communication is CMD + [ADDR +] [DUMMY +] [DOUT or DIN]. The states ADDR, DUMMY, DOUT, and DIN are optional, and can be disabled or enabled independently.

As described in Subsection 24.4.2, the properties of GP-SPI2 states CMD, ADDR, DUMMY, DOUT and DIN, such as cycle length, value, and parallel bus bit mode, can be set independently. For the register configuration, see Table 132 and Table 133.

The detailed properties of half-duplex GP-SPI2 are as follows:

1. CMD: 0 ~ 16 bits, master output, slave input.
2. ADDR: 0 ~ 32 bits, master output, slave input.
3. DUMMY: 0 ~ 256 FSPICLK cycles, master output, slave input.
4. DOUT: 0 ~ 576 bits (72 bytes) in CPU controlled mode and 0 ~ 8 Mbits in DMA controlled mode, master output, slave input.
5. DIN: 0 ~ 576 bits (72 bytes) in CPU controlled mode and 0 ~ 8 Mbits in DMA controlled mode, master input, slave output.

The register configuration is as follows:

1. Configure GP-SPI2 registers as shown in Table 132 and Table 133.
2. Configure SPI CS setup time and hold time according to Subsection 24.4.9.
3. Set the property of FSPICLK according to Subsection 24.9.
4. Prepare data in registers SPI_W0_REG ~ SPI_W17_REG in CPU-controlled MOSI mode. Or configure DMA TX/RX link in DMA-controlled mode, as shown in Subsection 24.8.
5. Configure signal path of FSPI bus and interrupts.
6. Wait for SPI slave to get ready for transfer.
7. Set SPI_USR in register SPI_CMD_REG to start the transfer and wait the interrupt set in Step 5.

The maximum frequency supported by GP-SPI2 is f_{apb}. FSPICLK is divided from APB_CLK, see Section 24.9.

In addition, a timing module is provided for input and output data, to add extra delay to each I/O line in units of $T_{\text{apb}}/2$. The timing module configuration is described in Subsection 24.9.4. Setting DUMMY length and delay select function in GPIO Matrix, GP-SPI2 can meet the timing requirement in usage.

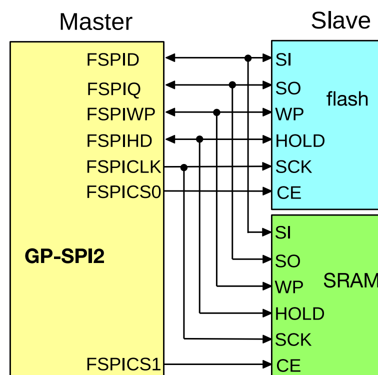


Figure 24-5. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

24.4.5 Access Flash and External RAM in Master Half-Duplex Mode

GP-SPI2 can be used to access 1/2/4-bit flash and external RAM, see Figure 24-5.

Figure 24-6 indicates GP-SPI2 quad read sequence according to standard flash specification. Other GP-SPI2 command sequences can be implemented in accordance with the requirements of SPI slaves.

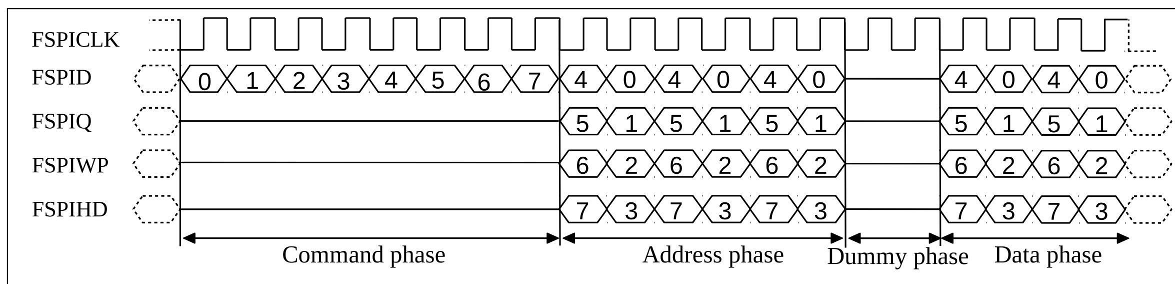


Figure 24-6. SPI Quad Read Command Sequence Sent by GP-SPI2 to Flash

The 1/2/4/8-bit SPI Double Transfer Rate (DTR) modes, in which data is sent and received at the positive and the negative edges of SPI clock, are also supported:

- If SPI_CMD_DTR_EN is set, the CMD value will be sent in DTR mode; otherwise CMD phase will be in Single Transfer Rate (STR) mode.
- If SPI_ADDR_DTR_EN is set, the ADDR value will be sent in DTR mode; otherwise ADDR phase will be in STR mode.
- If SPI_DATA_DTR_EN is set, the input data in state DIN or output data in state DOUT will be sent in DTR mode; otherwise the data will be in STR mode.

The control bits SPI_CMD_DTR_EN, SPI_ADDR_DTR_EN, and SPI_DATA_DTR_EN can be configured independently, which means CMD in STR mode, ADDR and DOUT or DIN in DTR mode are supported.

GP-SPI2 can only output FSPIDQS signal, but can not receive the signal. Therefore, only flash or external RAM working in fixed dummy mode (dummy cycles in a read sequence is fixed) are supported.

24.4.6 Access 8-bit I8080/MT6800 LCD in Master Half-Duplex Mode

The connection details of GP-SPI2 to an 8-bit LCD driver is shown in Figure 24-7.

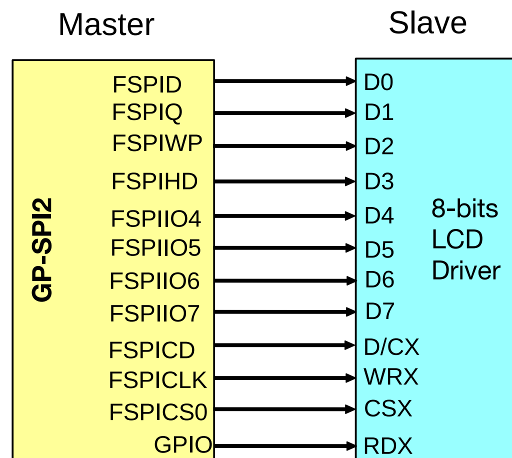


Figure 24-7. Connection of GP-SPI2 to 8-bit LCD Driver

Figure 24-8 shows the sequence of GP-SPI2 writing to I8080 interface using 8-bit LCD driver. The register configuration is as follows:

1. SPI_FCMD_OCT = 1, SPI_USR_COMMAND = 1, SPI_USR_COMMAND_VALUE[15:0] = 0x2C, SPI_USR_COMMAND_BITLEN = 0xF.
2. SPI_USR_ADDR = 0, SPI_USR_DUMMY = 0, SPI_USR_MISO = 0.
3. SPI_FWRITE_OCT =1, and SPI_USR_MOSI = 1, configure the output data bit length in SPI_USR_MOSI_DBITLEN. SPI_USR_MOSI_DBITLEN = data bit length expected - 1.
4. Configure the SPI CS setup time and hold time according to Subsection 24.4.9.
5. Prepare the output data in registers SPI_W0_REG ~ SPI_W17_REG in CPU-controlled MOSI mode. Or configure DMA TX link in DMA-controlled mode.
6. Set FSPICLK to SPI clock mode 3 according to Subsection 24.9. Configure the required frequency of FSPICLK.
7. Configure signal path of FSPI bus and interrupts.
8. Wait for the LCD driver to get ready for transfer.
9. Set SPI_USR in register SPI_CMD_REG to start the transfer and wait the interrupt set in Step 7.

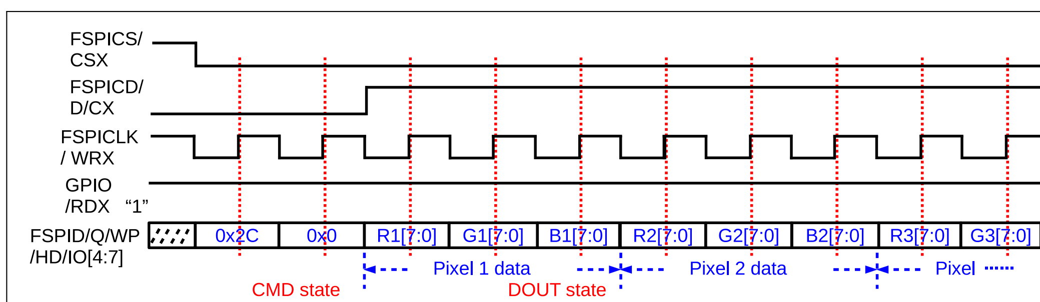


Figure 24-8. Write Command Sequence to an 8-bit LCD Driver

In I8080 LCD read operation, RDX pin is mapped to FSPICLK and WRX pin is set to high by a GPIO pin. For the application of Moto6800 interface LCD mode, please refer to the I8080 mode example above.

24.4.7 DMA Controlled Segmented-Configure-Transfer

An SPI transfer, controlled by the registers of GP-SPI2, needs to be configured and triggered by the CPU. To reduce CPU cost and to increase the efficiency of GP-SPI2, DMA segmented-configure-transfer function is provided in DMA-controlled master mode. (Note that CPU-controlled master mode does not support segmented-configure-transfer function.) Segmented-configure-transfer enables GP-SPI2 to do as many times of transfer as configured, with only one triggering by CPU. Figure 24-9 shows how the segmented-configure-transfer function works in DMA-controlled master mode.

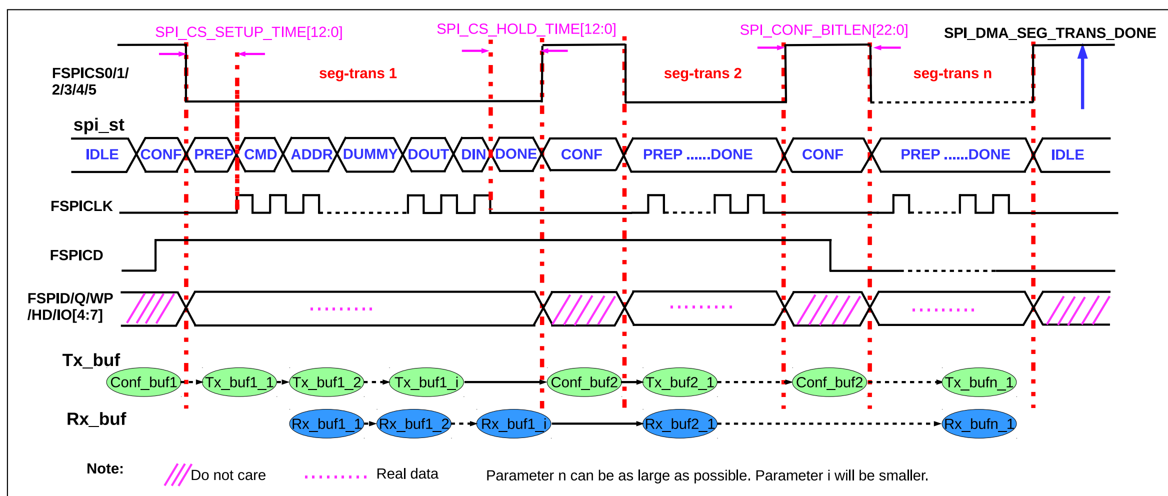


Figure 24-9. Segmented-Configure-Transfer in DMA Controlled Master Mode

As shown in Figure 24-9, GP-SPI2 registers can be reconfigured by GP-SPI2 hardware according to the content in a Conf_buf*i* during CONF state, before one single segmented-configure-transfer, SCT*i*, begins.

It is recommended that each SCT CONF state has its own DMA CONF link and CONF buffer. A serial DMA TX link is used to connect all the CONF buffer and TX data buffer into a chain, so that the behavior of FSPI bus in each SCT can be controlled independently and can be any supported one.

Meanwhile, the state of GP-SPI2, the related value and cycle length of the FSPI bus, and the behavior of DMA, can be configured independently for each SCT. When all the SCT are finished, a GP-SPI2 interrupt, SPI_DMA_SEG_TRANS_DONE, is triggered.

For example, as shown in Figure 24-9, SCT*i* can be configured to DMA-controlled full-duplex communication; SCT*j* can be configured to DMA-controlled half-duplex MISO mode; and SCT*k* can be configured to DMA-controlled half-duplex MOSI mode. *i*, *j*, and *k* are integer variables, which can be any SCT number.

In addition, GP-SPI2 CS setup time and hold time are programmable independently in each SCT, see Subsection 24.4.9 for detailed configuration. The CS high time in CONF state is determined by the sum of SPI_CONF_BITLEN[22:0] in SPI_CMD_REG and SPI_CONF_BASE_BITLEN[6:0] in SPI_SLV_WRBUF_DLEN_REG. The CS high time in CONF state can be set from 2 μs to 0.2 s when f_{apb} is 80 MHz and FSPICLK equals to APB_CLK (divided by 1). If the FSPICLK is slower (divided by n > 1), the CS high time can be even longer.

Here's an example of register configuration flow in segmented-configure-transfer mode:

1. Prepare descriptors for DMA CONF buffer and TX data (optional) for each segmented-configure-transfer. Chain the descriptors of CONF buffer and TX buffers of several transfers into one linked list.
2. Similarly, prepare descriptors for RX buffers for each segmented-configure-transfer and chain them into a

linked list.

3. Configure the values in all the CONF buffers, TX buffers and RX buffers, respectively for each transfers of segmented-configure-transfer before the segmented-configure-transfer begins.
4. Point SPI_OUTLINK_ADDR to the address of the head of the CONF and TX buffer descriptor linked list, and then set SPI_OUTLINK_START bit in the register SPI_DMA_OUT_LINK_REG to start the TX DMA.
5. Clear SPI_RX_EOF_EN bit in SPI_DMA_CONF_REG register. Point SPI_INLINK_ADDR to the address of the head of the RX buffer descriptor linked list, and then set SPI_INLINK_START bit in SPI_DMA_IN_LINK_REG to start the RX DMA.
6. Set SPI_USR_CONF in SPI_SLV_RD_BYTE_REG to enable CONF state.
7. Set SPI_INT_DMA_SEG_TRANS_EN in SPI_SLAVE_REG to enable the SPI_DMA_SEG_TRANS_DONE interrupt. Configure other interrupts if needed according to Subsection 24.11.
8. Wait for all the segmented-configure-transfer slaves to get ready for transfer.
9. Set SPI_USR in SPI_CMD_REG to start the DMA master mode segmented-configure-transfer.
10. Wait for SPI_DMA_SEG_TRANS_DONE interrupt, which means the DMA segmented-configure-transfer is ended and the data has been stored into corresponding memory.

Only changed registers of GP-SPI2 (compared to last transmission) need to be configured to new values in CONF state. The configuration of other registers can be skipped (and keep the same) to save time and chip resources.

The first word in DMA CONF buffer*i*, called SPI_BIT_MAP_WORD, defines whether each SPI register is updated or not in segmented-configure-transfer*i*. The relation of SPI_BIT_MAP_WORD and GP-SPI2 registers to update can be seen in Table 134 Bitmap (BM) Table. If a bit is 1 in the BM table, the register value of the corresponding bit will be updated in the single transfer. Otherwise, if some registers should be kept from being changed, the related bits should be set to 0.

Table 134: GP-SPI Master BM Table for CONF State

BM Bit	Register Name	BM Bit	Register Name
0	SPI_CMD	14	SPI_HOLD
1	SPI_ADDR	15	SPI_DMA_INT_ENA
2	SPI_CTRL	16	SPI_DMA_INT_RAW
3	SPI_CTRL1	17	SPI_DMA_INT_CLR
4	SPI_CTRL2	18	SPI_DIN_MODE
5	SPI_CLOCK	19	SPI_DIN_NUM
6	SPI_USER	20	SPI_DOUT_MODE
7	SPI_USER1	21	SPI_DOUT_NUM
8	SPI_USER2	22	SPI_LCD_CTRL
9	SPI_MOSI_DLEN	23	SPI_LCD_CTRL1
10	SPI_MISO_DLEN	24	SPI_LCD_CTRL2
11	SPI_MISC	25	SPI_LCD_D_MODE
12	SPI_SLAVE	26	SPI_LCD_D_NUM
13	SPI_FSM	-	-

Then new values of all the registers to modify should be placed right after the SPI_BIT_MAP_WORD, in consecutive words in the CONF buffer.

To ensure the correctness of the content in each CONF buffer, the value in SPI_BIT_MAP_WORD[31:28] is used as "magic value", and will be compared with SPI_DMA_SEG_MAGIC_VALUE[3:0] in the register SPI_SLV_RD_BYTE_REG. The value of SPI_DMA_SEG_MAGIC_VALUE[3:0] should be configured before segmented-configure-transfer starts, and can not be changed in the segmented-configure-transfers.

- If SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE[3:0], the segmented-configure-transfer continues normally; the interrupt SPI_DMA_SEG_TRANS_DONE will be triggered at the end of the segmented-configure-transfer.
- If SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE[3:0], GP-SPI2 state (spi_st) will go back to IDLE and the segmented-configure-transfer will be ended immediately. The interrupt SPI_DMA_SEG_TRANS_DONE will still be triggered, with SPI_SEG_MAGIC_ERR bit in SPI_SLV_RDBUF_DLEN_REG register set to 1.

Table 135 and Table 136 provide an example to show how to configure a CONF buffer for a transfer whose SPI_ADDR_REG, SPI_CTRL_REG, SPI_CLOCK_REG, SPI_USER_REG, SPI_USER1_REG need to be updated.

Table 135: An Example of CONF buffer$_i$ in Segmented-Configure-Transfer

CONF buffer$_i$	Note
SPI_BIT_MAP_WORD	The first word in this buffer, 0x000000E6 in this example. As shown in Table 136, bits 1, 2, 5, 6, and 7 are set, indicating the following registers will be updated.
SPI_ADDR_REG	The second word, stores the new value to SPI_ADDR_REG
SPI_CTRL_REG	The third word, stores the new value to SPI_CTRL_REG
SPI_CLOCK_REG	The fourth word, stores the new value to SPI_CLOCK_REG
SPI_USER_REG	The fifth word, stores the new value to SPI_USER_REG
SPI_USER1_REG	The sixth word, stores the new value to SPI_USER1_REG

Table 136: BM Bit Value v.s. Register to Be Updated in the Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
0	0	SPI_CMD_REG	14	0	SPI_HOLD_REG
1	1	SPI_ADDR_REG	15	0	SPI_DMA_INT_ENA_REG
2	1	SPI_CTRL_REG	16	0	SPI_DMA_INT_RAW_REG
3	0	SPI_CTRL1_REG	17	0	SPI_DMA_INT_CLR_REG
4	0	SPI_CTRL2_REG	18	0	SPI_DIN_MODE_REG
5	1	SPI_CLOCK_REG	19	0	SPI_DIN_NUM_REG
6	1	SPI_USER_REG	20	0	SPI_DOUT_MODE_REG
7	1	SPI_USER1_REG	21	0	SPI_DOUT_NUM_REG
8	0	SPI_USER2_REG	22	0	SPI_LCD_CTRL_REG
9	0	SPI_MOSI_DLEN_REG	23	0	SPI_LCD_CTRL1_REG
10	0	SPI_MISO_DLEN_REG	24	0	SPI_LCD_CTRL2_REG
11	0	SPI_MISC_REG	25	0	SPI_LCD_D_MODE_REG

Table 136: BM Bit Value v.s. Register to Be Updated in the Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
12	0	SPI_SLAVE_REG	26	0	SPI_LCD_D_NUM_REG
13	0	SPI_FSM_REG	-	-	-

When using DMA segmented-configure-transfer, please pay special attention to the following bits:

- SPI_USR_CONF in register SPI_SLV_RD_BYTE_REG
- SPI_USR_CONF_NXT in register SPI_USER_REG
- SPI_CONF_BITLEN[22:0] in register SPI_CMD_REG

Set bit SPI_USR_CONF before SPI_USR is set to enable segmented-configure-transfer function. If the segmented-configure-transfer corresponding to the CONF buffer is not the last transfer, the value of SPI_USR_CONF_NXT should be set to 1; otherwise the SPI_USR_CONF_NXT will stop when the transfer ends.

24.4.8 Access Parallel 8-bit RGB Mode LCD via Segmented-Configure-Transfer

Segmented-configure-transfer is very convenient and powerful for huge numbers of data transfer. One possible application is to communicate with a parallel 8-bit RGB mode LCD. You can refer to the steps in this example for other applications.

Figure 24-10 shows the video frame structure of a parallel 8-bit RGB mode LCD.

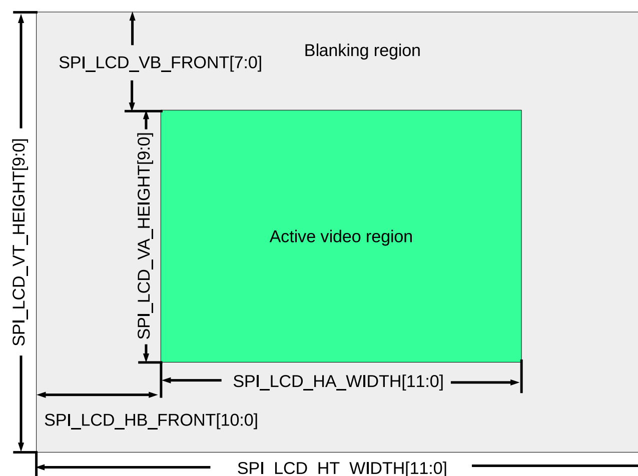


Figure 24-10. Video Frame Structure in Parallel RGB 8-bit LCD Mode

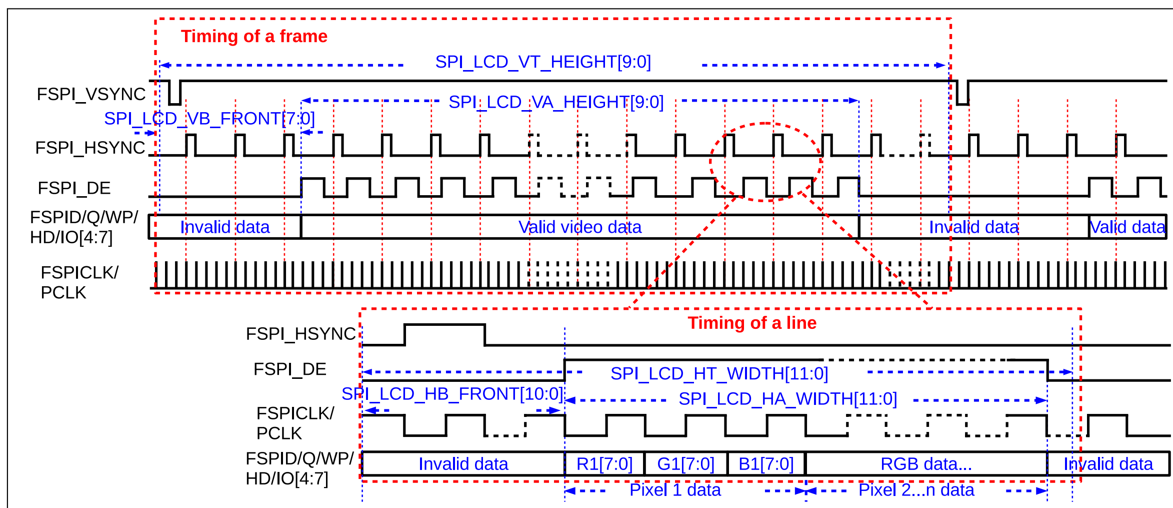


Figure 24-11. Timing Sequence in Parallel RGB 8-bit LCD Mode

It can be seen from Figure 24-9, Figure 24-10, and Figure 24-11 that a good way for GP-SPI2 to output parallel 8-bit RGB frame data is as follows:

- Data is output in each frame line in a segmented-configure-transfer. One valid video line is transferred in each single transfer (i.e. SCT*i* in Figure 24-9) of segmented-configure-transfer. There are SPI_LCD_VA_HEIGHT[9:0] active video region lines in a frame, so that a frame data is sent in SPI_LCD_VA_HEIGHT[9:0] times of SCT*i*.
- There is no TX buffer and just one CONF buffer in the SCT*i* corresponding to vertical blanking region. GP-SPI2 sends frame format signals in FSPI_VSYNC, FSPI_HSYNC, FSPI_DE and/or FSPICLK if needed.
- Horizontal blanking period is equal to the sum of CS high time in a CONF state, CS setup time and CS hold time, see Subsection 24.4.7 and Subsection 24.4.9.

GP-SPI2 can output the frame data automatically as shown in Figure 24-10 and Figure 24-11, which significantly lowers the CPU time required to control the SPI transfer. The configuration flow is as follows:

1. Set SPI_LCD_MODE_EN in register SPI_LCD_CTRL_REG to enable parallel RGB LCD mode.
2. Configure frame control registers, as shown in Figure 24-10 and Figure 24-11, in accordance with the application. For example, in the application, if *n* active video lines are needed, then set SPI_LCD_VA_HEIGHT[9:0] to *n*.
3. Configure CS setup time and hold time if needed.
4. Clear SPI_USR_COMMAND, SPI_USR_ADDR, SPI_USR_DUMMY, and SPI_USR_MISO bits.
5. Set SPI_FWRITE_OCT and SPI_USR_MOSI bits, configure SPI_USR_MOSI_DBITLEN to SPI_LCD_HA_WIDTH[11:0] * 8 -1.
6. Configure DMA TX link and prepare data in TX buffer. A ring-buffer or a ping-pong buffer is recommended in this application to reduce the time preparing the buffer before a transfer starts.
7. Set I/O path. Set SPI_INT_TRANS_DONE_EN bit or other related interrupt enable bits.
8. Wait for the LCD slave to get ready for transfer.
9. Set SPI_USR to start the transfer. Wait for the SPI_TRANS_DONE interrupt.

24.4.9 CS Setup Time and Hold Time Control

SPI CS setup time and hold time are very important to meet the timing requirements of various types of SPI devices (e.g. flash or PSRAM). CS setup time is the time between the CS active edge and the first latch edge of SPI CLK (rising edge for mode 0 and 3 while falling edge for mode 2 and 4). CS hold time is the time between the last latch edge of SPI CLK and the CS inactive edge.

Set the CS setup time by specifying SPI_CS_SETUP in SPI_USER_REG and SPI_CS_SETUP_TIME in SPI_CTRL2_REG:

- If SPI_CS_SETUP = 0, the SPI CS setup time is 0.5 x T_SPI_CLK. T_SPI_CLK: one cycle of SPI_CLK.
- If SPI_CS_SETUP = 1, the SPI CS setup time is (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK.

Set the CS hold time by specifying SPI_CS_HOLD in SPI_USER_REG and SPI_CS_HOLD_TIME in SPI_CTRL2_REG:

- If SPI_CS_HOLD = 0, the SPI CS hold time is 0.5 x T_SPI_CLK;
- If SPI_CS_HOLD = 1, the SPI CS hold time is (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK.

Figure 24-12 and Figure 24-13 show the recommended CS timing and configuration register value for Flash and External RAM.

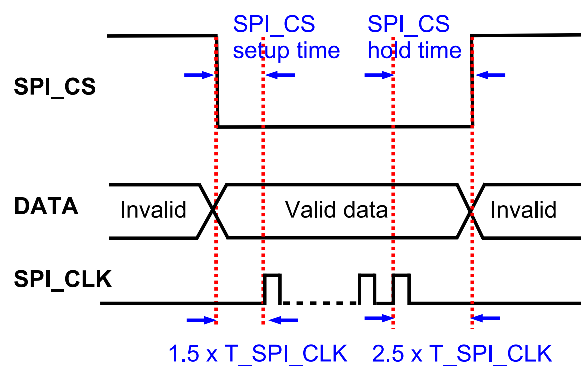
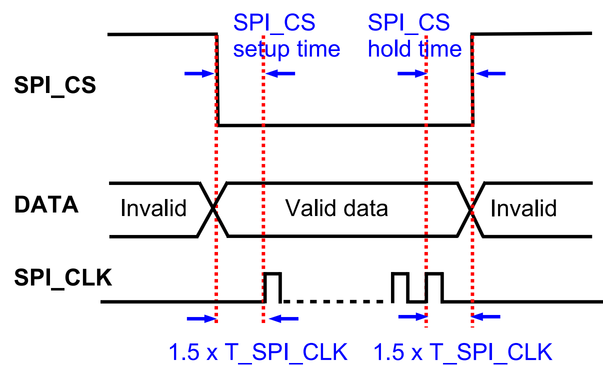


Figure 24-12. Recommended CS Timing and Settings When Access External RAM



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

Figure 24-13. Recommended CS Timing and Settings When Access Flash

GP-SPI2 can be configured as a master for many applications. The previous sections have shown how to configure the peripheral. See Section 24.13 for more information and detailed register descriptions.

24.5 GP-SPI2 Works as a Slave

GP-SPI2 can be used as a slave to communicate with an SPI master. As a slave, GP-SPI2 supports 1-bit SPI, 2-bit dual SPI, 4-bit quad SPI, and QPI mode, with specific communication formats. To enable slave mode, the bit SPI_SLAVE_MODE in register SPI_SLAVE_REG should be set to 1.

The CS signal should be held low during the transmission, and its falling/rising edges indicate the start/end of a transmission. The active length of transferred data should be in unit of bytes, otherwise the extra bits will be lost.

24.5.1 Communication Formats

SPI full-duplex and half-duplex communications are available in GP-SPI2 slave mode, depending on the bit SPI_DOUTDIN in register SPI_USER_REG:

- 1: enable full-duplex communication
- 0: enable half-duplex communication

In full-duplex communication, input and output data are transmitted simultaneously from the beginning of the transmission. All bits are treated as input/output data, which means no command, address or dummy phases are expected. The interrupt SPI_TRANS_DONE will be triggered when the transmission ends.

In half-duplex communication, the format should be CMD+ADDR+DUMMY+DATA (Read or Write). The read data phase means that an SPI master reads data from GP-SPI2, while the write data phase means that an SPI master writes data to GP-SPI2, which is defined from the perspective of the master. The detailed properties of each phase are as follows:

1. CMD: used to distinguish from different functions of SPI Slave; 1 byte from master to slave; only the values in Table 137 and Table 138 are valid; can be sent in 1-bit mode or 4-bit QPI mode.
2. ADDR: used to address data in CPU-controlled mode for CMD1 and 2, or placeholder bits in other modes;

1 byte from master to slave; can be sent 1-bit, 2-bit or 4-bit mode (according to the command).

3. DUMMY: value not meaningful, SPI Slave prepare data in this phase; 1 or 2 bytes (according to the command) from master to slave.

4. Read or write data: data length can be 0 ~ 72 bytes in CPU-controlled mode and 0 ~ 2 M-byte in DMA-controlled mode; can be 1-bit, 2-bit or 4-bit mode according to the CMD value.

The phases of ADDR and DUMMY can never be omitted in half-duplex communications, even in modes like the DMA-controlled slave mode.

When a half-duplex transmission is ended, the transferred CMD and ADDR values will be latched in SPI_SLV_LAST_COMMAND and SPI_SLV_LAST_ADDR bits in SPI_SLAVE1_REG register respectively. The SPI_SLV_CMD_ERR bit in SPI_SLAVE1_REG register will be set if the transferred CMD value is not supported by slave mode of GP-SPI2. The SPI_SLV_CMD_ERR bit can only be cleared by software.

The CPU-controlled and DMA-controlled mode configuration can be found in Subsection 24.8 and Subsection 24.7.

24.5.2 Supported CMD Values in Half-Duplex Communication

In half-duplex communication, the defined values of CMD determine the transfer types. Unsupported CMD values will be disregarded, meanwhile the related transfer will be ignored and the SPI_SLV_CMD_ERR bit will be set to 1. The transfer format is CMD (8 bits) + ADDR (8 bits) + DUMMY (8 SPI cycles in 1-bit SPI mode and 4 SPI cycles in other SPI modes) + DATA (unit in bytes). The detailed description of CMD[3:0] is as follows:

1. 0x1 (Wr_BUF): CPU-controlled write mode. Master sends data and GP-SPI2 receives data. The data will be stored in the related address of the registers SPI_W0_REG ~ SPI_W17_REG.

2. 0x2 (Rd_BUF): CPU-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from the related address of SPI_W0_REG ~ SPI_W17_REG.

3. 0x3 (Wr_DMA): DMA-controlled write mode. Master sends data and GP-SPI2 receives data. The data will be stored in GP-SPI2 DMA RX buffer.

4. 0x4 (Rd_DMA): DMA-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from GP-SPI2 DMA TX buffer.

5. 0x7 (CMD7): used to generate an SPI_SLV_CMD7_INT interrupt. It can also generate an SPI_IN_SUC_EOF_INT interrupt in a slave segmented-transfer when DMA RX link is used. But it will not end GP-SPI2's slave mode segmented-transfer.

6. 0x8 (CMD8): only used to generate an SPI_SLV_CMD8_INT interrupt, which will not end GP-SPI2's slave mode segmented-transfer.

7. 0x9 (CMD9): only used to generate an SPI_SLV_CMD9_INT interrupt, which will not end GP-SPI2's slave mode segmented-transfer.

8. 0xA (CMDA): only used to generate an SPI_SLV_CMDA_INT interrupt, which will not end GP-SPI2's slave mode segmented-transfer.

1/2/4-bit modes in phases of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY phase is always 1-bit mode and lasts for 8 SPI cycles in 1-bit SPI mode and 4 SPI cycles in other SPI modes. The definition is as follows:

1. 0x0: all the phases of CMD, ADDR, and DATA are in 1-bit mode.

2. 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
3. 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
4. 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
5. 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode. Or in QPI mode. Whereas, the CMD values of 0xA7, 0xA8, 0xA9 and 0xAA are only valid in QPI mode.

In addition, for CMD[7:0] values of 0x05, 0xA5, 0x06, and 0xDD, the phases of ADDR, DUMMY, and DATA are deleted. The definition is as follows:

1. 0x05 (End_SEG_TRANS): master sends 0x05 command to end the segmented-transfer in SPI mode.
2. 0xA5 (End_SEG_TRANS): master sends 0xA5 command to end the segmented-transfer in QPI mode.
3. 0x06 (En_QPI): GP-SPI2 enters QPI mode when receives the 0x06 command and the SPI_QPI_MODE bit in SPI_USER_REG register will be set.
4. 0xDD (Ex_QPI): GP-SPI2 exits QPI mode when receives the 0xDD command and the SPI_QPI_MODE bit will be cleared.

All the GP-SPI2 supported CMD values are given in Table 137 and Table 138.

Table 137: Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD Phase	ADDR Phase	DUMMY Phase	DATA Phase
Wr_BUF	0x01	1-bit mode	1-bit mode	8 cycles	1-bit mode
	0x11	1-bit mode	1-bit mode	4 cycles	2-bit mode
	0x21	1-bit mode	1-bit mode	4 cycles	4-bit mode
	0x51	1-bit mode	2-bit mode	4 cycles	2-bit mode
	0xA1	1-bit mode	4-bit mode	4 cycles	4-bit mode
Rd_BUF	0x02	1-bit mode	1-bit mode	8 cycles	1-bit mode
	0x12	1-bit mode	1-bit mode	4 cycles	2-bit mode
	0x22	1-bit mode	1-bit mode	4 cycles	4-bit mode
	0x52	1-bit mode	2-bit mode	4 cycles	2-bit mode
	0xA2	1-bit mode	4-bit mode	4 cycles	4-bit mode
Wr_DMA	0x03	1-bit mode	1-bit mode	8 cycles	1-bit mode
	0x13	1-bit mode	1-bit mode	4 cycles	2-bit mode
	0x23	1-bit mode	1-bit mode	4 cycles	4-bit mode
	0x53	1-bit mode	2-bit mode	4 cycles	2-bit mode
	0xA3	1-bit mode	4-bit mode	4 cycles	4-bit mode
Rd_DMA	0x04	1-bit mode	1-bit mode	8 cycles	1-bit mode
	0x14	1-bit mode	1-bit mode	4 cycles	2-bit mode
	0x24	1-bit mode	1-bit mode	4 cycles	4-bit mode
	0x54	1-bit mode	2-bit mode	4 cycles	2-bit mode
	0xA4	1-bit mode	4-bit mode	4 cycles	4-bit mode
CMD7	0x07	1-bit mode	-	-	-
	0x17	1-bit mode	-	-	-
	0x27	1-bit mode	-	-	-
	0x57	1-bit mode	-	-	-

Table 137: Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD Phase	ADDR Phase	DUMMY Phase	DATA Phase
CMD8	0x08	1-bit mode	-	-	-
	0x18	1-bit mode	-	-	-
	0x28	1-bit mode	-	-	-
	0x58	1-bit mode	-	-	-
CMD9	0x09	1-bit mode	-	-	-
	0x19	1-bit mode	-	-	-
	0x29	1-bit mode	-	-	-
	0x59	1-bit mode	-	-	-
CMDA	0x0A	1-bit mode	-	-	-
	0x1A	1-bit mode	-	-	-
	0x2A	1-bit mode	-	-	-
	0x5A	1-bit mode	-	-	-
End_SEG_TRANS	0x05	1-bit mode	-	-	-
En_QPI	0x06	1-bit mode	-	-	-

Table 138: Supported CMD Values in QPI Mode

Transfer Type	CMD[7:0]	CMD Phase	ADDR Phase	DUMMY Phase	DATA Phase
Wr_BUF	0xA1	4-bit mode	4-bit mode	4 cycles	4-bit mode
Rd_BUF	0xA2	4-bit mode	4-bit mode	4 cycles	4-bit mode
Wr_DMA	0xA3	4-bit mode	4-bit mode	4 cycles	4-bit mode
Rd_DMA	0xA4	4-bit mode	4-bit mode	4 cycles	4-bit mode
CMD7	0xA7	4-bit mode	-	-	-
CMD8	0xA8	4-bit mode	-	-	-
CMD9	0xA9	4-bit mode	-	-	-
CMDA	0xAA	4-bit mode	-	-	-
End_SEG_TRANS	0xA5	4-bit mode	-	-	-
Ex_QPI	0xDD	4-bit mode	-	-	-

Master should send 0x06 CMD (En_QPI) to set GP-SPI2 slave to QPI mode and all the phases of supported transfer will be in 4-bit mode afterwards. If 0xDD CMD (Ex_QPI) is received, GP-SPI2 slave will be back to SPI mode.

The other transfer types will be ignored. If the transferred bit length is smaller than 8, there will be no effect on GP-SPI2 as if it has not happened. However, if the bit length is larger than 8, SPI_TRANS_DONE will be triggered. For more information on interrupts triggered at the end of transmission, please refer to Subsection 24.11.

24.5.3 GP-SPI2 Slave Mode Single Transfer

In single transfer, CPU/DMA-controlled full-duplex and half-duplex communications are supported in GP-SPI2 slave mode. The register configuration flow is as follows:

1. Set the bits SPI_DOUTDIN and SPI_SLAVE_MODE as shown in Section 24.3.

2. Prepare data in registers SPI_W0_REG ~ SPI_W17_REG in CPU-controlled mode, if needed.
3. If in DMA-controlled mode, configure DMA descriptors and start DMA, as shown in Subsection 24.8.
4. Clear SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable single transfer mode.
5. Set SPI_INT_TRANS_DONE_EN in SPI_SLAVE_REG and wait the interrupt SPI_TRANS_DONE. In DMA-controlled mode, waiting for the interrupt SPI_IN_SUC_EOF_INT is better when DMA RX buffer is used, which means that data has been stored in the related memory.

24.5.4 GP-SPI2 Slave Mode Segmented-Transfer

To enhance the communication efficiency and reliability, segmented-transfer function is supported in GP-SPI2 slave mode. Since the registers SPI_W0_REG ~ SPI_W17_REG must be addressable in CPU-controlled slave half-duplex mode, the CPU-controlled segmented-transfer is not supported. Only DMA-controlled full-duplex and half-duplex segmented-transfer are available in GP-SPI2 slave mode.

In GP-SPI2 slave segmented-transfer mode, master supports all transfer types given in Table 137 and Table 138, in a segmented-transfer. It means that CPU-controlled transfer and DMA-controlled transfer can be mixed in one segmented-transfer.

It is recommended that in a segmented-transfer:

- CPU-controlled transfer is used in handshake communication and small-size data transfer, with related interrupts shown in Subsection 24.11.
- DMA-controlled transfer is used for huge numbers of data transfer with related interrupts in Subsection 24.11.

When End_SEG_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI2, the segmented-transfer will be ended and the interrupt SPI_DMA_SEG_TRANS_DONE will be triggered. The register configuration flow is as follows:

1. Set the bit SPI_SLAVE_MODE in SPI_SLAVE_REG.
2. Prepare data in registers SPI_W0_REG ~ SPI_W17_REG in CPU-controlled mode, if needed.
3. Configure DMA descriptors and start DMA. Clear SPI_RX_EOF_EN bit in SPI_DMA_CONF_REG.
4. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented-transfer mode.
5. Set SPI_INT_DMA_SEG_TRANS_EN in SPI_SLAVE_REG and wait for the interrupt SPI_DMA_SEG_TRANS_DONE, which means that segmented-transfer ends and data has been put into the related memory. Other interrupts described in Subsection 24.11 can also be used in this application.

In DMA full-duplex segmented-transfer, the data should be transferred from and to the DMA buffer. CPU-controlled mode is not supported in this segmented-transfer. The interrupt SPI_IN_SUC_EOF_INT_ST will be triggered when the transfer ends. The configuration flow is as follows:

1. Set the bit SPI_DOUTDIN in register SPI_USER_REG and the bit SPI_SLAVE_MODE in register SPI_SLAVE_REG.
2. Configure DMA descriptors and start DMA.
3. Set the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. The bits SPI_SLV_DMA_RD_BYTELEN[19:0] in register SPI_SLV_RDBUF_DLEN_REG should be configured to the receive data byte lengths of DMA.

4. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable segmented-transfer mode.
5. Set SPI_IN_SUC_EOF_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_IN_SUC_EOF_INT_ST.

24.6 Differences Between GP-SPI2 and GP-SPI3

The feature differences between GP-SPI2 and GP-SPI3 are as follows:

- The communication mode for each GP-SPI2 state (CMD, ADDR, DOUT or DIN) can be configured independently. Data can either be in 1/2/4/8-bit master mode or 1/2/4-bit slave mode. Whereas GP-SPI3 supports only 1-bit communication mode which does not allow for such flexibility.
- The supported CMD values of GP-SPI2 slave mode are shown in Table 137 and Table 138. While only 0x01, 0x02, 0x03, 0x04, 0x05, 0x07, 0x08, 0x09 and 0x0A of the CMD values are supported in GP-SPI3 slave mode.
- The I/O lines of GP-SPI2 can be mapped to physical GPIO pads either via GPIO Matrix or IO MUX. However, GP-SPI3 lines can be configured only via GPIO Matrix.
- GP-SPI2 has six CS signals in master mode. GP-SPI3 only has three CS signals in master mode.

Apart from that, the functions of GP-SPI2 and GP-SPI3 are the same. GP-SPI2 can use all the GP-SPI registers, while GP-SPI3 can only use some of the GP-SPI registers, see Table 139 for details.

Table 139: Invalid Registers and Fields for GP-SPI3

Invalid Register	Invalid Field
SPI_USER_REG	SPI_QPI_MODE SPI_OPI_MODE SPI_FWRITE_DUAL SPI_FWRITE_QUAD SPI_FWRITE_OCT
SPI_CTRL_REG	SPI_FADDR_DUAL SPI_FADDR_QUAD SPI_FADDR_OCT SPI_FCMD_DUAL SPI_FCMD_QUAD SPI_FCMD_OCT SPI_FREAD_DUAL SPI_FREAD_QUAD SPI_FREAD_OCT
SPI_MISC_REG	SPI_CS3_DIS SPI_CS4_DIS SPI_CS5_DIS SPI_MASTER_CS_POL[5:3] SPI_QUAD_DIN_PIN_SWAP
SPI_SLAVE1_REG	SPI_SLV_NO_QPI_EN
SPI_DMA_INT_ENA_REG	SPI_SLV_CMD6_INT_ENA
SPI_DMA_INT_RAW_REG	SPI_SLV_CMD6_INT_RAW
SPI_DMA_INT_ST_REG	SPI_SLV_CMD6_INT_ST
SPI_DMA_INT_CLR_REG	SPI_SLV_CMD6_INT_CLR

Table 139: Invalid Registers and Fields for GP-SPI3

Invalid Register	Invalid Field
SPI_DIN_MODE_REG	SPI_DIN2_MODE SPI_DIN3_MODE SPI_DIN4_MODE SPI_DIN5_MODE SPI_DIN6_MODE SPI_DIN7_MODE
SPI_DIN_NUM_REG	SPI_DIN2_NUM SPI_DIN3_NUM SPI_DIN4_NUM SPI_DIN5_NUM SPI_DIN6_NUM SPI_DIN7_NUM
SPI_DOUT_MODE_REG	SPI_DOUT2_MODE SPI_DOUT3_MODE SPI_DOUT4_MODE SPI_DOUT5_MODE SPI_DOUT6_MODE SPI_DOUT7_MODE
SPI_DOUT_NUM_REG	SPI_DOUT2_NUM SPI_DOUT3_NUM SPI_DOUT4_NUM SPI_DOUT5_NUM SPI_DOUT6_NUM SPI_DOUT7_NUM

GP-SPI3 has the same 1-bit mode functions and register configuration rules as to GP-SPI2. GP-SPI3 interface can be seen as a 1-bit mode GP-SPI2 interface. By such way, the guidelines on how to use GP-SPI3 can be deduced from the material provided in Section 24.4 and Section 24.5.

24.7 CPU Controlled Data Transfer

CPU-controlled transfer, in which the source or destination of data is GP-SPI data buffer, is supported in master mode and slave mode of GP-SPI. CPU-controlled mode can be used together with full-duplex communication, half-duplex communication, and functions described in Subsection 24.4 and Subsection 24.5. As shown in Figure 24-14, GP-SPI provides 18 x 32-bit data buffers, i.e., SPI_W0_REG ~ SPI_W17_REG. The data to send should be ready before the transmission.

24.7.1 CPU Controlled Master Mode

In a master full-duplex and half-duplex CPU-controlled transmission, the readable or writable data can be to/from SPI_W0_REG[7:0] ~ SPI_W17_REG[31:24] or to/from SPI_W8_REG[7:0] ~ SPI_W17_REG[7:0] in high-part mode. The bits SPI_USR_MOSI_HIGHPART and SPI_USR_MISO_HIGHPART in the register SPI_USER_REG control the sent and received data byte order respectively:

- If SPI_USR_MOSI_HIGHPART = 0, the sent data is from SPI_W0_REG[7:0] ~ SPI_W17_REG[31:24], and

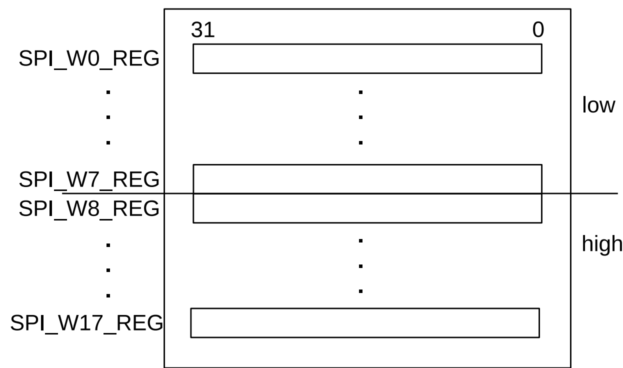


Figure 24-14. Data Buffer Used in CPU-Controlled Mode

the data address increases by 1 on each byte transferred. If the data byte length is greater than 72, the address will be the modular operation of 72, and the content of SPI_W0_REG ~ SPI_W17_REG may be sent for more than once.

- If SPI_USR_MOSI_HIGHPART = 1, the sent data is from SPI_W8_REG[7:0] ~ SPI_W17_REG[31:24], and the data address increases by 1 on each byte transferred. If the data byte length is greater than 40, the data in SPI_W8_REG[7:0] ~ SPI_W17_REG[31:24] may be sent more than once.
- If SPI_USR_MISO_HIGHPART = 0, the received data is saved to SPI_W0_REG[7:0] ~ SPI_W17_REG[31:24], and the data address increases 1 on each byte transferred. If the data byte length is greater than 72, the address will be the modular operation of 72. And the content of SPI_W0_REG ~ SPI_W17_REG may be replaced.
- If SPI_USR_MISO_HIGHPART = 1, the received data saved to SPI_W8_REG[7:0] ~ SPI_W17_REG[31:24], and the data address increases by 1 on each byte transferred. If the data byte length is greater than 40, the content of SPI_W0_REG ~ SPI_W17_REG may be replaced.

The bit SPI_DMA_RX_ENA in register SPI_DMA_IN_LINK_REG and the SPI_DMA_TX_ENA bit in register SPI_DMA_OUT_LINK_REG should be cleared during the transmission:

- SPI_DMA_RX_ENA
  - 1: receive data in DMA-controlled RX mode
  - 0: receive data in CPU-controlled mode
- SPI_DMA_TX_ENA
  - 1: send data in DMA-controlled TX mode
  - 0: send data in CPU-controlled mode

24.7.2 CPU Controlled Slave Mode

In CPU-controlled slave full-duplex mode, the buffers SPI_W0_REG ~ SPI_W17_REG are byte-addressable and the data address starts from 0 and increases by 1 on each byte. If the data address is greater than 71, only the content of SPI_W17_REG[31:24] will be updated.

In CPU-controlled slave half-duplex mode, the registers SPI_W0_REG ~ SPI_W17_REG are all 32-bit buffers and byte addressable. The ADDR value in transmission format is the start address of the read or write data, corresponding to the registers SPI_W0_REG ~ SPI_W17_REG. The read or write address increments by 1 on every byte, corresponding to SPI_W0_REG ~ SPI_W17_REG. If the address is greater than 71, the highest byte

address of SPI_W17_REG[31:24], the address will always be 71 and only the content of SPI_W17_REG[31:24] will be changed. Meanwhile, the SPI_SLV_ADDR_ERR bit in SPI_SLAVE1_REG register will be 1.

The registers SPI_W0_REG ~ SPI_W17_REG can all be used as data buffer, partial data buffer and partial status buffer, or all status buffer according to the application. In addition, the bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA can be 1 in the slave CPU-controlled mode.

24.8 DMA Controlled Data Transfer

DMA-controlled transfer, in which DMA RX module receives data and DMA TX module sends data, is supported both in master mode and in slave mode. Data is transferred by DMA engine from or to the DMA-linked memory, without CPU operation.

DMA-controlled mode can be used together with full-duplex communication, half-duplex communication and functions described in Subsection 24.4 and Subsection 24.5. Meanwhile, DMA RX module is independent from DMA TX module, which means that there are four kinds of full-duplex communications:

- Data is received in DMA-controlled mode and sent in DMA-controlled mode.
- Data is received in DMA-controlled mode and sent in CPU-controlled mode.
- Data is received in CPU-controlled mode and sent in DMA-controlled mode.
- Data is received in CPU-controlled mode and sent in CPU-controlled mode.

In half-duplex communication, the transfers of Wr_BUF, Rd_BUF, Wr_DMA and Rd_DMA can be supported in a slave segmented-transfer, see Subsection 24.5.4. GP-SPI master segmented-configure-transfer only supports DMA-controlled transfer.

Both for the DMA-controlled master mode and slave mode, the main configuration flow is as follows:

- Configure a DMA TX descriptor and set the bit SPI_OUTLINK_START in register SPI_DMA_OUT_LINK_REG to start a DMA TX engine. Before all the DMA TX buffer is used or the DMA TX engine is reset, SPI_OUTLINK_RESTART bit can be set to add a new TX buffer to the end of last TX buffer in use.
- DMA RX buffer can be linked by setting bits SPI_INLINK_START and SPI_INLINK_RESTART in the same way as the DMA TX buffer.
- The sent and received data lengths are determined by the configured DMA TX and RX buffer respectively, both of which can be 0 ~ 2 M-byte.
- DMA inlink and outlink should be initialized before DMA starts. The bit SPI_DMA_RX_ENA in register SPI_DMA_IN_LINK_REG and the bit SPI_DMA_TX_ENA in register SPI_DMA_OUT_LINK_REG should be set during the transmission.

The only difference between DMA-controlled transfers in master mode and in slave mode is on the DMA RX control:

- Clear the bit SPI_RX_EOF_EN.
  - In master mode, the interrupt SPI_IN_SUC_EOF_INT can be triggered when a single transfer or a DMA segmented-configure-transfer is ended;
  - In slave mode, the interrupt SPI_IN_SUC_EOF_INT can be triggered when a single transfer is ended and the SPI_DMA_SLV_SEG_TRANS_EN bit is 0. It can also be triggered when

SPI_DMA_SLV_SEG_TRANS_EN bit is 1 and a CMD7 or End_SEG_TRANS command is received correctly in a slave segmented-transfer.

- Set the bit SPI_RX_EOF_EN.
  - In master mode, the interrupt SPI_IN_SUC_EOF_INT can be triggered when a single transfer or a DMA segmented-configure-transfer is ended and the total DMA RX received data length is equal to the value of SPI_MST_DMA_RD_BYTELEN.
  - In slave mode, the total DMA RX received data length should be equal to the value of SPI_SLV_DMA_RD_BYTELEN. The interrupt SPI_IN_SUC_EOF_INT can be triggered when a single slave mode transfer is ended and the SPI_DMA_SLV_SEG_TRANS_EN bit is 0. It can also be triggered when SPI_DMA_SLV_SEG_TRANS_EN bit is 1 and a CMD7 or End_SEG_TRANS command is received correctly in a slave segmented-transfer.

In addition, if the configured DMA TX buffer length is smaller than the length of real transferred data, the extra data will be the same as the last transferred data. SPI_OUTFIFO_EMPTY_ERR_INT_RAW and SPI_OUT_EOF_INT_RAW in the register SPI_DMA_INT_RAW_REG will be set.

If the configured DMA TX buffer length is greater than the length of real transferred data, the TX buffer is not fully used, and the rest of the buffer will be available for further usage even if a new TX buffer is linked later. Keep it in mind or save the unused data and reset DMA.

If the configured DMA RX buffer length is smaller than the length of real transferred data, the extra data will be lost. The bit SPI_INFIFO_FULL_ERR_INT_RAW in register SPI_DMA_INT_RAW_REG and SPI_TRANS_DONE will be valid. But no valid SPI_IN_SUC_EOF_INT_RAW interrupt will be generated.

If the configured DMA RX buffer length is greater than the length of real transferred data, the RX buffer is not fully used, and the rest of the buffer will be available for further usage even if a new RX buffer is linked later. Keep it in mind or reset DMA.

24.9 GP-SPI Clock Control

In master mode, the maximum output clock frequency of GP-SPI is f_{apb}. To have slower rates, the output clock frequency can be divided as follows:

$$f_{\text{spi}} = \frac{f_{\text{apb}}}{(\text{SPI_CLKCNT_N}+1)(\text{SPI_CLKDIV_PRE}+1)}$$

The values of SPI_CLKCNT_N and SPI_CLKDIV_PRE in register SPI_CLOCK_REG can be configured to change the frequency. When the bit SPI_CLK_EQU_SYSCLK in register SPI_CLOCK_REG is 1, the output clock frequency of GP-SPI will be f_{apb}. And for other integral clock divisions, SPI_CLK_EQU_SYSCLK should be 0.

In slave mode, the maximum supported input clock frequency of GP-SPI is $f_{\text{apb}}/2$, and all the lower frequencies are available.

24.9.1 GP-SPI Clock Phase and Polarity

There are four clock modes in SPI protocol, modes 0 ~ 3, see Figure 24-15 and Figure 24-16:

1. Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge.

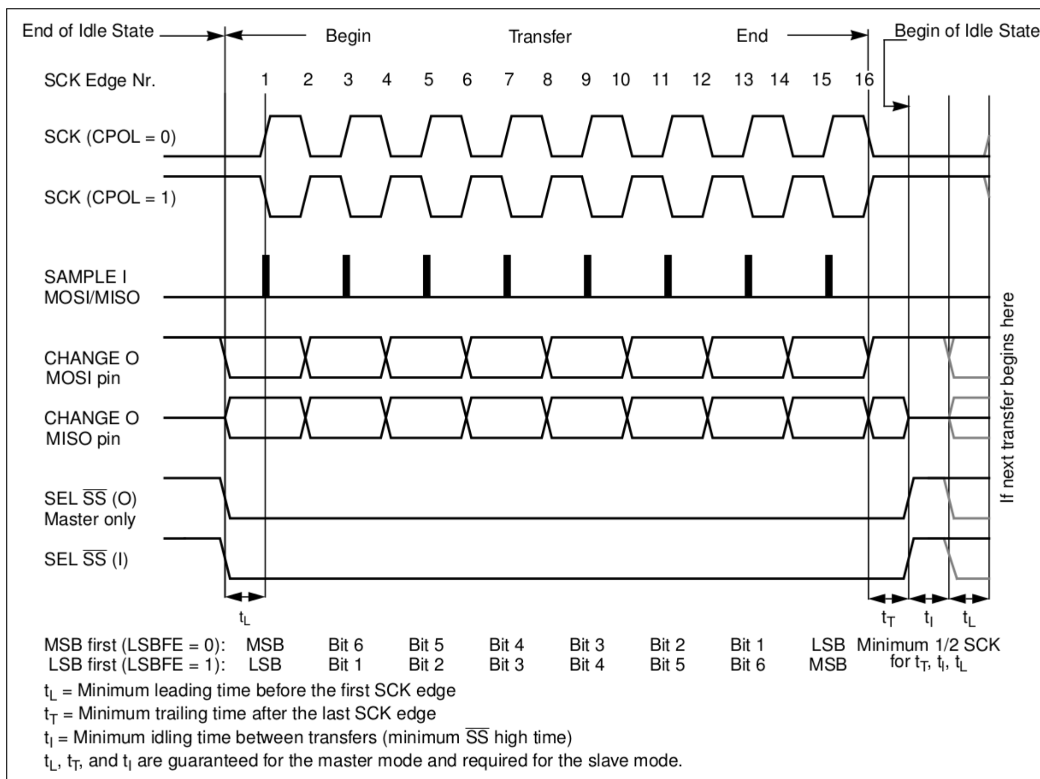


Figure 24-15. SPI CLK Mode 0 or 2

2. Mode 1: CPOL = 0, CPHA = 1; SCK is 0 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge.
3. Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge.
4. Mode 3: CPOL = 1, CPHA = 1; SCK is 1 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge.

24.9.2 GP-SPI Clock Control in Master Mode

The four clock modes 0 ~ 3 are supported in GP-SPI master mode. The polarity and phase of GP-SPI are controlled by the bit SPI_CK_IDLE_EDGE in register SPI_MISC_REG and the bit SPI_CK_OUT_EDGE in register SPI_USER_REG. The register configuration for SPI mode 0 ~ 3 can be seen in Table 140, and can be changed according to the path delay in application.

Table 140: Clock Phase and Polarity Configuration in Master Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

SPI_CLK_MODE[1:0] in register SPI_CTRL1_REG can be used to select the number of positive edges of FSPICLK, when CS raises high, to be 0, 1, 2 or FSPICLK always on.

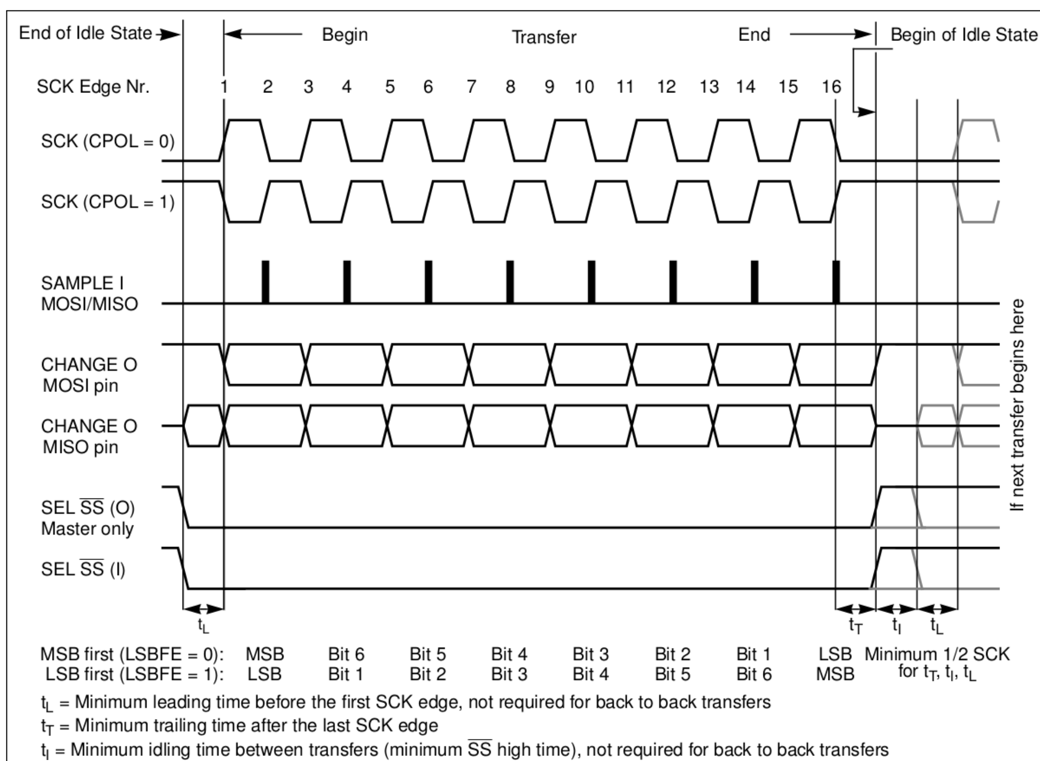


Figure 24-16. SPI CLK Mode 1 or 3

24.9.3 GP-SPI Clock Control in Slave Mode

GP-SPI slave mode also supports clock modes 0 ~ 3. The polarity and phase are configured by the bits SPI_TSCK_I_EDGE and SPI_RSCK_I_EDGE in register SPI_USER_REG. The output edge of data is controlled by SPI_CLK_MODE_13 in register SPI_CTRL1_REG. The detailed register configuration is shown in Table 141:

Table 141: Clock Phase and Polarity Configuration in Slave Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

24.9.4 GP-SPI Timing Compensation

The I/O lines can be mapped via GPIO Matrix or IO MUX. There is no timing adjustment in IO MUX. The input data and output data can be delayed for 0, 1 or 2 cycles of APB_CLK at the positive or negative edge in GPIO Matrix. The detailed register configuration can be seen in Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*.

In GP-SPI master mode, there are some timing adjustment modules for every input and output data, which can be used to achieve integral numbers of T_APB_CLK (one cycle of APB_CLK) delay on positive or negative edge. The registers SPI_DIN_MODE_REG and SPI_DIN_NUM_REG can be used to select the latch edge of input data. SPI_DOUT_MODE_REG and SPI_DOUT_NUM_REG can be used to select the latch of output data. The detailed description can be seen in Subsection 24.14. Meanwhile, the DUMMY cycle length can be changed to compensate the real I/O line delays, so as to enhance the performance of GP-SPI.

In GP-SPI slave mode, if the bit SPI_RSCK_DATA_OUT in register SPI_CTRL1_REG is set to 1, the output data

will be sent at latch edge, which will be half an SPI clock cycle earlier. It can be used for slave mode timing compensation.

24.10 SPI Pin Mapping

The IO path mapping between FSPI bus signals and GPIO pads can be seen in Figure 24-4, Figure 24-5, and Figure 24-7, which can also be used for the IO path of SPI0/1 and GP-SPI3.

The mapping between SPI/FSPI/SPI3 bus signals and GPIO pads in different communication modes is shown in Table 142. The signals in a line corresponds to each other. For example, the signal FSPID is connected to MOSI in GP-SPI2 full-duplex communication, and FSPIQ to MISO, see Figure 24-4. SPI3_Q will connect to MISO pin in half-duplex communication for GP-SPI3.

Table 142: Mapping of SPI Signal Buses and Chip Pads

Standard SPI		Extended SPI			
Full-Duplex	Half-Duplex	Chip Pad Signals			
SPI Signal Bus	SPI Signal Bus	Pin Functions	SPI Signal Bus	FSPI Signal Bus	SPI3 Signal Bus
MOSI	MOSI	D	SPID	FSPID	SPI3_D
MISO	(MISO)	Q	SPIQ	FSPIQ	SPI3_Q
CS	CS	CS	SPICS0 ~ 1	FSPICS0 ~ 5	SPI3_CS0 ~ 2
CLK	CLK	CLK	SPICLK	FSPICLK	SPI3_CLK
-	-	WP	SPIWP	FSPIWP	-
-	-	HD	SPIHD	FSPIHD	SPI3_HD
-	-	CD	-	FSPICD	SPI3_CD
-	-	DQS	SPIDQS	FSPIDQS	SPI3_DQS
-	-	IO4 ~ 7	SPIIO4 ~ 7	FSPIIO4 ~ 7	-
-	-	VSYNC	-	FSPI_VSYNC	-
-	-	HSYNC	-	FSPI_HSYNC	-
-	-	DE	-	FSPI_DE	-

24.11 GP-SPI Interrupt Control

There are two kinds of interrupts in GP-SPI2: SPI interface interrupts SPI_INT and SPI DMA interface interrupts SPI_DMA_INTR. When SPI transfer is ended, an interrupt will be generated.

The interrupt lists of GP-SPI are given in Table 143 and Table 144. Set the interrupt enable bit of SPI_INT_*_EN in register SPI_SLAVE_REG or SPI_*_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the SPI_INT or SPI_DMA_INTR interrupt. When the transfer ends, the related interrupt will be triggered and should be cleared by software before the next transfer.

Table 143: GP-SPI Master Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	SPI_IN_SUC_EOF_INT [1]
		CPU	SPI_TRANS_DONE [2]
	Half-duplex Master Output Slave Input Mode	DMA	SPI_TRANS_DONE
		CPU	SPI_TRANS_DONE
	Half-duplex Master Input Slave Output Mode	DMA	SPI_IN_SUC_EOF_INT
		CPU	SPI_TRANS_DONE
Segmented-Configure-Transfer	Full-duplex	DMA	SPI_DMA_SEG_TRANS_DONE [3]
		CPU	Not supported
	Half-duplex Master Output Slave Input Mode	DMA	SPI_DMA_SEG_TRANS_DONE
		CPU	Not supported
	Half-duplex Master Input Slave Output Mode	DMA	SPI_DMA_SEG_TRANS_DONE
		CPU	Not supported

Note:

1. If SPI_IN_SUC_EOF_INT is triggered, it means all the push data has been stored in RX buffer, and the pop data has been sent to the slave.
2. SPI_TRANS_DONE raises high when CS is high, which indicates that master has completed the data exchange in W0 ~ W17 with slave in this mode.
3. If SPI_DMA_SEG_TRANS_DONE is triggered, it means that the segmented-transfer ends, and the push data has been stored in RX buffer completely, which also indicates all the pop data has been sent out.

Table 144: GP-SPI Slave Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	SPI_IN_SUC_EOF_INT [1]
		CPU	SPI_TRANS_DONE [2]
	Half-duplex Master Output Slave Input Mode	DMA (Wr_DMA)	SPI_IN_SUC_EOF_INT[3]
		CPU (Wr_BUF)	SPI_TRANS_DONE[4]
	Half-duplex Master Input Slave Output Mode	DMA (Rd_DMA)	SPI_TRANS_DONE[5]
		CPU (Rd_BUF)	SPI_TRANS_DONE[6]
Segmented-Transfer	Full-duplex	DMA	SPI_IN_SUC_EOF_INT[7]
		CPU	Not supported[8]
	Half-duplex Master Output Slave Input Mode	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE[9]
		CPU (Wr_BUF)	Not supported[10]
	Half-duplex Master Input Slave Output Mode	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE[11]
		CPU (Rd_BUF)	Not supported[12]

Note:

1. If SPI_IN_SUC_EOF_INT is triggered, it means all the push data has been stored in RX buffer, and the pop data has been sent to slave.
2. SPI_TRANS_DONE raises high when CS is high, which indicates that master has completed the data exchange in W0 ~ W17 with slave in this mode.
3. SPI_SLV_WR_DMA_DONE just means that the transmission on the SPI bus is done, but can not ensure that all the push data has been stored in the RX buffer. For this reason, SPI_IN_SUC_EOF_INT is recommended.
4. Or wait for SPI_SLV_WR_BUF_DONE.
5. Or wait for SPI_SLV_RD_DMA_DONE.
6. Or wait for SPI_SLV_RD_BUF_DONE.
7. Slave should set the total read data byte length in SPI_SLV_DMA_RD_BYTELEN[19:0] before transmit begins. And set SPI_RX_EOF_EN 0->1 before the end of the interrupt program.
8. Master and slave should define a segmented-transfer end method, such as through GPIO as interrupt and so on.

9. Master sends COM5 to end segmented-transfer or slave sets the total read data byte length in SPI_SLV_DMA_RD_BYTELEN[19:0] and waits for SPI_DMA_IN_SUC_EOF_INT_ST.
10. Half-duplex Wr_BUF single transfer can occur during DMA segmented-transfer.
11. Master sends COM5 to end segmented-transfer.
12. Half-duplex Rd_BUF single transfer can occur during DMA segmented-transfer.

24.11.1 GP-SPI Interrupt

The description of SPI_INT interrupt sources is as follows:

- SPI_TRANS_DONE: triggered at the end of SPI bus transfer in both master mode and slave mode.
- SPI_SLV_WR_DMA_DONE: triggered at the end of Wr_DMA transfer in slave mode.
- SPI_SLV_RD_DMA_DONE: triggered at the end of Rd_DMA transfer in slave mode.
- SPI_SLV_WR_BUF_DONE: triggered at the end of Wr_BUF transfer in slave mode.
- SPI_SLV_RD_BUF_DONE: triggered at the end of Rd_BUF transfer in slave mode.
- SPI_DMA_SEG_TRANS_DONE: triggered at the end of End_SEG_TRANS transfer in GP-SPI slave segmented-transfer mode or at the end of master segmented-configure-transfer mode.
- SPI_SEG_MAGIC_ERR_INT: triggered when a Magic error occurs in CONF buffer during master DMA segmented-configure-transfer.

24.11.2 GP-SPI DMA Interrupts

The description of SPI_DMA_INTR interrupt sources is as follows:

- SPI_SLV_CMDA_INT: triggered when CMDA is received correctly in GP-SPI slave mode and the SPI transfer is ended.
- SPI_SLV_CMD9_INT: triggered when CMD9 is received correctly in GP-SPI slave mode and the SPI transfer is ended.
- SPI_SLV_CMD8_INT: triggered when CMD8 is received correctly in GP-SPI slave mode and the SPI transfer is ended.
- SPI_SLV_CMD7_INT: triggered when CMD7 is received correctly in GP-SPI slave mode and the SPI transfer is ended.
- SPI_SLV_CMD6_INT: triggered when En_QPI or Ex_QPI is received correctly in GP-SPI slave mode and the SPI transfer is ended.
- SPI_OUTFIFO_EMPTY_ERR_INT: triggered when DMA TX FIFO length is smaller than the real transferred data length.
- SPI_INFIFO_FULL_ERR_INT: triggered when DMA RX FIFO length is smaller than the real transferred data length.
- SPI_OUT_TOTAL_EOF_INT: triggered when all the data in out_link buffer has been sent out.
- SPI_OUT_EOF_INT: triggered when each out link buffer has been used up.
- SPI_OUT_DONE_INT: triggered when the length of last out link is 0.
- SPI_IN_SUC_EOF_INT: triggered when all the in_link buffers are full.
- SPI_IN_ERR_EOF_INT: triggered when there is an error in the in_link.
- SPI_IN_DONE_INT: triggered when the length of last in link is 0.
- SPI_INLINK_DSCR_ERROR_INT: triggered when there is an error in the in_link descriptor.
- SPI_OUTLINK_DSCR_ERROR_INT: triggered when the out_link in use is invalid.
- SPI_INLINK_DSCR_EMPTY_INT: triggered when there is no valid in_link descriptor.

24.12 Register Base Address

Users can access SPI0, SPI1, GP-SPI2, and GP-SPI3 registers with the base address as shown in Table 145. For more information, see Chapter 3 *System and Memory*.

Table 145: SPI Base Address

Module	Bus to Access Peripheral	Base Address
SPI0	PeriBUS1	0x3F403000
	PeriBUS2	0x60003000
SPI1	PeriBUS1	0x3F402000
	PeriBUS2	0x60002000
GP-SPI2	PeriBUS1	0x3F424000
	PeriBUS2	0x60024000
GP-SPI3	PeriBUS1	0x3F425000
	PeriBUS2	0x60025000

24.13 Register Summary

The addresses in the following table are relative to the SPI base addresses provided in Section 24.12.

Name	Description	Address	Access
User-defined control registers			
SPI_CMD_REG	Command control register	0x0000	R/W
SPI_ADDR_REG	Address value	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0018	R/W
SPI_USER1_REG	SPI USER control register 1	0x001C	R/W
SPI_USER2_REG	SPI USER control register 2	0x0020	R/W
SPI_MOSI_DLEN_REG	MOSI length	0x0024	R/W
SPI_MISO_DLEN_REG	MISO length	0x0028	R/W
Control and configuration registers			
SPI_CTRL_REG	SPI control register	0x0008	R/W
SPI_CTRL1_REG	SPI control register 1	0x000C	R/W
SPI_CTRL2_REG	SPI control register 2	0x0010	R/W
SPI_CLOCK_REG	SPI clock control register	0x0014	R/W
SPI_MISC_REG	SPI MISC register	0x002C	R/W
SPI_FSM_REG	SPI master status and DMA read byte control register	0x0044	varies
SPI_HOLD_REG	SPI hold register	0x0048	R/W
Slave mode configuration registers			
SPI_SLAVE_REG	SPI slave control register	0x0030	varies
SPI_SLAVE1_REG	SPI slave control register 1	0x0034	varies
SPI_SLV_WRBUF_DLEN_REG	SPI slave Wr_BUF interrupt and CONF control register	0x0038	R/W
SPI_SLV_RDBUF_DLEN_REG	SPI magic error and slave control register	0x003C	R/W
SPI_SLV_RD_BYTE_REG	SPI interrupt control register	0x0040	R/W
DMA configuration registers			

Name	Description	Address	Access
SPI_DMA_CONF_REG	SPI DMA control register	0x004C	R/W
SPI_DMA_OUT_LINK_REG	SPI DMA TX link configuration	0x0050	R/W
SPI_IN_ERR_EOF_DES_ADDR_REG	The latest SPI DMA RX descriptor address receiving error	0x0068	RO
SPI_IN_SUC_EOF_DES_ADDR_REG	The latest SPI DMA EOF RX descriptor address	0x006C	RO
SPI_INLINK_DSCR_REG	Current SPI DMA RX descriptor pointer	0x0070	RO
SPI_INLINK_DSCR_BF0_REG	Next SPI DMA RX descriptor pointer	0x0074	RO
SPI_OUT_EOF_BFR_DES_ADDR_REG	The latest SPI DMA EOF TX buffer address	0x007C	RO
SPI_OUT_EOF_DES_ADDR_REG	The latest SPI DMA EOF TX descriptor address	0x0080	RO
SPI_OUTLINK_DSCR_REG	Current SPI DMA TX descriptor pointer	0x0084	RO
SPI_OUTLINK_DSCR_BF0_REG	Next SPI DMA TX descriptor pointer	0x0088	RO
SPI_DMA_OUTSTATUS_REG	SPI DMA TX status	0x0090	RO
SPI_DMA_INSTATUS_REG	SPI DMA RX status	0x0094	RO
DMA interrupt registers			
SPI_DMA_IN_LINK_REG	SPI DMA RX link configuration	0x0054	R/W
SPI_DMA_INT_ENA_REG	SPI DMA interrupt enable register	0x0058	R/W
SPI_DMA_INT_RAW_REG	SPI DMA interrupt raw register	0x005C	varies
SPI_DMA_INT_ST_REG	SPI DMA interrupt status register	0x0060	varies
SPI_DMA_INT_CLR_REG	SPI DMA interrupt clear register	0x0064	R/W
CPU controlled data buffer			
SPI_W0_REG	Data buffer 0	0x0098	R/W
SPI_W1_REG	Data buffer 1	0x009C	R/W
SPI_W2_REG	Data buffer 2	0x00A0	R/W
SPI_W3_REG	Data buffer 3	0x00A4	R/W
SPI_W4_REG	Data buffer 4	0x00A8	R/W
SPI_W5_REG	Data buffer 5	0x00AC	R/W
SPI_W6_REG	Data buffer 6	0x00B0	R/W
SPI_W7_REG	Data buffer 7	0x00B4	R/W
SPI_W8_REG	Data buffer 8	0x00B8	R/W
SPI_W9_REG	Data buffer 9	0x00BC	R/W
SPI_W10_REG	Data buffer 10	0x00C0	R/W
SPI_W11_REG	Data buffer 11	0x00C4	R/W
SPI_W12_REG	Data buffer 12	0x00C8	R/W
SPI_W13_REG	Data buffer 13	0x00CC	R/W
SPI_W14_REG	Data buffer 14	0x00D0	R/W
SPI_W15_REG	Data buffer 15	0x00D4	R/W
SPI_W16_REG	Data buffer 16	0x00D8	R/W
SPI_W17_REG	Data buffer 17	0x00DC	R/W
Timing registers			
SPI_DIN_MODE_REG	SPI input delay mode configuration	0x00E0	R/W
SPI_DIN_NUM_REG	SPI input delay number configuration	0x00E4	R/W
SPI_DOUT_MODE_REG	SPI output delay mode configuration	0x00E8	R/W
SPI_DOUT_NUM_REG	SPI output delay number configuration	0x00EC	R/W

Name	Description	Address	Access
LCD control registers			
SPI_LCD_CTRL_REG	LCD frame control register	0x00F0	R/W
SPI_LCD_CTRL1_REG	LCD frame control register 1	0x00F4	R/W
SPI_LCD_CTRL2_REG	LCD frame control register 2	0x00F8	R/W
SPI_LCD_D_MODE_REG	LCD delay number	0x00FC	R/W
SPI_LCD_D_NUM_REG	LCD delay mode	0x0100	R/W
Version register			
SPI_DATE_REG	Version control register	0x03FC	R/W

24.14 Registers

Register 24.1: SPI_CMD_REG (0x0000)

(reserved)	SPI_USR	(reserved)	SPI_CONF_BITLEN	
31 ... 25	24	23	22 ... 0	
0 0 0 0 0 0 0	0	0	0	Reset

SPI_CONF_BITLEN Define the SPI_CLK cycles of SPI_CONF state. Can be configured in CONF state. (R/W)

SPI_USR User-defined command enable. An operation will be triggered when the bit is set. The bit will be cleared once the operation is done. 1: enable, 0: disable. Can not be changed by CONF_buf. (R/W)

Register 24.2: SPI_ADDR_REG (0x0004)

SPI_USR_ADDR_VALUE	
31 ... 0	
0x000000	Reset

SPI_USR_ADDR_VALUE [31:8]: the address to slave, [7:0]: Reserved. Can be configured in CONF state. (R/W)

Register 24.3: SPI_USER_REG (0x0018)

Bit(s)	Field	Reset
31	SPI_USR_COMMAND	1
30	SPI_USR_ADDR	0
29	SPI_USR_DUMMY	0
28	SPI_USR_MISO	0
27	SPI_USR_MOSI	0
26	SPI_USR_DUMMY_IDLE	0
25	SPI_USR_MOSI_HIGHPART	0
24	SPI_USR_MISO_HIGHPART	0
23:18	(reserved)	0 0 0 0 0 0
17	SPI_USR_HOLD_POL	0
16	SPI_SIO	0
15	SPI_USR_CONF_NXT	0
14	SPI_FWRITE_OCT	0
13	SPI_FWRITE_QUAD	0
12	SPI_FWRITE_DUAL	0
11	SPI_WR_BYTE_ORDER	0
10	SPI_RD_BYTE_ORDER	0
9	SPI_CK_OUT_EDGE	0
8	SPI_RSCK_I_EDGE	0
7	SPI_CS_SETUP	1
6	SPI_CS_HOLD	1
5	SPI_TSCK_I_EDGE	0
4	SPI_OPI_MODE	0
3	SPI_QPI_MODE	0
2:1	(reserved)	0 0
0	SPI_DOUTDIN	0

SPI_DOUTDIN Set the bit to enable full-duplex communication. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_QPI_MODE Both for master mode and slave mode. 1: SPI controller is in QPI mode. 0: other modes. Can be configured in CONF state. (R/W)

SPI_OPI_MODE Just for master mode. 1: SPI controller is in OPI mode (all in 8-bit mode). 0: other modes. Can be configured in CONF state. (R/W)

SPI_TSCK_I_EDGE In slave mode, this bit can be used to change the polarity of tsck. 0: TSCK = SPI_CK_I. 1: TSCK = !SPI_CK_I. (R/W)

SPI_CS_HOLD Keep SPI CS low when SPI is in DONE phase. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_CS_SETUP Enable SPI CS when SPI is in (PREP) prepare phase. 1: enable. 0: disable. Can be configured in CONF state. (R/W)

SPI_RSCK_I_EDGE In slave mode, this bit can be used to change the polarity of rsck. 0: RSCK = !SPI_CK_I. 1: RSCK = SPI_CK_I. (R/W)

SPI_CK_OUT_EDGE This bit together with SPI_DOUT_MODE is used to set MOSI signal delay mode. Can be configured in CONF state. (R/W)

SPI_RD_BYTE_ORDER In read-data (MISO) phase, 1: big-endian, 0: little-endian. Can be configured in CONF state. (R/W)

SPI_WR_BYTE_ORDER In CMD (command), ADDR (address), and write-data (MOSI) phases, 1: big-endian, 0: litte-endian. Can be configured in CONF state. (R/W)

SPI_FWRITE_DUAL In write operations, read-data phase is in 2-bit mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_QUAD In write operations, read-data phase is in 4-bit mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_OCT In write operations, read-data phase is in 8-bit mode. Can be configured in CONF state. (R/W)

SPI_USR_CONF_NXT 1: Enable the DMA CONF phase of next seg-trans operation, which means seg-trans will continue. 0: The segmented-transfer will end after the current SPI segmented-transfer or this is not segmented-transfer mode. Can be configured in CONF state. (R/W)

Continued on the next page...

Register 24.3: SPI_USER_REG (0x0018)

Continued from the previous page...

SPI_SIO Set the bit to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_HOLD_POL This bit together with the hold bits is used to set the polarity of SPI hold line. 1: SPI will be held when SPI hold line is high. 0: SPI will be held when SPI hold line is low. Can be configured in CONF state. (R/W)

SPI_USR_MISO_HIGHPART In read-data phase, access only to high-part of the buffer SPI_BUF8 ~ SPI_BUF17. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_MOSI_HIGHPART In write-data phase, access only to high-part of the buffer SPI_BUF8 ~ SPI_BUF17. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY_IDLE When this bit is enabled, the SPI clock is disabled in DUMMY phase. Can be configured in CONF state. (R/W)

SPI_USR_MOSI This bit enables the write-data phase of an operation. Can be configured in CONF state. (R/W)

SPI_USR_MISO This bit enables the read-data phase of an operation. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY This bit enables the DUMMY phase of an operation. Can be configured in CONF state. (R/W)

SPI_USR_ADDR This bit enables the address phase of an operation. Can be configured in CONF state. (R/W)

SPI_USR_COMMAND This bit enables the command phase of an operation. Can be configured in CONF state. (R/W)

Register 24.4: SPI_USER1_REG (0x001C)

SPI_USR_ADDR_BITLEN	(reserved)	SPI_USR_DUMMY_CYCLELEN	
31–27	26–8	7–0	
23	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	7	Reset

SPI_USR_DUMMY_CYCLELEN The length in SPI_CLK cycles of dummy phase. The register value shall be (cycle_num-1). Can be configured in CONF state. (R/W)

SPI_USR_ADDR_BITLEN The length in bits of address phase. The register value shall be (bit_num-1). Can be configured in CONF state. (R/W)

Register 24.5: SPI_USER2_REG (0x0020)

SPI_USR_COMMAND_BITLEN	(reserved)	SPI_USR_COMMAND_VALUE	
31–28	27–16	15–0	
7	0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SPI_USR_COMMAND_VALUE The value of command. Can be configured in CONF state. (R/W)

SPI_USR_COMMAND_BITLEN The length in bits of command phase. The register value shall be (bit_num-1). Can be configured in CONF state. (R/W)

Register 24.6: SPI_MOSI_DLEN_REG (0x0024)

(reserved)	SPI_USR_MOSI_DBITLEN	
31–23	22–0	
0 0 0 0 0 0 0 0 0	0x0000	Reset

SPI_USR_MOSI_DBITLEN The length in bits of write-data phase. The register value shall be (bit_num-1). Can be configured in CONF state. (R/W)

Register 24.7: SPI_MISO_DLEN_REG (0x0028)

(reserved)									SPI_USR_MISO_DBITLEN
31								23	22 ... 0
0	0	0	0	0	0	0	0	0	0x0000

Reset

SPI_USR_MISO_DBITLEN The length in bits of read-data phase. The register value shall be (bit_num-1). Can be configured in CONF state. (R/W)

Register 24.8: SPI_CTRL_REG (0x0008)

Bits	31–27	26	25	24–22	21	20	19	18	17	16	15	14	13–11	10	9	8	7	6	5	4	3	2–0
Field	(reserved)	SPI_WR_BIT_ORDER	SPI_RD_BIT_ORDER	(reserved)	SPI_WP_REG	(reserved)	SPI_D_POL	SPI_Q_POL	(reserved)	SPI_FREAD_OCT	SPI_FREAD_QUAD	SPI_FREAD_DUAL	(reserved)	SPI_FCMD_OCT	SPI_FCMD_QUAD	SPI_FCMD_DUAL	SPI_FADDR_OCT	SPI_FADDR_QUAD	SPI_FADDR_DUAL	(reserved)	SPI_DUMMY_OUT	(reserved)
Reset	0 0 0 0 0	0	0	0 0 0	1	0	1	1	0	0	0	0	0 0 0	0	0	0	0	0	0	0	0	0 0 0

SPI_DUMMY_OUT The signal level output by the SPI controller in the dummy phase. Can be configured in CONF state. (R/W)

SPI_FADDR_DUAL Apply 2-bit mode during address phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FADDR_QUAD Apply 4-bit mode during address phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FADDR_OCT Apply 8-bit mode during address phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_DUAL Apply 2-bit mode during command phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_QUAD Apply 4-bit mode during command phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_OCT Apply 8-bit mode during command phase. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_DUAL In the read operations, read-data phase is in 2-bit mode. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_QUAD In the read operations, read-data phase is in 4-bit mode. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_OCT In the read operations, read-data phase is in 8-bit mode. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_Q_POL The bit is used to set MISO line polarity. 1: high, 0: low. Can be configured in CONF state. (R/W)

SPI_D_POL The bit is used to set MOSI line polarity. 1: high, 0: low. Can be configured in CONF state. (R/W)

SPI_WP_REG Write protect signal output when SPI is idle. 1: output high, 0: output low. Can be configured in CONF state. (R/W)

SPI_RD_BIT_ORDER In read-data (MISO) phase, 1: LSB first, 0: MSB first. Can be configured in CONF state. (R/W)

SPI_WR_BIT_ORDER In command, address, and write-data (MOSI) phases, 1: LSB first, 0: MSB first. Can be configured in CONF state. (R/W)

Register 24.9: SPI_CTRL1_REG (0x000C)

(reserved)	SPI_CS_HOLD_DELAY	(reserved)	SPI_W16_17_WR_ENA	SPI_RSCK_DATA_OUT	SPI_CLK_MODE_13	SPI_CLK_MODE	
31–20	19–14	13–5	4	3	2	1–0	
0 0 0 0 0 0 0 0 0 0 0 0	0x1	0 0 0 0 0 0 0 0 0	1	0	0	0x0	Reset

SPI_CLK_MODE SPI clock mode bits. 0: SPI clock is off when CS inactive. 1: SPI clock is delayed one cycle after CS inactive. 2: SPI clock is delayed two cycles after CS inactive. 3: SPI clock is always on. Can be configured in CONF state. (R/W)

SPI_CLK_MODE_13 CPOL, CPHA, 1: support SPI CLK mode 1 and 3, and output data B[0]/B[7] at first edge. 0: support SPI CLK mode 0 and 2, and output data B[1]/B[6] at first edge. (R/W)

SPI_RSCK_DATA_OUT Save half a cycle when tsck is the same as rsck. 1: output data at rsck positive edge. 0: output data at tsck positive edge. (R/W)

SPI_W16_17_WR_ENA 1: SPI_BUF16 ~ SPI_BUF17 can be written. 0: SPI_BUF16 ~ SPI_BUF17 can not be written. Can be configured in CONF state. (R/W)

SPI_CS_HOLD_DELAY SPI CS signal is delayed by SPI clock cycles. Can be configured in CONF state. (R/W)

Register 24.10: SPI_CTRL2_REG (0x0010)

(reserved)	SPI_CS_DELAY_NUM	SPI_CS_DELAY_MODE	SPI_CS_HOLD_TIME	SPI_CS_SETUP_TIME	
31	30–29	28–26	25–13	12–0	
0	0x0	0x0	0x01	0x00	Reset

SPI_CS_SETUP_TIME (cycles+1) of prepare phase by SPI clock. These bits are used together with SPI_CS_SETUP bit. Can be configured in CONF state. (R/W)

SPI_CS_HOLD_TIME Delay cycles of CS pin by SPI clock. These bits are used together with SPI_CS_HOLD bit. Can be configured in CONF state. (R/W)

SPI_CS_DELAY_MODE SPI_CS signal is delayed by SPI_CLK. 0: zero cycle. 1: if SPI_CK_OUT_EDGE or SPI_CK_IDLE_EDGE is set to 1, SPI_CLK will be delayed by half cycle, else delayed by one cycle. 2: if SPI_CK_OUT_EDGE or SPI_CK_IDLE_EDGE is set to 1, SPI_CLK will be delayed by one cycle, else delayed by half cycle. 3: delay one cycle. Can be configured in CONF state. (R/W)

SPI_CS_DELAY_NUM SPI_CS signal is delayed by system clock cycles. Can be configured in CONF state. (R/W)

Register 24.11: SPI_CLOCK_REG (0x0014)

SPI_CLK_EQU_SYSCLK	SPI_CLKDIV_PRE	SPI_CLKCNT_N	SPI_CLKCNT_H	SPI_CLKCNT_L	
31	30 — 18	17 — 12	11 — 6	5 — 0	
1	0	0x3	0x1	0x3	Reset

SPI_CLKCNT_L In the master mode it must be equal to SPI_CLKCNT_N. In the slave mode it must be 0. Can be configured in CONF state. (R/W)

SPI_CLKCNT_H In the master mode it must be floor((SPI_CLKCNT_N+1)/2-1). floor() here is to down round a number, floor(2.2) = 2. In the slave mode it must be 0. Can be configured in CONF state. (R/W)

SPI_CLKCNT_N In the master mode it is the divider of SPI_CLK. So SPI_CLK frequency is f_{apb}/(SPI_CLKDIV_PRE+1)/(SPI_CLKCNT_N+1). Can be configured in CONF state. (R/W)

SPI_CLKDIV_PRE In the master mode it is pre-divider of SPI_CLK. Can be configured in CONF state. (R/W)

SPI_CLK_EQU_SYSCLK In the master mode, 1: SPI_CLK is equal to APB clock. 0: SPI_CLK is obtained by dividing from APB clock. Can be configured in CONF state. (R/W)

Register 24.12: SPI_MISC_REG (0x002C)

Field	SPI_QUAD_DIN_PIN_SWAP	SPI_CS_KEEP_ACTIVE	SPI_CK_IDLE_EDGE	(reserved)	SPI_CD_IDLE_EDGE	SPI_CD_CMD_SET	SPI_DQS_IDLE_EDGE	SPI_SLAVE_CS_POL	SPI_CD_ADDR_SET	SPI_CD_DUMMY_SET	SPI_CD_DATA_SET	SPI_CMD_DTR_EN	SPI_ADDR_DTR_EN	SPI_DATA_DTR_EN	SPI_CLK_DATA_DTR_EN	(reserved)	SPI_MASTER_CS_POL	SPI_CK_DIS	SPI_CS5_DIS	SPI_CS4_DIS	SPI_CS3_DIS	SPI_CS2_DIS	SPI_CS1_DIS	SPI_CS0_DIS
Bits	31	30	29	28–27	26	25	24	23	22	21	20	19	18	17	16	15–13	12–7	6	5	4	3	2	1	0
Reset	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0 0 0	0	0	1	1	1	1	1	0

SPI_CS0_DIS SPI CS0 pin disable bit. 1: disable CS0, 0: SPI_CS0 signal is from/to CS0 pin. Can be configured in CONF state. (R/W)

SPI_CS1_DIS SPI CS1 pin disable bit. 1: disable CS1, 0: SPI_CS1 signal is from/to CS1 pin. Can be configured in CONF state. (R/W)

SPI_CS2_DIS SPI CS2 pin disable bit. 1: disable CS2, 0: SPI_CS2 signal is from/to CS2 pin. Can be configured in CONF state. (R/W)

SPI_CS3_DIS SPI CS3 pin disable bit. 1: disable CS3, 0: SPI_CS3 signal is from/to CS3 pin. Can be configured in CONF state. (R/W)

SPI_CS4_DIS SPI CS4 pin disable bit. 1: disable CS4, 0: SPI_CS4 signal is from/to CS4 pin. Can be configured in CONF state. (R/W)

SPI_CS5_DIS SPI CS5 pin disable bit. 1: disable CS5, 0: SPI_CS5 signal is from/to CS5 pin. Can be configured in CONF state. (R/W)

SPI_CK_DIS 1: SPI CLK output disable, 0: SPI CLK output enable. Can be configured in CONF state. (R/W)

SPI_MASTER_CS_POL In master mode, the bits are the polarity of SPI CS line. The value is equivalent to SPI_CS ^ SPI_MASTER_CS_POL. Can be configured in CONF state. (R/W)

SPI_CLK_DATA_DTR_EN 1: SPI master DTR mode is applied to SPI CLK, data and SPI_DQS. 0: SPI master DTR mode is only applied to SPI_DQS. This bit should be used with bit 17/18/19. (R/W)

SPI_DATA_DTR_EN 1: SPI CLK and data of SPI_DOUT and SPI_DIN states are in DTR mode, including master 1/2/4/8-bit mode. 0: SPI CLK and DATA of SPI_DOUT and SPI_DIN states are in STR mode. Can be configured in CONF state. (R/W)

SPI_ADDR_DTR_EN 1: SPI CLK and data of SPI_SEND_ADDR state are in DTR mode, including master 1/2/4/8-bit mode. 0: SPI CLK and data of SPI_SEND_ADDR state are in STR mode. Can be configured in CONF state. (R/W)

SPI_CMD_DTR_EN 1: SPI CLK and data of SPI_SEND_CMD state are in DTR mode, including master 1/2/4/8-bit mode. 0: SPI CLK and data of SPI_SEND_CMD state are in STR mode. Can be configured in CONF state. (R/W)

SPI_CD_DATA_SET 1: SPI_CD = !SPI_CD_IDLE_EDGE when SPI_ST[3:0] is in SPI_DOUT or SPI_DIN state. 0: SPI_CD = SPI_CD_IDLE_EDGE. Can be configured in CONF state. (R/W)

Continued on the next page...

Register 24.12: SPI_MISC_REG (0x002C)

Continued from the previous page...

SPI_CD_DUMMY_SET 1: SPI_CD = !SPI_CD_IDLE_EDGE when SPI_ST[3:0] is in SPI_DUMMY state. 0: SPI_CD = SPI_CD_IDLE_EDGE. Can be configured in CONF state. (R/W)

SPI_CD_ADDR_SET 1: SPI_CD = !SPI_CD_IDLE_EDGE when SPI_ST[3:0] is in SPI_SEND_ADDR state. 0: SPI_CD = SPI_CD_IDLE_EDGE. Can be configured in CONF state. (R/W)

SPI_SLAVE_CS_POL Select polarity of SPI slave input CS. 1: inverted, 0: not change. Can be configured in CONF state. (R/W)

SPI_DQS_IDLE_EDGE The default value of SPI_DQS. Can be configured in CONF state. (R/W)

SPI_CD_CMD_SET 1: SPI_CD = !SPI_CD_IDLE_EDGE when SPI_ST[3:0] is in SPI_SEND_CMD state. 0: SPI_CD = SPI_CD_IDLE_EDGE. Can be configured in CONF state. (R/W)

SPI_CD_IDLE_EDGE The default value of SPI_CD. Can be configured in CONF state. (R/W)

SPI_CK_IDLE_EDGE 1: SPI CLK line is high when idle. 0: SPI CLK line is low when idle. Can be configured in CONF state. (R/W)

SPI_CS_KEEP_ACTIVE Set this bit to keep the SPI CS line low. Can be configured in CONF state. (R/W)

SPI_QUAD_DIN_PIN_SWAP 1: SPI quad input swap enable. 0: SPI quad input swap disable. Can be configured in CONF state. (R/W)

Register 24.13: SPI_FSM_REG (0x0044)

SPI_MST_DMA_RD_BYTELEN	(reserved)	SPI_ST	
31 … 12	11 … 4	3 … 0	
0x000	0 0 0 0 0 0 0 0	0	Reset

SPI_ST The status of SPI state machine. 0: idle state, 1: preparation state, 2: send command state, 3: send data state, 4: read data state, 5: write data state, 6: wait state, 7: done state. (RO)

SPI_MST_DMA_RD_BYTELEN Define the master DMA read byte length in segmented-configure-transfer mode or in other modes. Invalid when SPI_RX_EOF_EN is 0. Can be configured in CONF state. (R/W)

Register 24.14: SPI_HOLD_REG (0x0048)

31 … 8	7	6 … 4	3	2	1 … 0	
(reserved)	SPI_DMA_SEG_TRANS_DONE	SPI_HOLD_OUT_TIME	SPI_HOLD_OUT_EN	SPI_HOLD_VAL_REG	(reserved)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0	Reset

SPI_HOLD_VAL_REG SPI output hold value, which should be used with SPI_HOLD_OUT_EN. Can be configured in CONF state. (R/W)

SPI_HOLD_OUT_EN Enable set SPI output hold value to SPI_HOLD_REG. It can be used to hold SPI state machine with SPI_EXT_HOLD_EN and other user hold signals. Can be configured in CONF state. (R/W)

SPI_HOLD_OUT_TIME Set the hold cycles of output SPI_HOLD signal when SPI_HOLD_OUT_EN is enabled. Can be configured in CONF state. (R/W)

SPI_DMA_SEG_TRANS_DONE 1: SPI master DMA full-duplex/half-duplex segmented-configure-transfer ends or slave half-duplex segmented-transfer ends. And data has been pushed to corresponding memory. 0: segmented-transfer or segmented-configure-transfer is not ended or not occurred. Can not be changed by CONF_buf. (R/W)

Register 24.15: SPI_SLAVE_REG (0x0030)

Bits	31	30	29	28–27	26–23	22–12	11	10	9	8	7	6	5	4	3–0
Field	SPI_SOFT_RESET	SPI_SLAVE_MODE	SPI_TRANS_DONE_AUTO_CLR_EN	(reserved)	SPI_TRANS_CNT	(reserved)	SPI_SEG_MAGIC_ERR_INT_EN	SPI_INT_DMA_SEG_TRANS_EN	SPI_INT_TRANS_DONE_EN	SPI_INT_WR_DMA_DONE_EN	SPI_INT_RD_DMA_DONE_EN	SPI_INT_WR_BUF_DONE_EN	SPI_INT_RD_BUF_DONE_EN	SPI_TRANS_DONE	(reserved)
Reset	0	0	0	0 0	0	0 0 0 0 0 0 0 0 0 0 0	0	0	1	0	0	0	0	0	0 0 0 0

SPI_TRANS_DONE The interrupt raw bit for the completion of any operation in both the master mode and the slave mode. Can not be changed by CONF_buf. (R/W)

SPI_INT_RD_BUF_DONE_EN SPI_SLV_RD_BUF_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_INT_WR_BUF_DONE_EN SPI_SLV_WR_BUF_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_INT_RD_DMA_DONE_EN SPI_SLV_RD_DMA_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_INT_WR_DMA_DONE_EN SPI_SLV_WR_DMA_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_INT_TRANS_DONE_EN SPI_TRANS_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_INT_DMA_SEG_TRANS_EN SPI_DMA_SEG_TRANS_DONE interrupt enable. 1: enable, 0: disable. Can be configured in CONF state. (R/W)

SPI_SEG_MAGIC_ERR_INT_EN 1: Enable magic value error interrupt. 0: Others. Can be configured in CONF state. (R/W)

SPI_TRANS_CNT The operations counter in both the master mode and the slave mode. (RO)

SPI_TRANS_DONE_AUTO_CLR_EN SPI_TRANS_DONE auto clear enable, clear it 3 APB cycles after the positive edge of SPI_TRANS_DONE. 0: disable. 1: enable. Can be configured in CONF state. (R/W)

SPI_SLAVE_MODE Set SPI work mode. 1: slave mode, 0: master mode. (R/W)

SPI_SOFT_RESET Software reset enable, to reset the SPI clock line, CS line and data lines. Can be configured in CONF state. (R/W)

Register 24.16: SPI_SLAVE1_REG (0x0034)

SPI_SLV_LAST_ADDR	SPI_SLV_LAST_COMMAND	SPI_SLV_WR_DMA_DONE	SPI_SLV_CMD_ERR	SPI_SLV_ADDR_ERR	SPI_SLV_NO_QPI_EN	SPI_SLV_CMD_ERR_CLR	SPI_SLV_ADDR_ERR_CLR	(reserved)	
31–24	23–16	15	14	13	12	11	10	9–0	
0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

SPI_SLV_ADDR_ERR_CLR 1: Clear SPI_SLV_ADDR_ERR. 0: not valid. Can be changed by CONF_buf. (R/W)

SPI_SLV_CMD_ERR_CLR 1: Clear SPI_SLV_CMD_ERR. 0: not valid. Can be changed by CONF_buf. (R/W)

SPI_SLV_NO_QPI_EN 1: SPI slave QPI mode is not supported. 0: SPI slave QPI mode is supported. (R/W)

SPI_SLV_ADDR_ERR 1: The address value of the last SPI transfer is not supported by SPI slave. 0: The address value is supported or no address value is received. (RO)

SPI_SLV_CMD_ERR 1: The command value of the last SPI transfer is not supported by SPI slave. 0: The command value is supported or no command value is received. (RO)

SPI_SLV_WR_DMA_DONE The interrupt raw bit for the completion of DMA write operation in slave mode. Can not be changed by CONF_buf. (R/W)

SPI_SLV_LAST_COMMAND In slave mode, it is the value of command. (R/W)

SPI_SLV_LAST_ADDR In slave mode, it is the value of address. (R/W)

Register 24.17: SPI_SLV_WRBUF_DLEN_REG (0x0038)

SPI_CONF_BASE_BITLEN	SPI_SLV_WR_BUF_DONE	(reserved)	
31–25	24	23–0	
108	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_SLV_WR_BUF_DONE The interrupt raw bit for the completion of write-buffer operation in slave mode. Can not be changed by CONF_buf. (R/W)

SPI_CONF_BASE_BITLEN The basic SPI_CLK cycles of CONF state. The real cycle length of CONF state, if SPI_USR_CONF is enabled, is SPI_CONF_BASE_BITLEN[6:0] + SPI_CONF_BITLEN[23:0]. (R/W)

Register 24.18: SPI_SLV_RDBUF_DLEN_REG (0x003C)

31–26	25	24	23–20	19–0	
(reserved)	SPI_SEG_MAGIC_ERR	SPI_SLV_RD_BUF_DONE	(reserved)	SPI_SLV_DMA_RD_BYTELEN	
0 0 0 0 0 0	0	0	0 0 0 0	0x000	Reset

SPI_SLV_DMA_RD_BYTELEN In slave mode, it is the length in bytes for read operations. The register value shall be byte_num. (R/W)

SPI_SLV_RD_BUF_DONE The interrupt raw bit for the completion of read-buffer operation in slave mode. Can not be changed by CONF_buf. (R/W)

SPI_SEG_MAGIC_ERR 1: The recent magic value in CONF buffer is not right in master DMA segmented-configure-transfer mode. 0: others. (R/W)

Register 24.19: SPI_SLV_RD_BYTE_REG (0x0040)

31	30	29 28	27 24	23	22	21	20	19 0	
SPI_USR_CONF	SPI_SLV_RD_DMA_DONE	(reserved)	SPI_DMA_SEG_MAGIC_VALUE	SPI_SLV_WRBUF_BYTELEN_EN	SPI_SLV_RDBUF_BYTELEN_EN	SPI_SLV_WRDMA_BYTELEN_EN	SPI_SLV_RDDMA_BYTELEN_EN	SPI_SLV_DATA_BYTELEN	
0	0	0 0	10	0	0	0	0	0	Reset

SPI_SLV_DATA_BYTELEN The full-duplex or half-duplex data byte length of the last SPI transfer in slave mode. In half-duplex mode, this value is controlled by bits [23:20]. (R/W)

SPI_SLV_RDDMA_BYTELEN_EN 1: SPI_SLV_DATA_BYTELEN stores data byte length of master-read-slave data length in DMA controlled mode (Rd_DMA). 0: others (R/W)

SPI_SLV_WRDMA_BYTELEN_EN 1: SPI_SLV_DATA_BYTELEN stores data byte length of master-write-to-slave data length in DMA controlled mode (Wr_DMA). 0: others (R/W)

SPI_SLV_RDBUF_BYTELEN_EN 1: SPI_SLV_DATA_BYTELEN stores data byte length of master-read-slave data length in CPU controlled mode (Rd_BUF). 0: others (R/W)

SPI_SLV_WRBUF_BYTELEN_EN 1: SPI_SLV_DATA_BYTELEN stores data byte length of master-write-to-slave data length in CPU controlled mode (Wr_BUF). 0: others (R/W)

SPI_DMA_SEG_MAGIC_VALUE The magic value of BM table in master DMA segmented-configure-transfer. (R/W)

SPI_SLV_RD_DMA_DONE The interrupt raw bit for the completion of Rd_DMA operation in slave mode. Can not be changed by CONF_buf. (R/W)

SPI_USR_CONF 1: Enable the DMA CONF phase of current segmented-configure-transfer operation, which means segmented-configure-transfer will start. 0: This is not segmented-configure-transfer mode. (R/W)

Register 24.20: SPI_DMA_CONF_REG (0x004C)

Bits	Field	Reset
31:29	(reserved)	0 0 0
28	SPI_DMA_SEG_TRANS_CLR	0
27:26	SPI_EXT_MEM_BK_SIZE	0
25:24	(reserved)	0 0
23	SPI_DMA_OUTFIFO_EMPTY_CLR	0
22	SPI_DMA_INFIFO_FULL_CLR	0
21	SPI_RX_EOF_EN	0
20	SPI_SLV_TX_SEG_TRANS_CLR_EN	0
19	SPI_SLV_RX_SEG_TRANS_CLR_EN	0
18	SPI_DMA_SLV_SEG_TRANS_EN	0
17	SPI_SLV_LAST_SEG_POP_CLR	0
16	SPI_DMA_CONTINUE	0
15	SPI_DMA_TX_STOP	0
14	SPI_DMA_RX_STOP	0
13	SPI_MEM_TRANS_EN	0
12	SPI_OUT_DATA_BURST_EN	0
11	SPI_INDSCR_BURST_EN	0
10	SPI_OUTDSCR_BURST_EN	0
9	SPI_OUT_EOF_MODE	1
8:6	(reserved)	0 0 0
5	SPI_AHBM_RST	0
4	SPI_AHBM_FIFO_RST	0
3	SPI_OUT_RST	0
2	SPI_IN_RST	0
1:0	(reserved)	0 0

SPI_IN_RST The bit is used to reset in_DMA FSM and in_data FIFO pointer. (R/W)

SPI_OUT_RST The bit is used to reset out_DMA FSM and out_data FIFO pointer. (R/W)

SPI_AHBM_FIFO_RST Reset SPI DMA AHB master FIFO pointer. (R/W)

SPI_AHBM_RST Reset SPI DMA AHB master. (R/W)

SPI_OUT_EOF_MODE out_EOF flag generation mode. 1: when DMA pops all data from FIFO. 0: when AHB pushes all data to FIFO. (R/W)

SPI_OUTDSCR_BURST_EN Read descriptor uses burst mode when read data from memory. (R/W)

SPI_INDSCR_BURST_EN Read descriptor uses burst mode when write data to memory. (R/W)

SPI_OUT_DATA_BURST_EN SPI DMA reads data from memory in burst mode. (R/W)

SPI_MEM_TRANS_EN 1: Internal memory data transfer enable bit. Send SPI DMA RX buffer data to SPI DMA TX buffer. 0: Disable this function. (R/W)

SPI_DMA_RX_STOP SPI DMA read data stops when in continuous tx/rx mode. (R/W)

SPI_DMA_TX_STOP SPI DMA write data stops when in continuous tx/rx mode. (R/W)

SPI_DMA_CONTINUE SPI DMA continues sending/receiving data. (R/W)

SPI_SLV_LAST_SEG_POP_CLR Before slave segmented-transfer starts, set first and then clear this bit, to clear the data not used in last transfer. (R/W)

SPI_DMA_SLV_SEG_TRANS_EN Enable DMA segment transfer in SPI DMA half slave mode. 1: enable. 0: disable. (R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN In DMA-controlled half-duplex slave transfer, if the size of DMA RX buffer is smaller than the size of the received data, 1: the data in next transfer will not be received; 0: the transferred data this time will not be received, but in the next transfer, if the size of DMA RX buffer is not 0, the following transferred data will be received, otherwise not. (R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN In DMA-controlled half-duplex slave transfer, if the size of DMA TX buffer is smaller than the size of the transferred data, 1: the data in next transfer will not be updated, i.e. the old data is transferred repeatedly; 0: the data in this transfer will not be updated. But in next transfer, if new data is filled in DMA TX FIFO, new data will be transferred, otherwise not. (R/W)

Continued on the next page...

Register 24.20: SPI_DMA_CONF_REG (0x004C)

Continued from the previous page...

SPI_RX_EOF_EN 1: SPI_IN_SUC_EOF_INT_RAW is set when the number of DMA pushed data bytes is equal to the value of SPI_SLV_DMA_RD_BYTELEN[19:0]/ SPI_MST_DMA_RD_BYTELEN[19:0] in SPI DMA transition. 0: SPI_IN_SUC_EOF_INT_RAW is set by SPI_TRANS_DONE in non-segment-transfer or SPI_DMA_SEG_TRANS_DONE in segment-transfer. (R/W)

SPI_DMA_INFIFO_FULL_CLR In DMA-controlled half-duplex slave transfer, if the bit SPI_SLV_RX_SEG_TRANS_CLR_EN is set and the size of DMA RX buffer is smaller than the size of the transferred data, the bit SPI_DMA_INFIFO_FULL_CLR should be set first and then cleared, to avoid affecting next transfer. (R/W)

SPI_DMA_OUTFIFO_EMPTY_CLR In DMA-controlled half-duplex slave transfer, if the bit SPI_SLV_TX_SEG_TRANS_CLR_EN is set and the size of DMA TX buffer is smaller than the size of the transferred data, this bit SPI_DMA_OUTFIFO_EMPTY_CLR should be set first and then cleared, to avoid affecting next transfer. (R/W)

SPI_EXT_MEM_BK_SIZE Select the external memory block size. (R/W)

SPI_DMA_SEG_TRANS_CLR 1: End slave segment-transfer, which acts as 0x05 command. 2 or more: end segment-transfer signals will induce error in DMA RX. 0: others. This bit will be cleared in 1 APB CLK cycle by hardware. (R/W)

Register 24.21: SPI_DMA_OUT_LINK_REG (0x0050)

SPI_DMA_TX_ENA	SPI_OUTLINK_RESTART	SPI_OUTLINK_START	SPI_OUTLINK_STOP	(reserved)	SPI_OUTLINK_ADDR	
31	30	29	28	27 … 20	19 … 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000	Reset

SPI_OUTLINK_ADDR The address of the first outlink descriptor. (R/W)

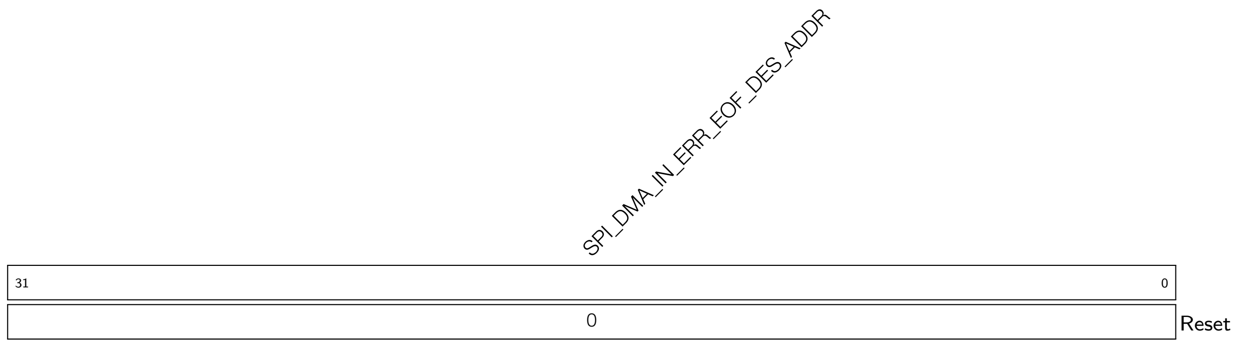
SPI_OUTLINK_STOP Set the bit to stop using outlink descriptor. (R/W)

SPI_OUTLINK_START Set the bit to start using outlink descriptor. (R/W)

SPI_OUTLINK_RESTART Set the bit to mount on new outlink descriptors. (R/W)

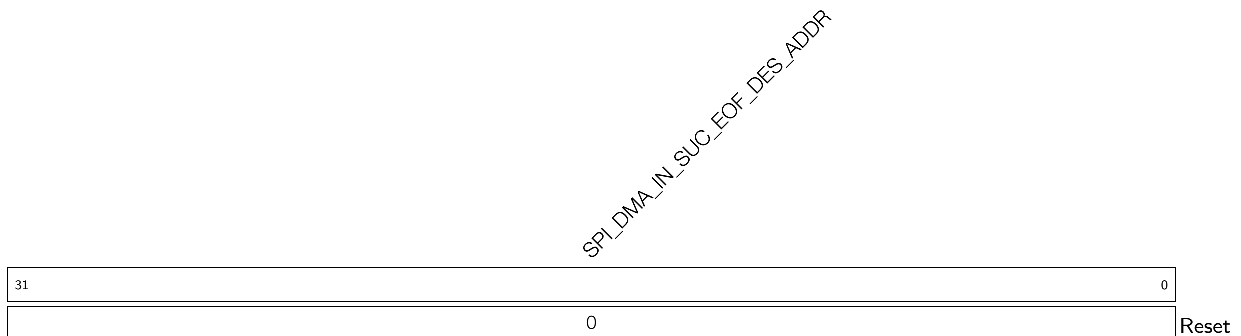
SPI_DMA_TX_ENA 1: enable DMA controlled TX mode. 0: enable CPU controlled TX mode. (R/W)

Register 24.22: SPI_IN_ERR_EOF_DES_ADDR_REG (0x0068)



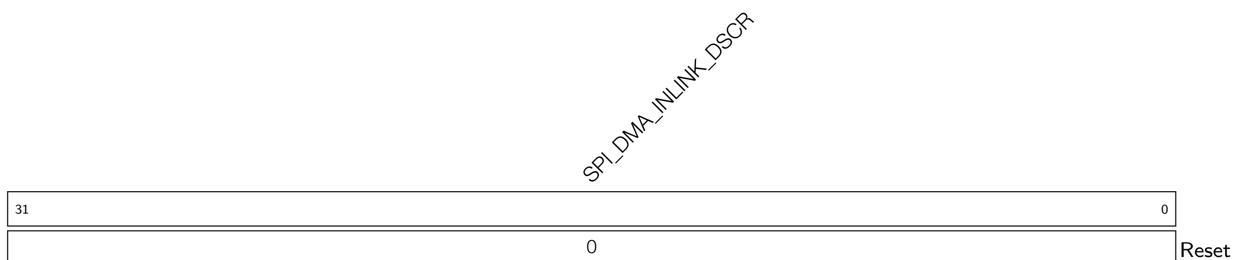
SPI_DMA_IN_ERR_EOF_DES_ADDR The inlink descriptor address when SPI DMA generates receiving error. (RO)

Register 24.23: SPI_IN_SUC_EOF_DES_ADDR_REG (0x006C)



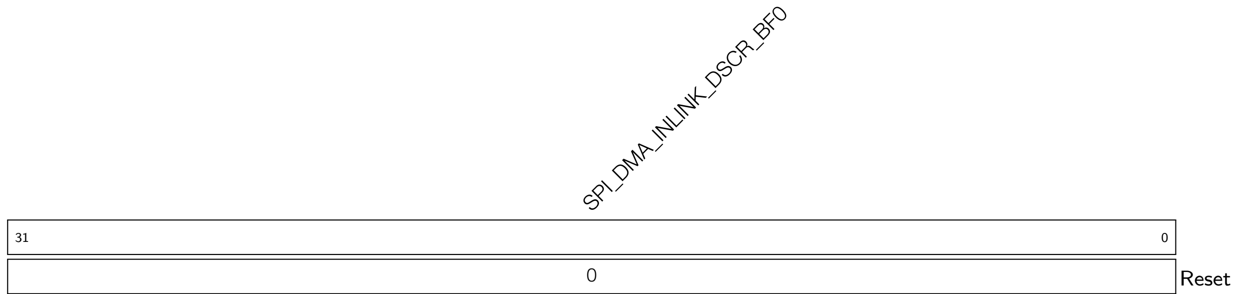
SPI_DMA_IN_SUC_EOF_DES_ADDR The last inlink descriptor address when SPI DMA generates FROM_SUC_EOF. (RO)

Register 24.24: SPI_INLINK_DSCR_REG (0x0070)



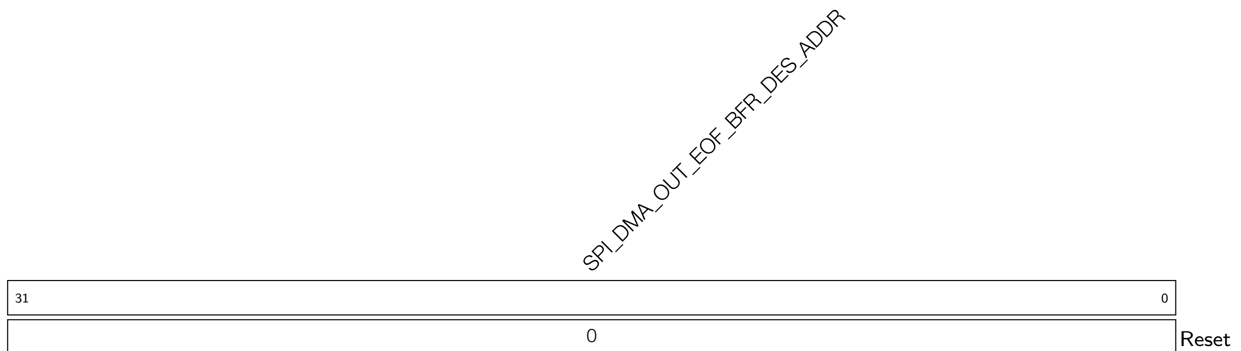
SPI_DMA_INLINK_DSCR The content of current inlink descriptor pointer. (RO)

Register 24.25: SPI_INLINK_DSCR_BF0_REG (0x0074)



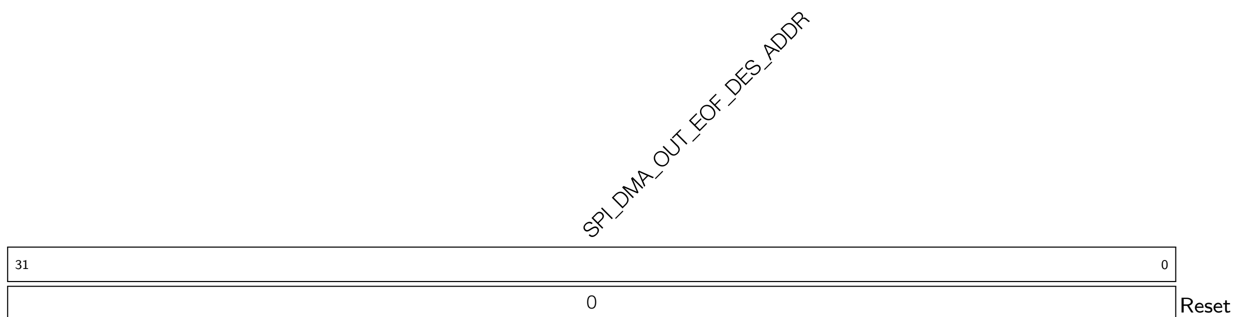
SPI_DMA_INLINK_DSCR_BF0 The content of next inlink descriptor pointer. (RO)

Register 24.26: SPI_OUT_EOF_BFR_DES_ADDR_REG (0x007C)



SPI_DMA_OUT_EOF_BFR_DES_ADDR The address of buffer relative to the outlink descriptor that generates EOF. (RO)

Register 24.27: SPI_OUT_EOF_DES_ADDR_REG (0x0080)



SPI_DMA_OUT_EOF_DES_ADDR The last outlink descriptor address when SPI DMA generates TO_EOF. (RO)

Register 24.28: SPI_OUTLINK_DSCR_REG (0x0084)

SPI_DMA_OUTLINK_DSCR	
31 … 0	
0	Reset

SPI_DMA_OUTLINK_DSCR The content of current outlink descriptor pointer. (RO)

Register 24.29: SPI_OUTLINK_DSCR_BF0_REG (0x0088)

SPI_DMA_OUTLINK_DSCR_BF0	
31 … 0	
0	Reset

SPI_DMA_OUTLINK_DSCR_BF0 The content of next outlink descriptor pointer. (RO)

Register 24.30: SPI_DMA_OUTSTATUS_REG (0x0090)

SPI_DMA_OUTFIFO_EMPTY	SPI_DMA_OUTFIFO_FULL	SPI_DMA_OUTFIFO_CNT	SPI_DMA_OUT_STATE	SPI_DMA_OUTDSCR_STATE	SPI_DMA_OUTDSCR_ADDR	
31	30	29 … 23	22 … 20	19 … 18	17 … 0	
1	0	0	0	0	0	Reset

SPI_DMA_OUTDSCR_ADDR SPI DMA out descriptor address. (RO)

SPI_DMA_OUTDSCR_STATE SPI DMA out descriptor state. (RO)

SPI_DMA_OUT_STATE SPI DMA out data state. (RO)

SPI_DMA_OUTFIFO_CNT The remaining part of SPI DMA out_FIFO data. (RO)

SPI_DMA_OUTFIFO_FULL SPI DMA out_FIFO is full. (RO)

SPI_DMA_OUTFIFO_EMPTY SPI DMA out_FIFO is empty. (RO)

Register 24.31: SPI_DMA_INSTATUS_REG (0x0094)

SPI_DMA_INFIFO_EMPTY	SPI_DMA_INFIFO_FULL	SPI_DMA_INFIFO_CNT	SPI_DMA_IN_STATE	SPI_DMA_INDSCR_STATE	SPI_DMA_INDSCR_ADDR	
31	30	29 – 23	22 – 20	19 – 18	17 – 0	
1	0	0	0	0	0	Reset

SPI_DMA_INDSCR_ADDR SPI DMA in descriptor address. (RO)

SPI_DMA_INDSCR_STATE SPI DMA in descriptor state. (RO)

SPI_DMA_IN_STATE SPI DMA in data state. (RO)

SPI_DMA_INFIFO_CNT The remaining part of SPI DMA inFIFO data. (RO)

SPI_DMA_INFIFO_FULL SPI DMA inFIFO is full. (RO)

SPI_DMA_INFIFO_EMPTY SPI DMA inFIFO is empty. (RO)

Register 24.32: SPI_DMA_IN_LINK_REG (0x0054)

SPI_DMA_RX_ENA	SPI_INLINK_RESTART	SPI_INLINK_START	SPI_INLINK_STOP	(reserved)	SPI_INLINK_AUTO_RET	SPI_INLINK_ADDR	
31	30	29	28	27 – 21	20	19 – 0	
0	0	0	0	0 0 0 0 0 0 0	0	0x000	Reset

SPI_INLINK_ADDR The address of the first inlink descriptor. (R/W)

SPI_INLINK_AUTO_RET When the bit is set, the inlink descriptor returns to the first link node when a packet is in error. (R/W)

SPI_INLINK_STOP Set the bit to stop using inlink descriptor. (R/W)

SPI_INLINK_START Set the bit to start using inlink descriptor. (R/W)

SPI_INLINK_RESTART Set the bit to mount on new inlink descriptors. (R/W)

SPI_DMA_RX_ENA 1: enable DMA controlled RX mode. 0: enable CPU controlled RX mode. (R/W) (R/W)

Register 24.33: SPI_DMA_INT_ENA_REG (0x0058)

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	SPI_SLV_CMDA_INT_ENA	SPI_SLV_CMD9_INT_ENA	SPI_SLV_CMD8_INT_ENA	SPI_SLV_CMD7_INT_ENA	SPI_SLV_CMD6_INT_ENA	SPI_OUTFIFO_EMPTY_ERR_INT_ENA	SPI_INFIFO_FULL_ERR_INT_ENA	SPI_OUT_TOTAL_EOF_INT_ENA	SPI_OUT_EOF_INT_ENA	SPI_OUT_DONE_INT_ENA	SPI_IN_SUC_EOF_INT_ENA	SPI_IN_ERR_EOF_INT_ENA	SPI_IN_DONE_INT_ENA	SPI_INLINK_DSCR_ERROR_INT_ENA	SPI_OUTLINK_DSCR_ERROR_INT_ENA	SPI_INLINK_DSCR_EMPTY_INT_ENA	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_EMPTY_INT_ENA The enable bit for not enough inlink descriptors. Can be configured in CONF state. (R/W)

SPI_OUTLINK_DSCR_ERROR_INT_ENA The enable bit for outlink descriptor error. Can be configured in CONF state. (R/W)

SPI_INLINK_DSCR_ERROR_INT_ENA The enable bit for inlink descriptor error. Can be configured in CONF state. (R/W)

SPI_IN_DONE_INT_ENA The enable bit for completing usage of an inlink descriptor. Can be configured in CONF state. (R/W)

SPI_IN_ERR_EOF_INT_ENA The enable bit for receiving error. Can be configured in CONF state. (R/W)

SPI_IN_SUC_EOF_INT_ENA The enable bit for completing receiving all the packets from host. Can be configured in CONF state. (R/W)

SPI_OUT_DONE_INT_ENA The enable bit for completing usage of an outlink descriptor. Can be configured in CONF state. (R/W)

SPI_OUT_EOF_INT_ENA The enable bit for sending a packet to host done. Can be configured in CONF state. (R/W)

SPI_OUT_TOTAL_EOF_INT_ENA The enable bit for sending all the packets to host done. Can be configured in CONF state. (R/W)

SPI_INFIFO_FULL_ERR_INT_ENA The enable bit for inFIFO full error interrupt. (R/W)

SPI_OUTFIFO_EMPTY_ERR_INT_ENA The enable bit for outFIFO empty error interrupt. (R/W)

SPI_SLV_CMD6_INT_ENA The enable bit for SPI slave CMD6 interrupt. (R/W)

SPI_SLV_CMD7_INT_ENA The enable bit for SPI slave CMD7 interrupt. (R/W)

SPI_SLV_CMD8_INT_ENA The enable bit for SPI slave CMD8 interrupt. (R/W)

SPI_SLV_CMD9_INT_ENA The enable bit for SPI slave CMD9 interrupt. (R/W)

SPI_SLV_CMDA_INT_ENA The enable bit for SPI slave CMDA interrupt. (R/W)

Register 24.34: SPI_DMA_INT_RAW_REG (0x005C)

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	SPI_SLV_CMDA_INT_RAW	SPI_SLV_CMD9_INT_RAW	SPI_SLV_CMD8_INT_RAW	SPI_SLV_CMD7_INT_RAW	SPI_SLV_CMD6_INT_RAW	SPI_OUTFIFO_EMPTY_ERR_INT_RAW	SPI_INFIFO_FULL_ERR_INT_RAW	SPI_OUT_TOTAL_EOF_INT_RAW	SPI_OUT_EOF_INT_RAW	SPI_OUT_DONE_INT_RAW	SPI_IN_SUC_EOF_INT_RAW	SPI_IN_ERR_EOF_INT_RAW	SPI_IN_DONE_INT_RAW	SPI_INLINK_DSCR_ERROR_INT_RAW	SPI_OUTLINK_DSCR_ERROR_INT_RAW	SPI_INLINK_DSCR_EMPTY_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_EMPTY_INT_RAW The raw bit for not enough inlink descriptors. Can be configured in CONF state. (RO)

SPI_OUTLINK_DSCR_ERROR_INT_RAW The raw bit for outlink descriptor error. Can be configured in CONF state. (RO)

SPI_INLINK_DSCR_ERROR_INT_RAW The raw bit for inlink descriptor error. Can be configured in CONF state. (RO)

SPI_IN_DONE_INT_RAW The raw bit for completing usage of an inlink descriptor. Can be configured in CONF state. (RO)

SPI_IN_ERR_EOF_INT_RAW The raw bit for receiving error. Can be configured in CONF state. (RO)

SPI_IN_SUC_EOF_INT_RAW The raw bit for completing receiving all the packets from host. Can be configured in CONF state. (RO)

SPI_OUT_DONE_INT_RAW The raw bit for completing usage of an outlink descriptor. Can be configured in CONF state. (RO)

SPI_OUT_EOF_INT_RAW The raw bit for sending a packet to host done. Can be configured in CONF state. (RO)

SPI_OUT_TOTAL_EOF_INT_RAW The raw bit for sending all the packets to host done. Can be configured in CONF state. (RO)

SPI_INFIFO_FULL_ERR_INT_RAW If the size of the DMA RX buffer is smaller than the size of the transferred data, the interrupt SPI_INFIFO_FULL_ERR_INT will be triggered. Can not be configured in CONF phase. (RO)

SPI_OUTFIFO_EMPTY_ERR_INT_RAW DMA TX buffer CONF If the size of the DMA TX buffer is smaller than the size of the transferred data, the interrupt SPI_OUTFIFO_EMPTY_ERR_INT will be triggered. Can not be configured in CONF phase. (RO)

SPI_SLV_CMD6_INT_RAW The raw bit for SPI slave CMD6 interrupt. (R/W)

SPI_SLV_CMD7_INT_RAW The raw bit for SPI slave CMD7 interrupt. (R/W)

SPI_SLV_CMD8_INT_RAW The raw bit for SPI slave CMD8 interrupt. (R/W)

Continued on the next page...

Register 24.34: SPI_DMA_INT_RAW_REG (0x005C)

Continued from the previous page...

SPI_SLV_CMD9_INT_RAW The raw bit for SPI slave CMD9 interrupt. (R/W)

SPI_SLV_CMDA_INT_RAW The raw bit for SPI slave CMDA interrupt. (R/W)

Register 24.35: SPI_DMA_INT_ST_REG (0x0060)

(reserved)	SPI_SLV_CMDA_INT_ST	SPI_SLV_CMD9_INT_ST	SPI_SLV_CMD8_INT_ST	SPI_SLV_CMD7_INT_ST	SPI_SLV_CMD6_INT_ST	SPI_OUTFIFO_EMPTY_ERR_INT_ST	SPI_INFIFO_FULL_ERR_INT_ST	SPI_OUT_TOTAL_EOF_INT_ST	SPI_OUT_EOF_INT_ST	SPI_OUT_DONE_INT_ST	SPI_IN_SUC_EOF_INT_ST	SPI_IN_ERR_EOF_INT_ST	SPI_IN_DONE_INT_ST	SPI_INLINK_DSCR_ERROR_INT_ST	SPI_OUTLINK_DSCR_ERROR_INT_ST	SPI_INLINK_DSCR_EMPTY_INT_ST	
31 ... 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_EMPTY_INT_ST The status bit for not enough inlink descriptors. (RO)

SPI_OUTLINK_DSCR_ERROR_INT_ST The status bit for outlink descriptor error. (RO)

SPI_INLINK_DSCR_ERROR_INT_ST The status bit for inlink descriptor error. (RO)

SPI_IN_DONE_INT_ST The status bit for completing usage of a inlink descriptor. (RO)

SPI_IN_ERR_EOF_INT_ST The status bit for receiving error. (RO)

SPI_IN_SUC_EOF_INT_ST The status bit for completing receiving all the packets from host. (RO)

SPI_OUT_DONE_INT_ST The status bit for completing usage of a outlink descriptor. (RO)

SPI_OUT_EOF_INT_ST The status bit for sending a packet to host done. (RO)

SPI_OUT_TOTAL_EOF_INT_ST The status bit for sending all the packets to host done. (RO)

SPI_INFIFO_FULL_ERR_INT_ST The status bit for inFIFO full error. (RO)

SPI_OUTFIFO_EMPTY_ERR_INT_ST The status bit for outFIFO empty error. (RO)

SPI_SLV_CMD6_INT_ST The status bit for SPI slave CMD6 interrupt. (R/W)

SPI_SLV_CMD7_INT_ST The status bit for SPI slave CMD7 interrupt. (R/W)

SPI_SLV_CMD8_INT_ST The status bit for SPI slave CMD8 interrupt. (R/W)

SPI_SLV_CMD9_INT_ST The status bit for SPI slave CMD9 interrupt. (R/W)

SPI_SLV_CMDA_INT_ST The status bit for SPI slave CMDA interrupt. (R/W)

Register 24.36: SPI_DMA_INT_CLR_REG (0x0064)

Bits	31..16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field	(reserved)	SPI_SLV_CMDA_INT_CLR	SPI_SLV_CMD9_INT_CLR	SPI_SLV_CMD8_INT_CLR	SPI_SLV_CMD7_INT_CLR	SPI_SLV_CMD6_INT_CLR	SPI_OUTFIFO_EMPTY_ERR_INT_CLR	SPI_INFIFO_FULL_ERR_INT_CLR	SPI_OUT_TOTAL_EOF_INT_CLR	SPI_OUT_EOF_INT_CLR	SPI_OUT_DONE_INT_CLR	SPI_IN_SUC_EOF_INT_CLR	SPI_IN_ERR_EOF_INT_CLR	SPI_IN_DONE_INT_CLR	SPI_INLINK_DSCR_ERROR_INT_CLR	SPI_OUTLINK_DSCR_ERROR_INT_CLR	SPI_INLINK_DSCR_EMPTY_INT_CLR	
Value	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_EMPTY_INT_CLR The clear bit for lack of enough inlink descriptors. Can be configured in CONF state. (R/W)

SPI_OUTLINK_DSCR_ERROR_INT_CLR The clear bit for outlink descriptor error. Can be configured in CONF state. (R/W)

SPI_INLINK_DSCR_ERROR_INT_CLR The clear bit for inlink descriptor error. Can be configured in CONF state. (R/W)

SPI_IN_DONE_INT_CLR The clear bit for completing usage of a inlink descriptor. Can be configured in CONF state. (R/W)

SPI_IN_ERR_EOF_INT_CLR The clear bit for receiving error. Can be configured in CONF state. (R/W)

SPI_IN_SUC_EOF_INT_CLR The clear bit for completing receiving all the packets from host. Can be configured in CONF state. (R/W)

SPI_OUT_DONE_INT_CLR The clear bit for completing usage of a outlink descriptor. Can be configured in CONF state. (R/W)

SPI_OUT_EOF_INT_CLR The clear bit for sending a packet to host done. Can be configured in CONF state. (R/W)

SPI_OUT_TOTAL_EOF_INT_CLR The clear bit for sending all the packets to host done. Can be configured in CONF state. (R/W)

SPI_INFIFO_FULL_ERR_INT_CLR 1: Clear SPI_INFIFO_FULL_ERR_INT_RAW. 0: not valid. Can be changed by CONF_buf. (R/W)

SPI_OUTFIFO_EMPTY_ERR_INT_CLR 1: Clear SPI_OUTFIFO_EMPTY_ERR_INT_RAW signal. 0: not valid. Can be changed by CONF_buf. (R/W)

SPI_SLV_CMD6_INT_CLR The clear bit for SPI slave CMD6 interrupt. (R/W)

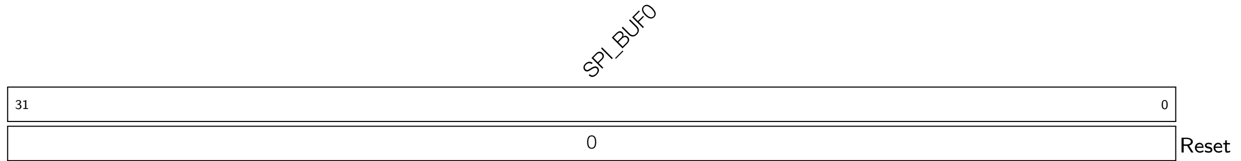
SPI_SLV_CMD7_INT_CLR The clear bit for SPI slave CMD7 interrupt. (R/W)

SPI_SLV_CMD8_INT_CLR The clear bit for SPI slave CMD8 interrupt. (R/W)

SPI_SLV_CMD9_INT_CLR The clear bit for SPI slave CMD9 interrupt. (R/W)

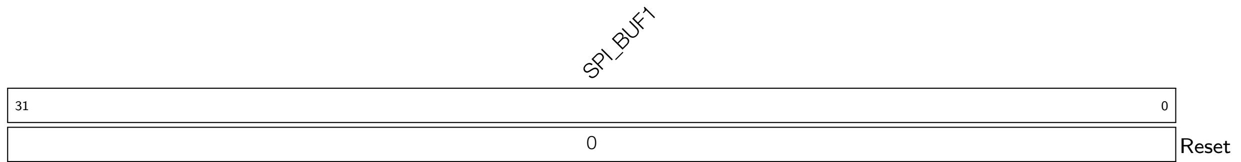
SPI_SLV_CMDA_INT_CLR The clear bit for SPI slave CMDA interrupt. (R/W)

Register 24.37: SPI_W0_REG (0x0098)



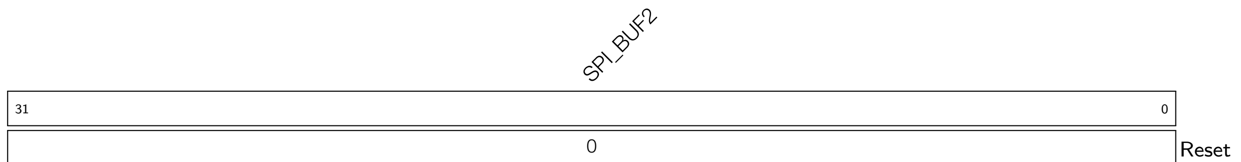
SPI_BUF0 32 bits data buffer 0, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.38: SPI_W1_REG (0x009C)



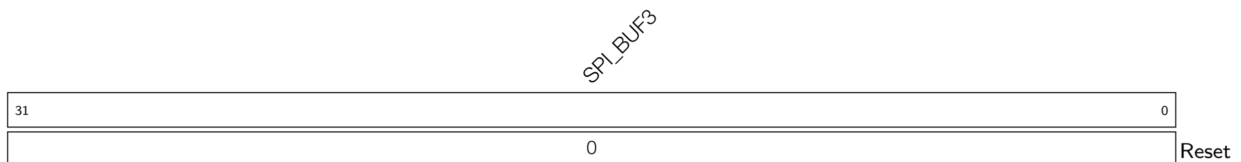
SPI_BUF1 32 bits data buffer 1, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.39: SPI_W2_REG (0x00A0)



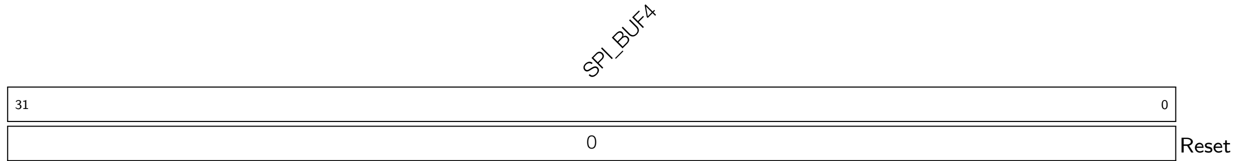
SPI_BUF2 32 bits data buffer 2, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.40: SPI_W3_REG (0x00A4)



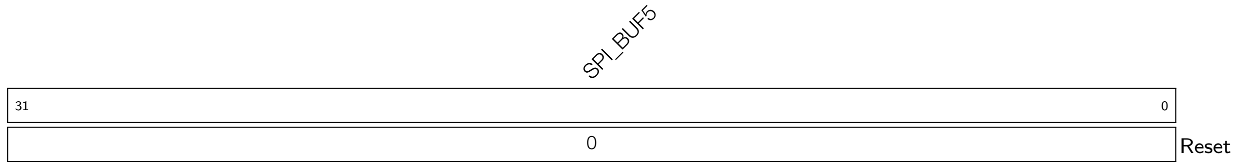
SPI_BUF3 32 bits data buffer 3, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.41: SPI_W4_REG (0x00A8)



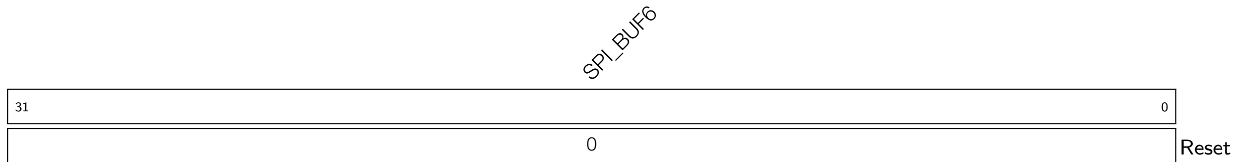
SPI_BUF4 32 bits data buffer 4, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.42: SPI_W5_REG (0x00AC)



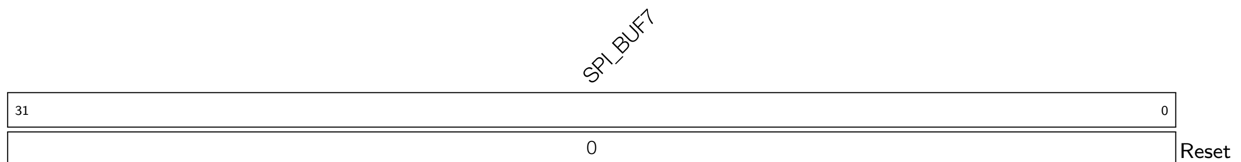
SPI_BUF5 32 bits data buffer 5, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.43: SPI_W6_REG (0x00B0)



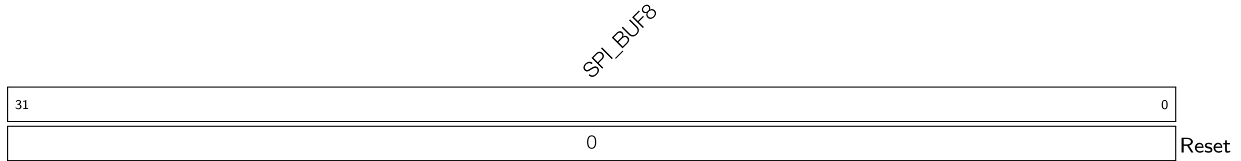
SPI_BUF6 32 bits data buffer 6, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.44: SPI_W7_REG (0x00B4)



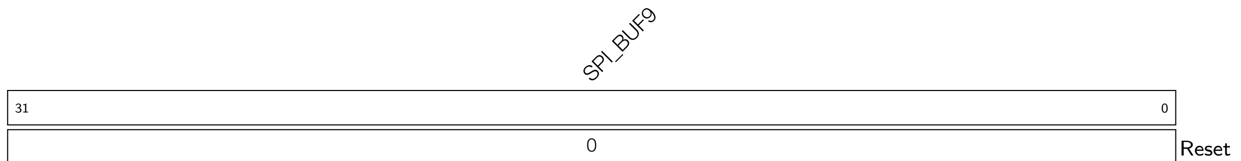
SPI_BUF7 32 bits data buffer 7, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.45: SPI_W8_REG (0x00B8)



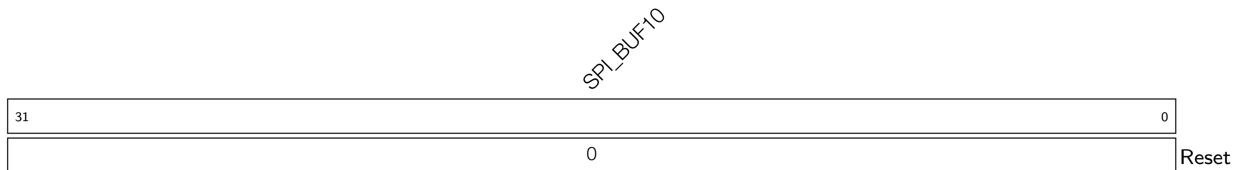
SPI_BUF8 32 bits data buffer 8, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.46: SPI_W9_REG (0x00BC)



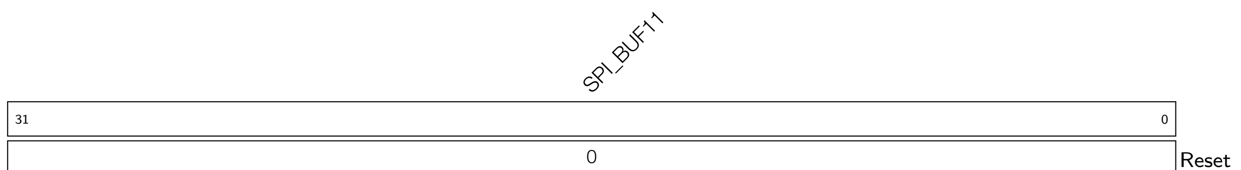
SPI_BUF9 32 bits data buffer 9, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.47: SPI_W10_REG (0x00C0)



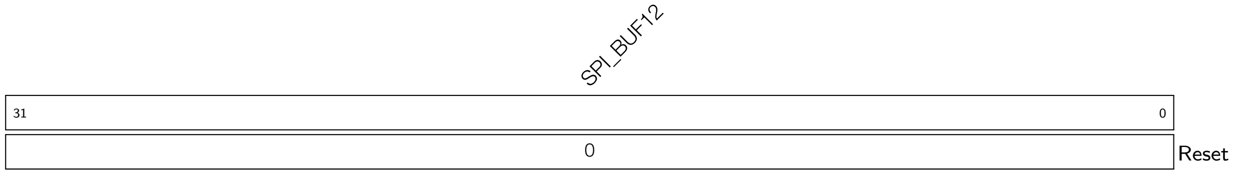
SPI_BUF10 32 bits data buffer 10, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.48: SPI_W11_REG (0x00C4)



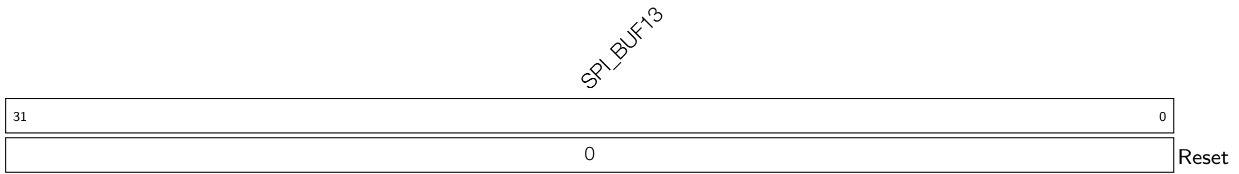
SPI_BUF11 32 bits data buffer 11, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.49: SPI_W12_REG (0x00C8)



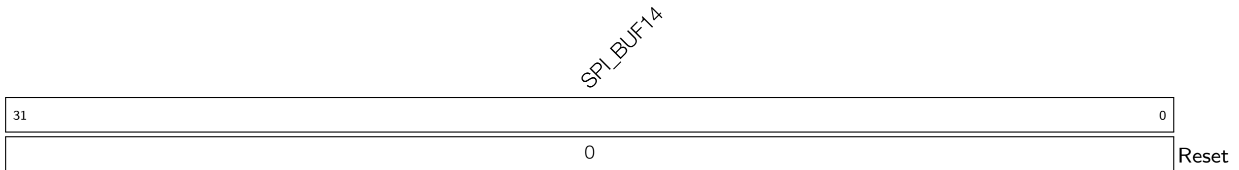
SPI_BUF12 32 bits data buffer 12, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.50: SPI_W13_REG (0x00CC)



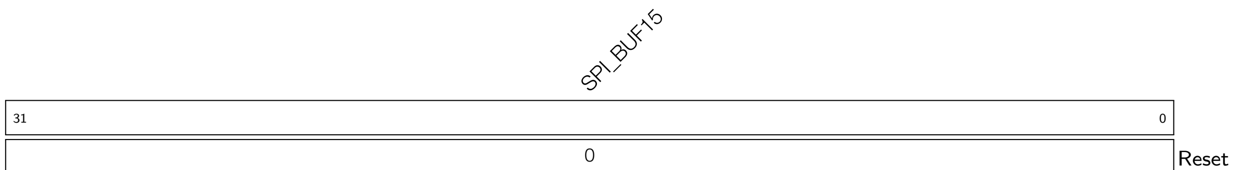
SPI_BUF13 32 bits data buffer 13, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.51: SPI_W14_REG (0x00D0)



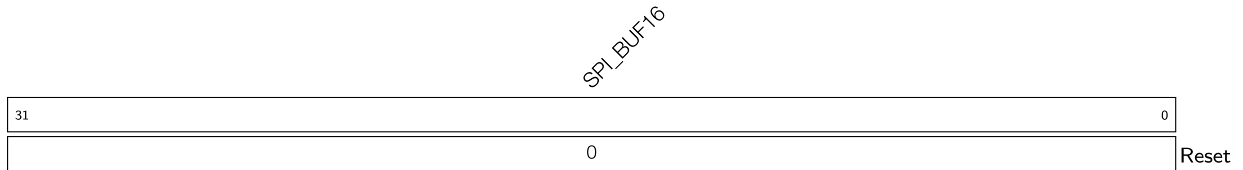
SPI_BUF14 32 bits data buffer 14, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.52: SPI_W15_REG (0x00D4)



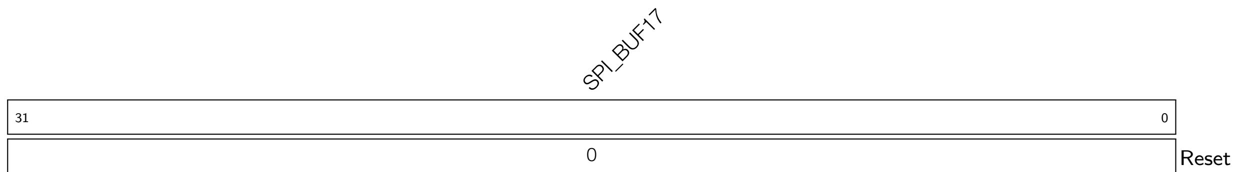
SPI_BUF15 32 bits data buffer 15, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.53: SPI_W16_REG (0x00D8)



SPI_BUF16 32 bits data buffer 16, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.54: SPI_W17_REG (0x00DC)



SPI_BUF17 32 bits data buffer 17, transferred in the unit of byte. Byte addressable in slave half-duplex mode. (R/W)

Register 24.55: SPI_DIN_MODE_REG (0x00E0)

(reserved)	SPI_TIMING_CLK_ENA	SPI_DIN7_MODE	SPI_DIN6_MODE	SPI_DIN5_MODE	SPI_DIN4_MODE	SPI_DIN3_MODE	SPI_DIN2_MODE	SPI_DIN1_MODE	SPI_DIN0_MODE	
31 25	24	23 21	20 18	17 15	14 12	11 9	8 6	5 3	2 0	
0 0 0 0 0 0 0	0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_DIN0_MODE Select clock source, and sample input signal FSPID from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN1_MODE Select clock source, and sample input signal FSPIQ from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN2_MODE Select clock source, and sample input signal FSPIWP from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN3_MODE Select clock source, and sample input signal FSPIHD from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN4_MODE Select clock source, and sample input signal FSPIIO4 from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN5_MODE Select clock source, and sample input signal FSPIIO5 from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN6_MODE Select clock source, and sample input signal FSPIIO6 from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DIN7_MODE Select clock source, and sample input signal FSPIIO7 from the master with delays. 0: input without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_TIMING_CLK_ENA 1: enable high-frequency clock: HCLK 160 MHz. 0: disable. Can be configured in CONF state. (R/W)

Register 24.56: SPI_DIN_NUM_REG (0x00E4)

(reserved)	SPI_DIN7_NUM	SPI_DIN6_NUM	SPI_DIN5_NUM	SPI_DIN4_NUM	SPI_DIN3_NUM	SPI_DIN2_NUM	SPI_DIN1_NUM	SPI_DIN0_NUM	
31 ... 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_DIN0_NUM Configure the delays to input signal FSPID based on the setting of SPI_DIN0_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN1_NUM The Configure the delays to input signal FSPIQ based on the setting of SPI_DIN1_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN2_NUM Configure the delays to input signal FSPIWP based on the setting of SPI_DIN2_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN3_NUM Configure the delays to input signal FSPIHD based on the setting of SPI_DIN3_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN4_NUM Configure the delays to input signal FSPIIO4 based on the setting of SPI_DIN4_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN5_NUM Configure the delays to input signal FSPIIO5 based on the setting of SPI_DIN5_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN6_NUM Configure the delays to input signal FSPIIO6 based on the setting of SPI_DIN6_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DIN7_NUM Configure the delays to input signal FSPIIO7 based on the setting of SPI_DIN7_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

Register 24.57: SPI_DOUT_MODE_REG (0x00E8)

(reserved)	SPI_DOUT7_MODE	SPI_DOUT6_MODE	SPI_DOUT5_MODE	SPI_DOUT4_MODE	SPI_DOUT3_MODE	SPI_DOUT2_MODE	SPI_DOUT1_MODE	SPI_DOUT0_MODE	
31 24	23 21	20 18	17 15	14 12	11 9	8 6	5 3	2 0	
0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_DOUT0_MODE Select clock source, and sample output signal FSPID from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT1_MODE Select clock source, and sample output signal FSPIQ from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT2_MODE Select clock source, and sample output signal FSPIWP from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT3_MODE Select clock source, and sample output signal FSPIHD from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT4_MODE Select clock source, and sample output signal FSPIIO4 from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT5_MODE Select clock source, and sample output signal FSPIIO5 from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT6_MODE Select clock source, and sample output signal FSPIIO6 from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DOUT7_MODE Select clock source, and sample output signal FSPIIO7 from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

Register 24.58: SPI_DOUT_NUM_REG (0x00EC)

(reserved)	SPI_DOUT7_NUM	SPI_DOUT6_NUM	SPI_DOUT5_NUM	SPI_DOUT4_NUM	SPI_DOUT3_NUM	SPI_DOUT2_NUM	SPI_DOUT1_NUM	SPI_DOUT0_NUM	
31 … 16	15 … 14	13 … 12	11 … 10	9 … 8	7 … 6	5 … 4	3 … 2	1 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_DOUT0_NUM Configure the delays to output signal FSPID based on the setting of SPI_DOUT0_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT1_NUM Configure the delays to output signal FSPIQ based on the setting of SPI_DOUT1_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT2_NUM Configure the delays to output signal FSPIWP based on the setting of SPI_DOUT2_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT3_NUM Configure the delays to output signal FSPIHD based on the setting of SPI_DOUT3_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT4_NUM Configure the delays to output signal FSPIIO4 based on the setting of SPI_DOUT4_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT5_NUM Configure the delays to output signal FSPIIO5 based on the setting of SPI_DOUT5_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT6_NUM Configure the delays to output signal FSPIIO6 based on the setting of SPI_DOUT6_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_DOUT7_NUM Configure the delays to output signal FSPIIO7 based on the setting of SPI_DOUT7_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

Register 24.59: SPI_LCD_CTRL_REG (0x00F0)

SPI_LCD_MODE_EN	SPI_LCD_VT_HEIGHT	SPI_LCD_VA_HEIGHT	SPI_LCD_HB_FRONT	
31	30 – 21	20 – 11	10 – 0	
0	0	0	0	Reset

SPI_LCD_HB_FRONT It is the horizontal blank front porch of a frame. Can be configured in CONF state. (R/W)

SPI_LCD_VA_HEIGHT It is the vertical active height of a frame. Can be configured in CONF state. (R/W)

SPI_LCD_VT_HEIGHT It is the vertical total height of a frame. Can be configured in CONF state. (R/W)

SPI_LCD_MODE_EN 1: Enable LCD mode output vsync, hsync, de. 0: Disable. Can be configured in CONF state. (R/W)

Register 24.60: SPI_LCD_CTRL1_REG (0x00F4)

SPI_LCD_HT_WIDTH	SPI_LCD_HA_WIDTH	SPI_LCD_VB_FRONT	
31 – 20	19 – 8	7 – 0	
0	0	0	Reset

SPI_LCD_VB_FRONT It is the vertical blank front porch of a frame. Can be configured in CONF state. (R/W)

SPI_LCD_HA_WIDTH It is the horizontal active width of a frame. Can be configured in CONF state. (R/W)

SPI_LCD_HT_WIDTH It is the horizontal total width of a frame. Can be configured in CONF state. (R/W)

Register 24.61: SPI_LCD_CTRL2_REG (0x00F8)

SPI_LCD_HSYNC_POSITION	SPI_HSYNC_IDLE_POL	SPI_LCD_HSYNC_WIDTH	(reserved)	SPI_VSYNC_IDLE_POL	SPI_LCD_VSYNC_WIDTH	
31 24	23	22 16	15 8	7	6 0	
0	0	1	0 0 0 0 0 0 0 0	0	1	Reset

SPI_LCD_VSYNC_WIDTH It is the position of spi_vsync active pulse in a line. Can be configured in CONF state. (R/W)

SPI_VSYNC_IDLE_POL It is the idle value of spi_vsync. Can be configured in CONF state. (R/W)

SPI_LCD_HSYNC_WIDTH It is the position of spi_hsync active pulse in a line. Can be configured in CONF state. (R/W)

SPI_HSYNC_IDLE_POL It is the idle value of spi_hsync. Can be configured in CONF state. (R/W)

SPI_LCD_HSYNC_POSITION It is the position of spi_hsync active pulse in a line. Can be configured in CONF state. (R/W)

Register 24.62: SPI_LCD_D_MODE_REG (0x00FC)

(reserved)	SPI_HS_BLANK_EN	SPI_DE_IDLE_POL	SPI_D_VSYNC_MODE	SPI_D_HSYNC_MODE	SPI_D_DE_MODE	SPI_D_CD_MODE	SPI_D_DQS_MODE	
31 … 17	16	15	14 … 12	11 … 9	8 … 6	5 … 3	2 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_D_DQS_MODE Select clock source, and sample output signal FSPIDQS from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_D_CD_MODE Select clock source, and sample output signal FSPICD from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_D_DE_MODE Select clock source, and sample output signal FSPI_DE from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_D_HSYNC_MODE SPI_HSYNC output mode. 0: output Select clock source, and sample output signal FSPI_HSYNC from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_D_VSYNC_MODE Select clock source, and sample output signal FSPI_VSYNC from the master with delays. 0: output without delay. 1: sample at the rising edge of APB_CLK. 2: sample at the falling edge of APB_CLK. 3: sample at the rising edge of HCLK. 4: sample at the falling edge of HCLK. Can be configured in CONF state. (R/W)

SPI_DE_IDLE_POL It is the idle value of SPI_DE. (R/W)

SPI_HS_BLANK_EN 1: The pulse of SPI_HSYNC is out in vertical blanking lines in seg-trans or one trans. 0: SPI_HSYNC pulse is valid only in active region lines in segment transfer. (R/W)

Register 24.63: SPI_LCD_D_NUM_REG (0x0100)

31 ... 10	9 8	7 6	5 4	3 2	1 0	
(reserved)	SPI_D_VSYNC_NUM	SPI_D_HSYNC_NUM	SPI_D_DE_NUM	SPI_D_CD_NUM	SPI_D_DQS_NUM	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	0x0	0x0	0x0	Reset

SPI_D_DQS_NUM Configure the delays to output signal FSPIDQS based on the setting of SPI_D_DQS_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_D_CD_NUM Configure the delays to output signal FSPI_CD based on the setting of SPI_D_CD_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_D_DE_NUM Configure the delays to output signal FSPI_DE based on the setting of SPI_D_DE_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_D_HSYNC_NUM Configure the delays to output signal FSPI_HSYNC based on the setting of SPI_D_HSYNC_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

SPI_D_VSYNC_NUM Configure the delays to output signal FSPI_VSYNC based on the setting of SPI_D_VSYNC_MODE. 0: delayed by 1 clock cycle, 1: delayed by 2 clock cycles,... Can be configured in CONF state. (R/W)

Register 24.64: SPI_DATE_REG (0x03FC)

31 ... 28	27 ... 0	
(reserved)	SPI_DATE	
0 0 0 0	0x1907240	Reset

SPI_DATE Version control register (R/W)

25. I2C Controller (I2C)

25.1 Overview

The I2C (Inter-Integrated Circuit) controller allows ESP32-S2 to communicate with multiple peripheral devices. These peripheral devices can share one bus.

25.2 Features

The I2C controller has the following features:

- Master mode and slave mode
- Multi-master and multi-slave communication
- Standard mode (100 kbit/s)
- Fast mode (400 kbit/s)
- 7-bit addressing and 10-bit addressing
- Continuous data transfer in master mode achieved by pulling SCL low
- Programmable digital noise filtering
- Double addressing mode

25.3 I2C Functional Description

25.3.1 I2C Introduction

The I2C bus has two lines, namely a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I2C bus is connected to multiple devices, usually a single or multiple masters and a single or multiple slaves. However, only one master device can access a slave at a time.

The master initiates communication by generating a start condition: pulling the SDA line low while SCL is high, and sending nine clock pulses via SCL. The first eight pulses are used to transmits a byte, which consists of a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of a slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master and the slave can send or receive data according to the $R/\overline{W}$ bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once finishing communication, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send a RESTART condition, a slave address and a $R/\overline{W}$ bit before changing its operation.

25.3.2 I2C Architecture

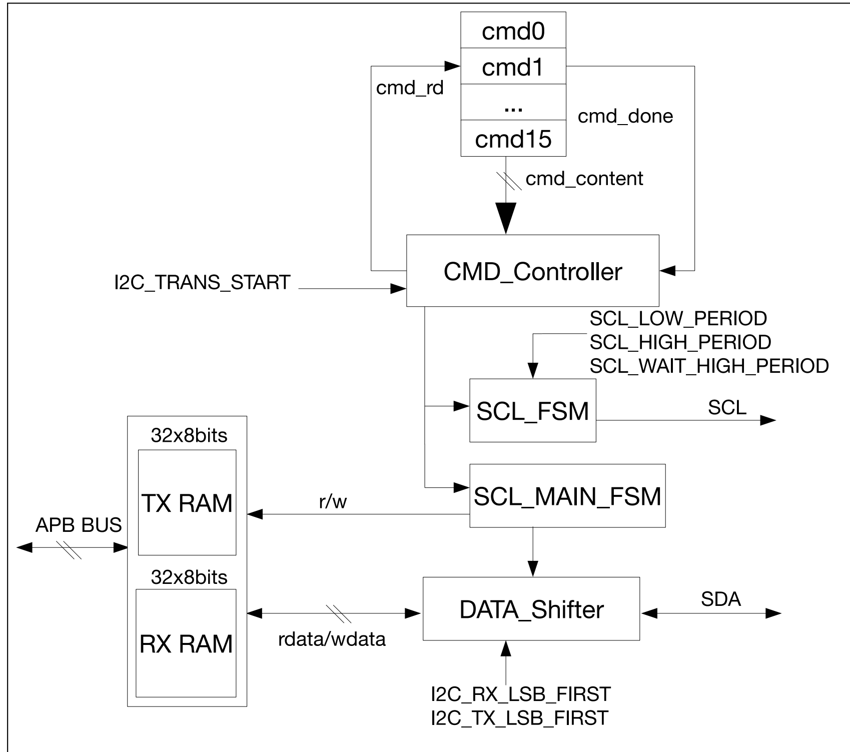


Figure 25-1. I2C Master Architecture

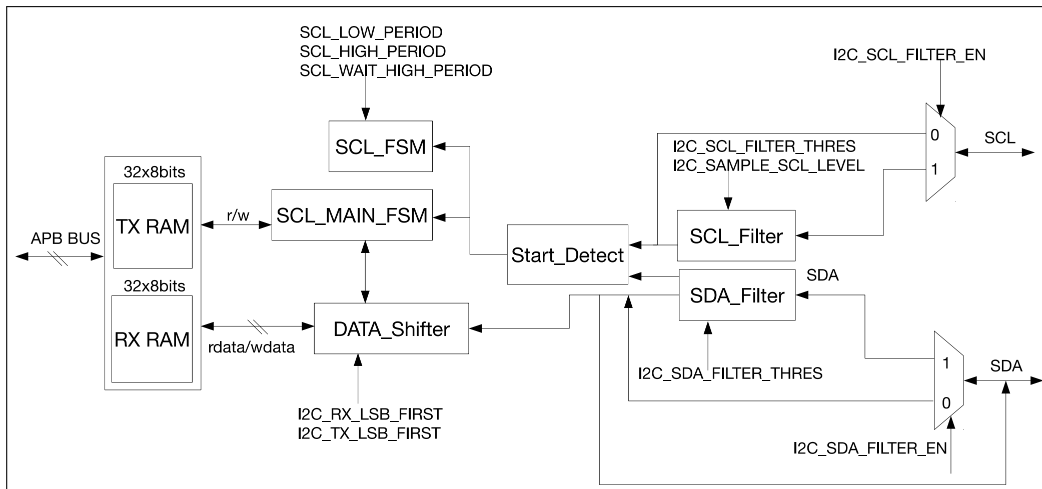


Figure 25-2. I2C Slave Architecture

The I2C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE bit. Figure 25-1 shows the architecture of a master, while Figure 25-2 shows that of a slave. The I2C controller has the following main parts: transmit and receive memory (TX/RX RAM), command controller (CMD_Controller), SCL clock controller (SCL_FSM), SDA data controller (SCL_MAIN_FSM), serial-to-parallel data converter (DATA_Shifter), filter for SCL (SCL_Filter) and filter for SDA (SDA_Filter).

25.3.2.1 TX/RX RAM

Both TX RAM and RX RAM are 32 × 8 bits. TX RAM stores data that the I2C controller needs to send. During communication, when the I2C controller needs to send data (except acknowledgement bits), it reads data from TX RAM and sends it sequentially via SDA. When the I2C controller works in master mode, all data must be stored in TX RAM in the order they will be sent to slaves. The data stored in TX RAM include slave addresses, read/write bits, register addresses (only in double addressing mode) and data to be sent. When the I2C controller works in slave mode, TX RAM only stores data to be sent.

RX RAM stores data the I2C controller receives during communication. When the I2C controller works in slave mode, neither slave addresses sent by the master nor register addresses (only in double addressing mode) will be stored into RX RAM. Values of RX RAM can be read by software after I2C communication completes.

Both TX RAM and RX RAM can be accessed in FIFO or non-FIFO mode. The I2C_NONFIFO_EN bit sets FIFO or non-FIFO mode.

TX RAM can be read and written by the CPU. The CPU writes to TX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU writes to TX RAM via fixed address I2C_DATA_REG, with addresses for writing in TX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x100) ~(I2C Base Address + 0x17C). Each byte in TX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base Address + 0x100, the second byte I2C Base Address + 0x104, the third byte I2C Base Address + 0x108, and so on. The CPU can only read TX RAM via direct addresses. Unlike addresses for writing, TX RAM must be read back from addresses starting at I2C Base Address + 0x80.

RX RAM can only be read by the CPU. The CPU reads RX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU reads RX RAM via fixed address I2C_DATA_REG, with addresses for reading RX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x100) ~(I2C Base Address + 0x17C). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base Address + 0x100, the second byte I2C Base Address + 0x104, the third byte I2C Base Address + 0x108 and so on.

Given that addresses for writing to TX RAM have an identical range with those for reading RX RAM, TX RAM and RX RAM can be seen as a 32 x 8 RAM. In following sections TX RAM and RX RAM are referred to as RAM.

25.3.2.2 CMD_Controller

When the I2C controller works in master mode, the integrated CMD_Controller module reads commands from 16 sequential command registers and controls SCL_FSM and SDA_FSM accordingly.

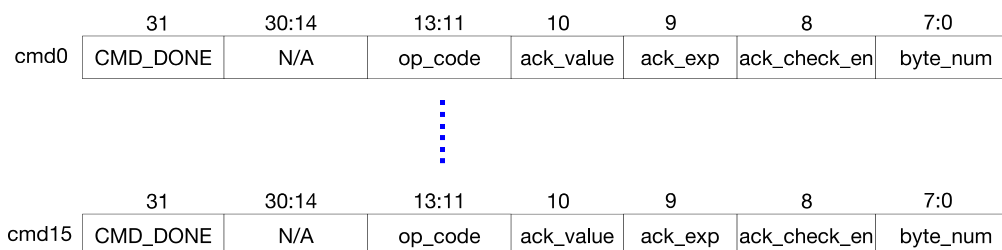


Figure 25-3. Structure of I2C Command Register

Command registers, whose structure is illustrated in Figure 25-3, are active only when the I2C controller works in

master mode. Fields of command registers are:

1. CMD_DONE: Indicates that a command has been executed. After each command has been executed, the corresponding CMD_DONE bit is set to 1 by hardware. By reading this bit, software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. op_code: Indicates the command. The I2C controller supports five commands:
   - RSTART: op_code = 0: The I2C controller sends a START bit and a RESTART bit defined by the I2C protocol.
   - WRITE: op_code = 1: The I2C controller sends a slave address, a register address (only in double addressing mode) and data to the slave.
   - READ: op_code = 2: The I2C controller reads data from the slave.
   - STOP: op_code = 3: The I2C controller sends a STOP bit defined by the I2C protocol. This code also indicates that the command sequence has been executed, and the CMD_Controller stops reading commands. After restarted by software, the CMD_Controller resumes reading commands from command register 0.
   - END: op_code = 4: The I2C controller pulls the SCL line down and suspends I2C communication. This code also indicates that the command sequence has completed, and the CMD_Controller stops executing commands. Once software refreshes data in command registers and the RAM, the CMD_Controller can be restarted to execute commands from command register 0 again.
3. ack_value: Used to configure the level of the ACK bit sent by the I2C controller during a read operation. This bit is ignored during RSTART, STOP, END and WRITE conditions.
4. ack_exp: Used to configure the level of the ACK bit expected by the I2C controller during a write operation. This bit is ignored during RSTART, STOP, END and READ conditions.
5. ack_check_en: Used to enable the I2C controller during a write operation to check whether ACK's level sent by the slave matches ack_exp in the command. If this bit is set and the level received does not match ack_exp in the WRITE command, the master will generate an I2C_NACK_INT interrupt and a STOP condition for data transfer. If this bit is cleared, the controller will not check the ACK level sent by the slave. This bit is ignored during RSTART, STOP, END and READ conditions.
6. byte_num: Specifies the length of data (in bytes) to be read or written. Can range from 1 to 255 bytes. This bit is ignored during RSTART, STOP and END conditions.

Each command sequence is executed starting from command register 0 and terminated by a STOP or an END. Therefore, all 16 command registers must have a STOP or an END command.

A complete data transfer on the I2C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, separated by END commands. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, data to be transmitted, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I2C communication.

25.3.2.3 SCL_FSM

The integrated SCL_FSM module controls the SCL clock line. The frequency and duty cycle of SCL is configured using I2C_SCL_LOW_PERIOD_REG, I2C_SCL_HIGH_PERIOD_REG and I2C_SCL_WAIT_HIGH_PERIOD. After

being in non-IDLE state for over I2C_SCL_ST_TO clock cycles, SCL_FSM triggers an I2C_SCL_ST_TO_INT interrupt and returns to IDLE state.

25.3.2.4 SCL_MAIN_FSM

The integrated SCL_MAIN_FSM module controls the SDA data line and data storage. After being in non-IDLE state for over I2C_SCL_MAIN_ST_TO clock cycles, SCL_MAIN_FSM triggers an I2C_SCL_MAIN_ST_TO_INT interrupt and returns to IDLE state.

25.3.2.5 DATA_Shifter

The integrated DATA_Shifter module is used for serial/parallel conversion, converting TX RAM byte data to an outgoing serial bitstream or an incoming serial bitstream to RX RAM byte data. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used to select LSB- or MSB-first storage and transmission of data.

25.3.2.6 SCL_Filter and SDA_Filter

The integrated SCL_Filter and SDA_Filter modules are identical and are used to filter signal noises on SCL and SDA, respectively. These filters can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN.

SCL_Filter samples input signals on the SCL line continuously. These input signals are valid only if they remain unchanged for consecutive I2C_SCL_FILTER_THRES clock cycles. Given that only valid input signals can pass through the filter, SCL_Filter can remove glitches whose pulse width is lower than I2C_SCL_FILTER_THRES APB clock cycles.

SDA_Filter is identical to SCL_Filter, only applied to the SDA line. The threshold pulse width is provided in the I2C_SDA_FILTER_THRES register.

25.3.3 I2C Bus Timing

The I2C controller may use APB_CLK or REF_TICK as its clock source. When I2C_REF_ALWAYS_ON is 1, APB_CLK is used; when I2C_REF_ALWAYS_ON is 0, REF_TICK is used.

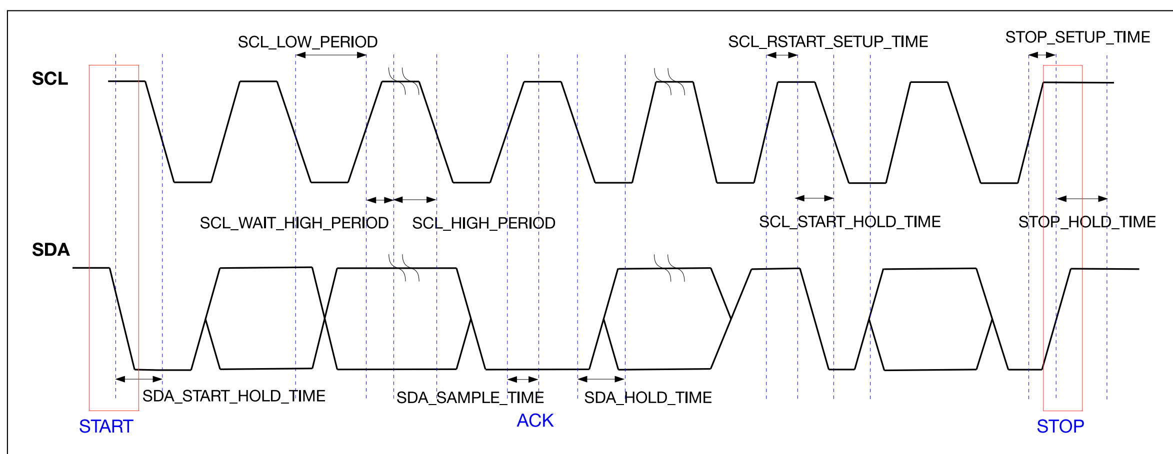


Figure 25-4. I2C Timing Diagram

Figure 25-4 shows the timing diagram of an I2C master. The unit of parameters in this figure is I2C_CLK (T_{I2C_CLK}). Specifically, when I2C_REF_ALWAYS_ON = 1, T_{I2C_CLK} is T_{APB_CLK}; when

I2C_REF_ALWAYS_ON = 0, T_{I2C_CLK} is T_{REF_TICK}. Figure 25-4 also specifies registers used to configure the START bit, STOP bit, data hold time, data sample time, waiting time on the rising SCL edge, etc. Parameters in Figure 25-4 are described as the following:

1. I2C_SCL_START_HOLD_TIME: Specifies the interval between pulling SDA low and pulling SCL low when the master generates a START condition. This interval is (I2C_SCL_START_HOLD_TIME +1) × T_{I2C_CLK}. This register is active only when the I2C controller works in master mode.
2. I2C_SCL_LOW_PERIOD: Specifies the low period of SCL. This period lasts (I2C_SCL_START_HOLD_TIME +1) × T_{I2C_CLK}. However, it could be extended when SCL is pulled low by peripheral devices or by an END command executed by the I2C controller, or when the clock is stretched. This register is active only when the I2C controller works in master mode.
3. I2C_SCL_WAIT_HIGH_PERIOD: Specifies time for SCL to go high in T_{I2C_CLK}. Please make sure that SCL could be pulled high within this time period. Otherwise, the high period of SCL may be incorrect. This register is active only when the I2C controller works in master mode.
4. I2C_SCL_HIGH_PERIOD: Specifies the high period of SCL in T_{I2C_CLK}. This register is active only when the I2C controller works in master mode. When SCL goes high within (I2C_SCL_WAIT_HIGH_PERIOD + 1) × T_{I2C_CLK}, its frequency is:

$$f_{scl} = \frac{f_{\text{I2C_CLK}}}{\text{I2C_SCL_LOW_PERIOD} + 1 + \text{I2C_SCL_HIGH_PERIOD} + \text{I2C_SCL_WAIT_HIGH_PERIOD}}$$

5. I2C_SDA_SAMPLE_TIME: Specifies the interval between the rising edge of SCL and the level sampling time of SDA. It is advised to set a value in the middle of SCL's high period. This register is active both in master mode and slave mode.
6. I2C_SDA_HOLD_TIME: Specifies the interval between changing the SDA output level and the falling edge of SCL. This register is active both in master mode and slave mode.

SCL and SDA output drivers must be configured as open drain. There are two ways to achieve this:

1. Set I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT, and configure GPIO_PIN*n*_PAD_DRIVER for corresponding SCL and SDA pads as open-drain.
2. Clear I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT.

Because these lines are configured as open-drain, the low to high transition time of each line is determined together by the pull-up resistance and the capacitance on the line. The output frequency of I2C is limited by the SDA and SCL line's pull-up speed, mainly SCL's speed.

In addition, when I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN are set to 1, SCL can be forced low; when I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN are set to 1, SDA can be forced low.

25.4 Typical Applications

For the convenience of description, I2C masters and slaves in all subsequent figures refer to ESP32-S2 I2C peripheral controllers. However, these controllers can communicate with any other I2C devices.

25.4.1 An I2C Master Writes to an I2C Slave with a 7-bit Address in One Command Sequence

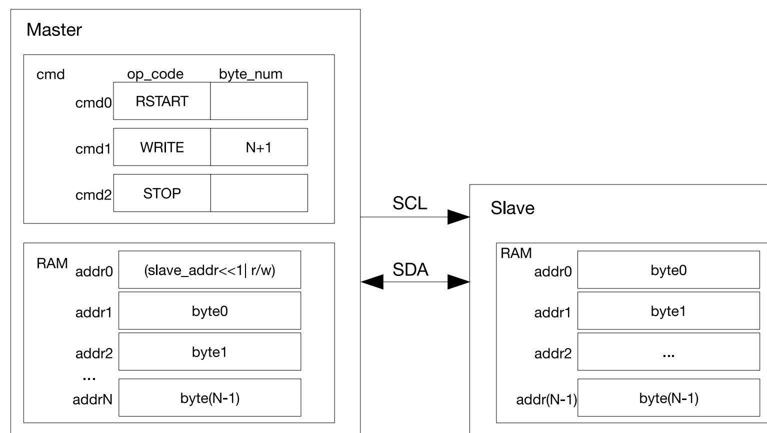


Figure 25-5. An I2C Master Writing to an I2C Slave with a 7-bit Address

Figure 25-5 shows how an I2C master writes N bytes of data using 7-bit addressing. As shown in figure 25-5, the first byte in the master's RAM is a 7-bit slave address followed by a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is zero, it indicates a WRITE operation. The remaining bytes are used to store data ready for transfer. The cmd box contains related command sequences.

After the command sequence is configured and data in RAM is ready, the master enables the controller and initiates data transfer by setting I2C_TRANS_START bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL used by other masters or slaves.
2. Execute a RSTART command and send a START bit.
3. Execute a WRITE command by taking N+1 bytes from the RAM in order and send them to the slave in the same order. The first byte is the address of the slave.
4. Send a STOP. Once the I2C master transfers a STOP bit, an I2C_TRANS_COMPLETE_INT interrupt is generated.

If data to be transferred is larger than 32 bytes, the RAM access will wrap around. While the controller sends data, software must replace data already sent in RAM. To assist with this process, the master will generate an I2C_TXFIFO_WM_INT interrupt when less than I2C_TXFIFO_WM_THRHD bytes of data remain to be sent.

After detecting this interrupt, software can refresh data in RAM. When the RAM is accessed in non-FIFO mode, in order to overwrite existing data in the RAM with new ones, software needs to first configure I2C_TX_UPDATE bit to latch the start address and the end address of data sent in the RAM, and then read I2C_TXFIFO_START_ADDR and I2C_TXFIFO_END_ADDR field in I2C_FIFO_ST_REG register to obtain these addresses. When the RAM is accessed in FIFO mode, new data can be written to the RAM directly via I2C_DATA_REG register.

The controller stops executing the command sequence after a STOP command, or when one of the following two events occurs:

1. When ack_check_en is set to 1, the I2C master checks the ACK value each time it sends a data byte. If the ACK value received does not match ack_exp (the expected ACK value) in the WRITE command, the master generates an I2C_NACK_INT interrupt and stops the transmission.

2. During the high period of SCL, if the input value and the output value of SDA do not match, the I2C master generates an I2C_ARBITRATION_LOST_INT interrupt, stops executing the command sequence, returns to IDLE state and releases SCL and SDA.

Once detecting a START bit sent by the I2C master, the I2C slave receives the address and compares it with its own address. If the received address does not match I2C_SLAVE_ADDR[6:0], the slave stops receiving data. If the received address does match I2C_SLAVE_ADDR[6:0], the slave receives data and stores them into the RAM in order.

If data to be transferred is larger than 32 bytes, the RAM may wrap around. While the controller receives data, software reclaim data already received by the slave. To assist with this process, the master will generate an I2C_RXFIFO_WM_INT after I2C_RXFIFO_WM_THRHD bytes are received in RAM.

Once detecting this interrupt, software can read data from the RAM registers. When the RAM is accessed in non-FIFO mode, in order to read data, software needs to configure I2C_RX_UPDATE bit to latch the start address and the end address of data to be reclaimed, and read I2C_RXFIFO_START_ADDR and I2C_RXFIFO_END_ADDR fields in I2C_FIFO_ST_REG register to obtain these addresses. When the RAM is accessed in FIFO mode, data can be read directly via I2C_DATA_REG register.

25.4.2 An I2C Master Writes to an I2C Slave with a 10-bit Address in One Command Sequence

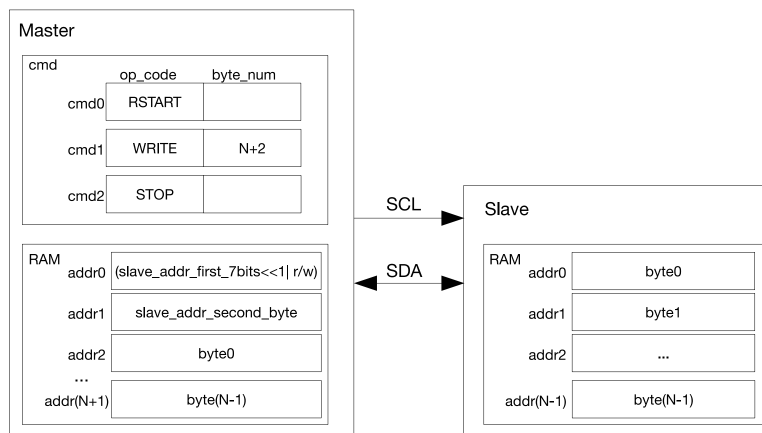


Figure 25-6. A Master Writing to a Slave with a 10-bit Address

Besides 7-bit addressing (SLV_ADDR[6:0]), the ESP32-S2 I2C controller also supports 10-bit addressing (SLV_ADDR[9:0]). In the following text, the slave address is referred to as SLV_ADDR.

Figure 25-6 shows how an I2C master writes N-bytes of data using 10-bit addressing. Unlike a 7-bit address, a 10-bit slave address is formed from two bytes. In master mode, the first byte of the slave address, which comprises slave_addr_first_7bits and a $R/\overline{W}$ bit, is stored into addr0 in the RAM. slave_addr_first_7bits should be configured as (0x78 | SLV_ADDR[9:8]). The second byte slave_addr_second_byte is stored into addr1 in the RAM. slave_addr_second_byte should be configured as SLV_ADDR[7:0].

In the slave, the 10-bit addressing mode is enabled by configuring I2C_ADDR_10BIT_EN bit. The address of the I2C slave is configured using I2C_SLAVE_ADDR. I2C_SLAVE_ADDR[14:7] should be configured as SLV_ADDR[7:0], and I2C_SLAVE_ADDR[6:0] should be configured as (0x78 | SLV_ADDR[9:8]). Since a 10-bit slave address has one more byte than a 7-bit address, byte_num of the WRITE command and the number of bytes in the RAM increase by one.

25.4.3 An I2C Master Writes to an I2C Slave with Two 7-bit Addresses in One Command Sequence

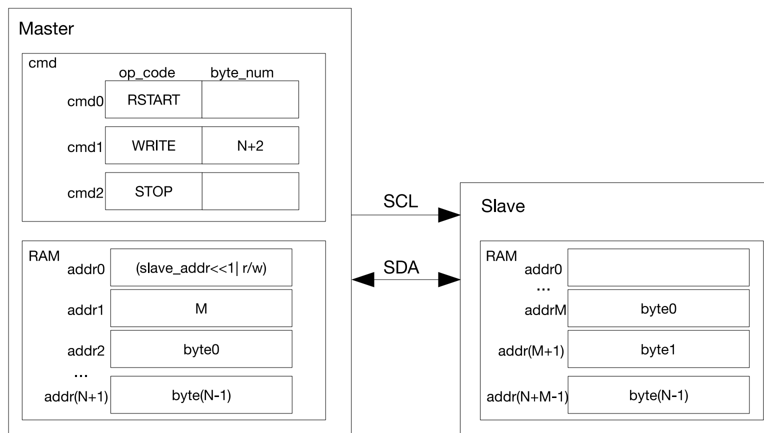


Figure 25-7. An I2C Master Writing Address M in the RAM to an I2C Slave with a 7-bit Address

When working in slave mode, the controller supports double addressing, where the first address is the address of an I2C slave, and the second one is the slave's memory address. Double addressing is enabled by configuring I2C_FIFO_ADDR_CFG_EN. When using double addressing, RAM must be accessed in non-FIFO mode. As figure 25-7 illustrates, the I2C slave put received byte0 ~ byte(N-1) into its RAM in an order staring from addrM. The RAM is overwritten every 32 bytes.

25.4.4 An I2C Master Writes to an I2C Slave with a 7-bit Address in Multiple Command Sequences

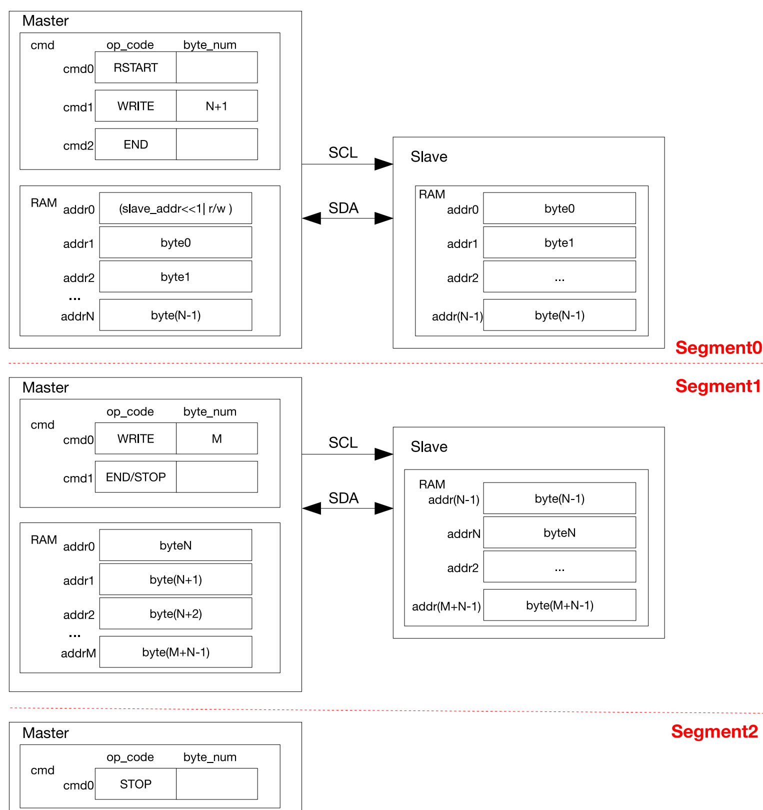


Figure 25-8. An I2C Master Writing to an I2C Slave with a 7-bit Address in Multiple Sequences

Given that the I2C Controller RAM holds only 32 bytes, when data are too large to be processed even by the wrapped RAM, it is advised to transmit them in multiple command sequences. At the end of every command sequence is an END command. When the controller executes this END command to pull SCL low, software refreshes command sequence registers and the RAM for next the transmission.

Figure 25-8 shows how an I2C master writes to an I2C slave in two or three segments. For the first segment, the CMD_Controller registers are configured as shown in Segment0. Once data in the master's RAM is ready and I2C_TRANS_START is set, the master initiates data transfer. After executing the END command, the master turns off the SCL clock and pulls the SCL low to reserve the bus. Meanwhile, the controller generates an I2C_END_DETECT_INT interrupt.

For the second segment, after detecting the I2C_END_DETECT_INT interrupt, software refreshes the CMD_Controller registers, reloads the RAM and clears this interrupt, as shown in Segment1. If cmd1 in the second segment is a STOP, then data is transmitted to the slave in two segments. The master resumes data transfer after I2C_TRANS_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C_END_DETECT_INT is detected, the CMD_Controller registers of the master are configured as shown in Segment2. Once I2C_TRANS_START is set, the master generates a STOP bit and terminates the transfer.

Please note that other I2C masters will not transact on the bus between two segments. The bus is only released after a STOP signal is sent. To interrupt the transfer, the I2C controller can be reset by setting I2C_FSM_RST at any time. This register will later be cleared automatically by hardware.

When the master is in IDLE state and I2C_SCL_RST_SLV_EN is set, hardware sends I2C_SCL_RST_SLV_NUM SCL pulses. I2C_SCL_RST_SLV_EN will later be cleared automatically by hardware.

Note that the operation of other I2C masters and I2C slaves may differ from that of the ESP32-S2 I2C devices. Please refer to datasheets of specific I2C devices.

25.4.5 An I2C Master Reads an I2C Slave with a 7-bit Address in One Command Sequence

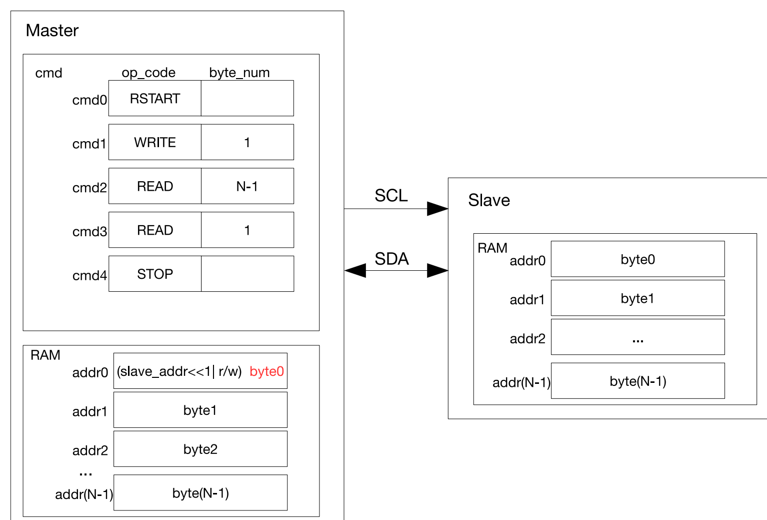


Figure 25-9. An I2C Master Reading an I2C Slave with a 7-bit Address

Figure 25-9 shows how an I2C master reads N bytes of data from an I2C slave using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed the master sends a slave address. The byte sent

comprises a 7-bit slave address and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 1, it indicates a READ operation. If the address of an I2C slave matches the sent address, this matching slave starts sending data to the master. The master generates acknowledgements according to ack_value defined in the READ command upon receiving every byte.

As illustrated in Figure 25-9, the master executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd 3. This configuration may be changed as required. The master writes received data into the controller RAM from addr0, whose original content (a slave address and a $R/\overline{W}$ bit) is overwritten by byte0 marked red in Figure 25-9.

25.4.6 An I2C Master Reads an I2C Slave with a 10-bit Address in One Command Sequence

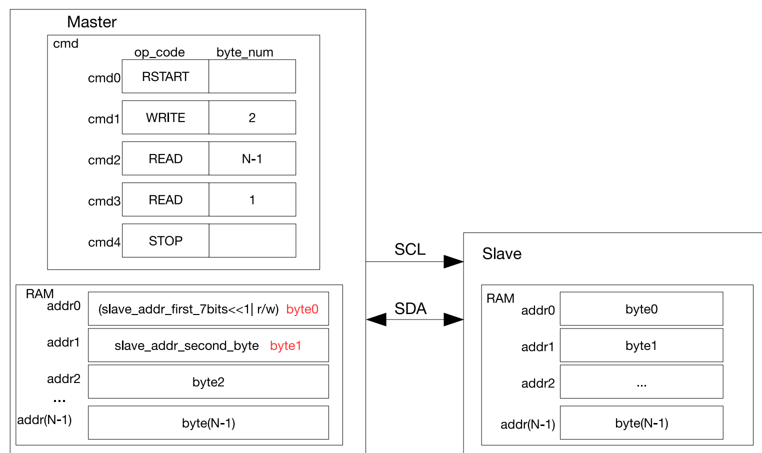


Figure 25-10. An I2C Master Reading an I2C Slave with a 10-bit Address

Figure 25-10 shows how an I2C master reads data from an I2C slave using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the I2C master is formed from two bytes, and correspondingly the RAM of this master stores a 10-bit address of two bytes. I2C_ADDR_10BIT_EN and I2C_SLAVE_ADDR[14:0] should be set. Please refer to 25.4.2 for how to set this register.

25.4.7 An I2C Master Reads an I2C Slave with Two 7-bit Addresses in One Command Sequence

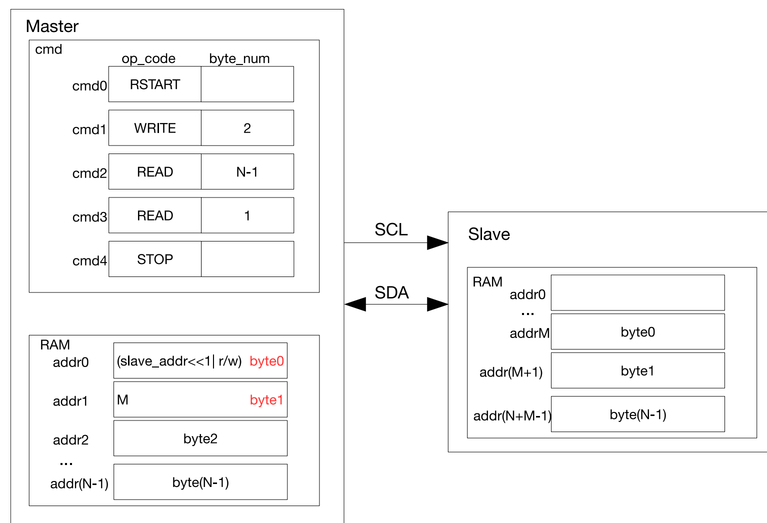


Figure 25-11. An I2C Master Reading N Bytes of Data from addrM of an I2C Slave with a 7-bit Address

Figure 25-11 shows how an I2C master reads data from specified addresses in an I2C slave. This procedure is as follows:

1. Set I2C_FIFO_ADDR_CFG_EN and prepare data to be sent in the RAM of the slave.
2. Prepare the slave address and register address M in the master.
3. Set I2C_TRANS_START and start transferring N bytes of data in the slave's RAM starting from address M to the master.

25.4.8 An I2C Master Reads an I2C Slave with a 7-bit Address in Multiple Command Sequences

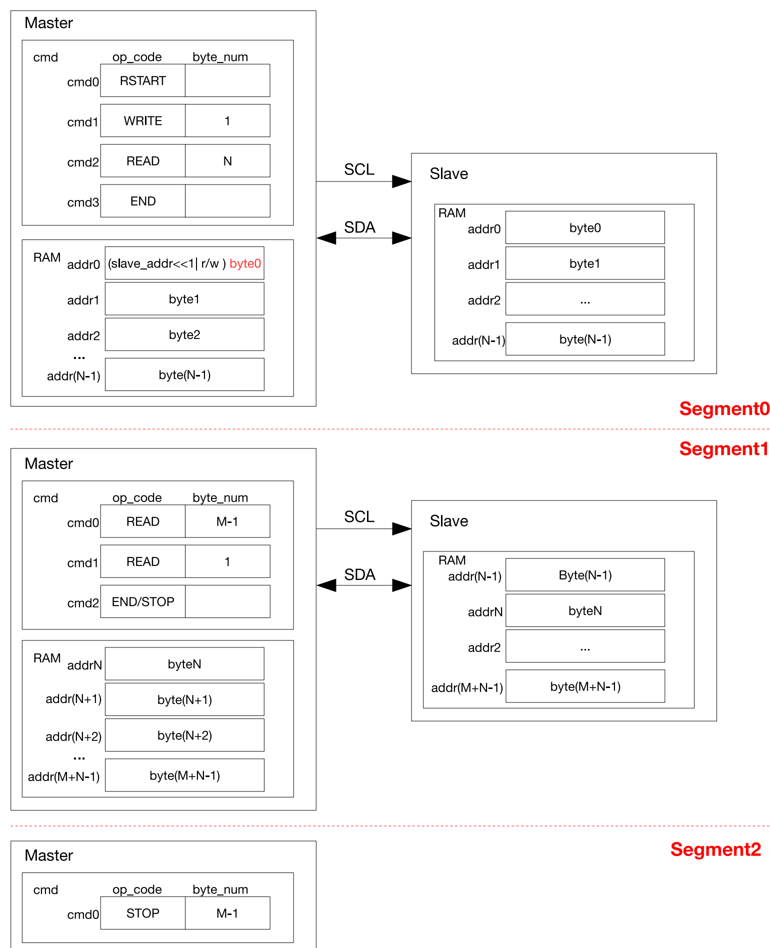


Figure 25-12. An I2C Master Reading an I2C Slave with a 7-bit Address in Segments

Figure 25-12 shows how an I2C master reads (N+M) bytes of data from an I2C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. Configure the command register and the RAM, as shown in Segment0.
2. Prepare data in the RAM of the slave, and set I2C_TRANS_START to start data transfer. After executing the END command, the master refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C_END_DETECT_INT interrupt. If cmd2 in the second segment is a STOP, then data is read from the slave in two segments. The master resumes data transfer by setting I2C_TRANS_START and terminates the transfer by sending a STOP bit.
3. If cmd2 in Segment1 is an END, then data is read from the slave in three segments. After the second data transfer finishes and an I2C_END_DETECT_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once I2C_TRANS_START is set, the master terminates the transfer by sending a STOP bit.

25.5 Clock Stretching

In slave mode, the I2C controller can hold the SCL line low in exchange for more processing time. This function called clock stretching is enabled by setting I2C_SLAVE_SCL_STRETCH_EN bit. The time period of clock stretching is configured by setting I2C_STRETCH_PROTECT_NUM bit. The SCL line will be held low when one

of the following three events occurs:

1. Finding a match: In slave mode, the address of the I2C controller matches the address sent via the SDA line.
2. RAM being full: In slave mode, RX RAM of the I2C controller is full.
3. Data all sent: In slave mode, TX RAM of the I2C controller is empty.

After SCL has been stretched low, the cause of stretching can be read from I2C_STRETCH_CAUSE bit. Clock stretching is disabled by setting I2C_SLAVE_SCL_STRETCH_CLR bit.

25.6 Interrupts

- I2C_SLAVE_STRETCH_INT: Generated when a slave holds SCL low.
- I2C_DET_START_INT: Triggered when a START bit is detected.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO[23:0] clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO[23:0] clock cycles.
- I2C_RXFIFO_UDF_INT: Triggered when the I2C controller receives I2C_NONFIFO_RX_THRES bytes of data in non-FIFO mode.
- I2C_TXFIFO_OVF_INT: Triggered when the I2C controller sends I2C_NONFIFO_TX_THRES bytes of data.
- I2C_NACK_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C_TRANS_START_INT: Triggered when the I2C controller sends a START bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than I2C_TIME_OUT clock cycles during data transfer.
- I2C_TRANS_COMPLETE_INT: Triggered when the I2C controller detects a STOP bit.
- I2C_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_ARBITRATION_LOST_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.
- I2C_BYTE_TRANS_DONE_INT: Triggered when the I2C controller sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_RXFIFO_OVF_INT: Triggered when Rx FIFO of the I2C controller overflows.
- I2C_TXFIFO_WM_INT: I2C TX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of TX FIFO are less than I2C_TXFIFO_WM_THRHD[4:0].
- I2C_RXFIFO_WM_INT: I2C Rx FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of Rx FIFO are greater than I2C_RXFIFO_WM_THRHD[4:0].

25.7 Base Address

Users can access the I2C Controller with base addresses in Table 147. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 147: I2C Controller Base Address

Module	Bus to Access Peripheral	Base Address
I2C0	PeriBUS1	0x3F413000
	PeriBUS2	0x60013000
I2C1	PeriBUS1	0x3F427000
	PeriBUS2	0x60027000

25.8 Register Summary

Addresses in the following table are relative to I2C base addresses provided in Section 25.7.

Name	Description	Address	Access
Timing Register			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL clock	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of the SCL clock	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the interval between pulling SDA low and pulling SCL low when the master generates a START condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the interval between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0098	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x009C	R/W
Configuration Register			
I2C_CTR_REG	Transmission setting	0x0004	R/W
I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
I2C_SLAVE_ADDR_REG	Local slave address setting	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
I2C_SCL_SP_CONF_REG	Power configuration register	0x00A0	R/W
I2C_SCL_STRETCH_CONF_REG	Set SCL stretch of I2C slave	0x00A4	varies
Status Register			
I2C_SR_REG	Describe I2C work status	0x0008	RO
I2C_FIFO_ST_REG	FIFO status register	0x0014	varies
I2C_DATA_REG	RX FIFO read data	0x001C	RO

Name	Description	Address	Access
Interrupt Register			
I2C_INT_RAW_REG	Raw interrupt status	0x0020	RO
I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WO
I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W
I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Filter Register			
I2C_SCL_FILTER_CFG_REG	SCL filter configuration register	0x0050	R/W
I2C_SDA_FILTER_CFG_REG	SDA filter configuration register	0x0054	R/W
Command Register			
I2C_COMD0_REG	I2C command register 0	0x0058	R/W
I2C_COMD1_REG	I2C command register 1	0x005C	R/W
I2C_COMD2_REG	I2C command register 2	0x0060	R/W
I2C_COMD3_REG	I2C command register 3	0x0064	R/W
I2C_COMD4_REG	I2C command register 4	0x0068	R/W
I2C_COMD5_REG	I2C command register 5	0x006C	R/W
I2C_COMD6_REG	I2C command register 6	0x0070	R/W
I2C_COMD7_REG	I2C command register 7	0x0074	R/W
I2C_COMD8_REG	I2C command register 8	0x0078	R/W
I2C_COMD9_REG	I2C command register 9	0x007C	R/W
I2C_COMD10_REG	I2C command register 10	0x0080	R/W
I2C_COMD11_REG	I2C command register 11	0x0084	R/W
I2C_COMD12_REG	I2C command register 12	0x0088	R/W
I2C_COMD13_REG	I2C command register 13	0x008C	R/W
I2C_COMD14_REG	I2C command register 14	0x0090	R/W
I2C_COMD15_REG	I2C command register 15	0x0094	R/W
Version Register			
I2C_DATE_REG	Version control register	0x00F8	R/W

25.9 Registers

Register 25.1: I2C_SCL_LOW_PERIOD_REG (0x0000)

I2C_SCL_LOW_PERIOD This register is used to configure for how long SCL remains low in master mode, in I2C module clock cycles. (R/W)

Register 25.2: I2C_SDA_HOLD_REG (0x0030)

(reserved)	I2C_SDA_HOLD_TIME	
31 ... 10	9 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

I2C_SDA_HOLD_TIME This register is used to configure the interval between changing the SDA output level and the falling edge of SCL, in I2C module clock cycles. (R/W)

Register 25.3: I2C_SDA_SAMPLE_REG (0x0034)

(reserved)	I2C_SDA_SAMPLE_TIME	
31 ... 10	9 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

I2C_SDA_SAMPLE_TIME This register is used to configure the interval between the rising edge of SCL and the level sampling time of SDA, in I2C module clock cycles. (R/W)

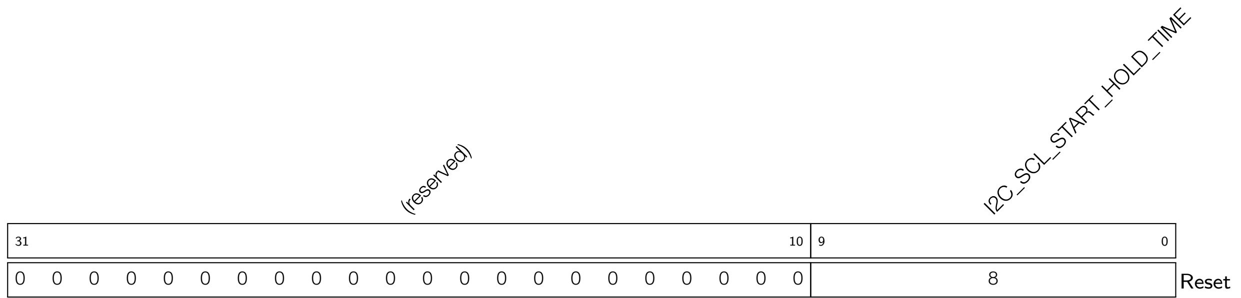
Register 25.4: I2C_SCL_HIGH_PERIOD_REG (0x0038)

(reserved)	I2C_SCL_WAIT_HIGH_PERIOD	I2C_SCL_HIGH_PERIOD	
31 ... 28	27 ... 14	13 ... 0	
0 0 0 0	0x00	0x00	Reset

I2C_SCL_HIGH_PERIOD This register is used to configure for how long SCL remains high in master mode, in I2C module clock cycles. (R/W)

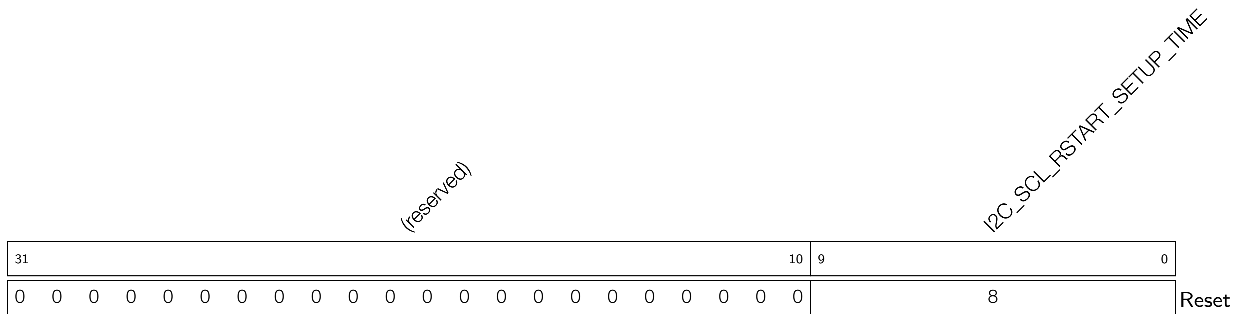
I2C_SCL_WAIT_HIGH_PERIOD This register is used to configure for the SCL_FSM's waiting period for SCL to go high in master mode, in I2C module clock cycles. (R/W)

Register 25.5: I2C_SCL_START_HOLD_REG (0x0040)



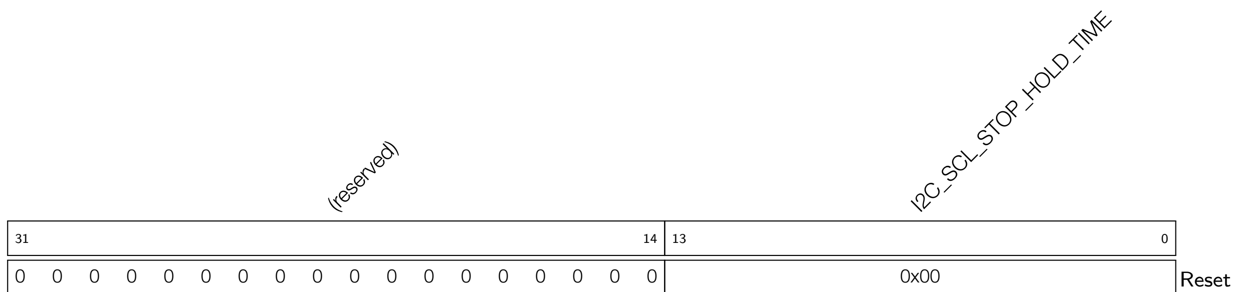
I2C_SCL_START_HOLD_TIME This register is used to configure interval between pulling SDA low and pulling SCL low when the master generates a START condition, in I2C module clock cycles. (R/W)

Register 25.6: I2C_SCL_RSTART_SETUP_REG (0x0044)



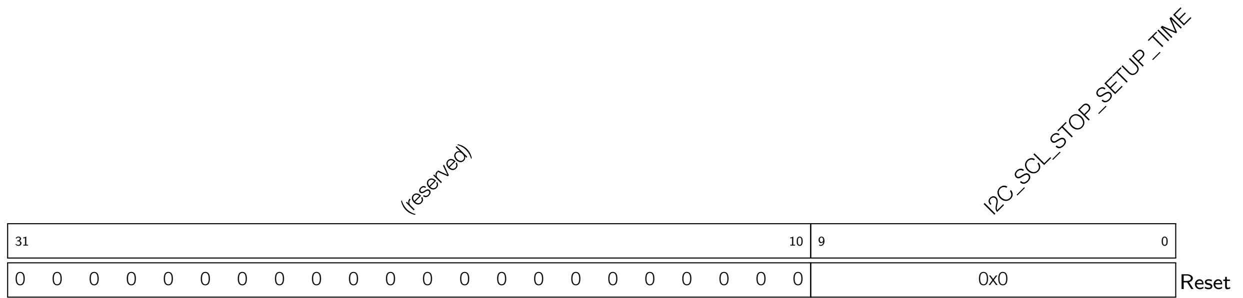
I2C_SCL_RSTART_SETUP_TIME This register is used to configure the interval between the positive edge of SCL and the negative edge of SDA for a RESTART condition, in I2C module clock cycles. (R/W)

Register 25.7: I2C_SCL_STOP_HOLD_REG (0x0048)



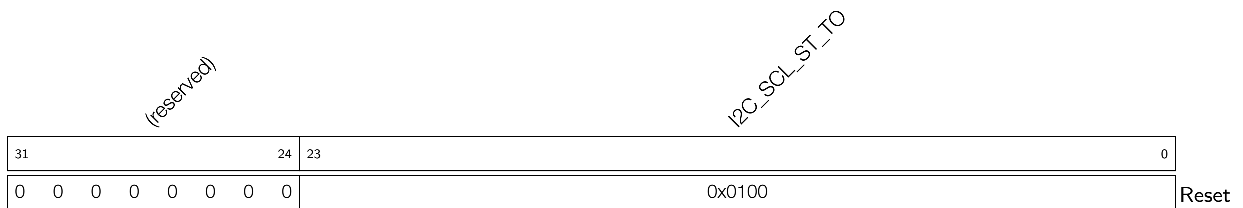
I2C_SCL_STOP_HOLD_TIME This register is used to configure the delay after the STOP condition, in I2C module clock cycles. (R/W)

Register 25.8: I2C_SCL_STOP_SETUP_REG (0x004C)



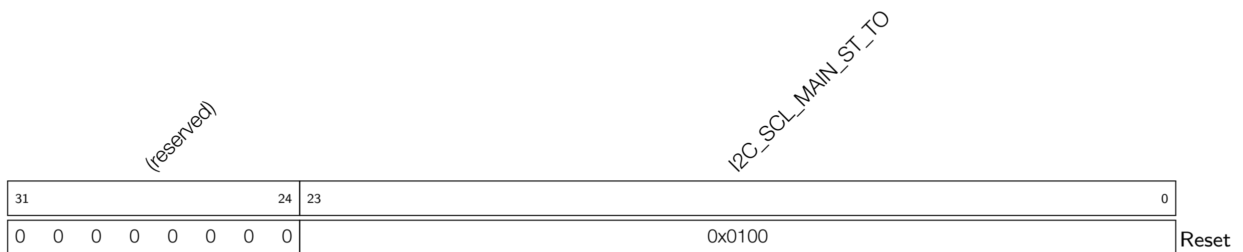
I2C_SCL_STOP_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the positive edge of SDA, in I2C module clock cycles. (R/W)

Register 25.9: I2C_SCL_ST_TIME_OUT_REG (0x0098)



I2C_SCL_ST_TO The threshold value of SCL_FSM state unchanged period. (R/W)

Register 25.10: I2C_SCL_MAIN_ST_TIME_OUT_REG (0x009C)



I2C_SCL_MAIN_ST_TO The threshold value of SCL_MAIN_FSM state unchanged period. (R/W)

Register 25.11: I2C_CTR_REG (0x0004)

(reserved)	I2C_REF_ALWAYS_ON	I2C_FSM_RST	I2C_ARBITRATION_EN	I2C_CLK_EN	I2C_RX_LSB_FIRST	I2C_TX_LSB_FIRST	I2C_TRANS_START	I2C_MS_MODE	I2C_RX_FULL_ACK_LEVEL	I2C_SAMPLE_SCL_LEVEL	I2C_SCL_FORCE_OUT	I2C_SDA_FORCE_OUT	
31 ... 12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	1	0	0	0	0	0	1	0	1	1	Reset

I2C_SDA_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SCL_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SAMPLE_SCL_LEVEL This register is used to select the sample mode. 1: sample SDA data on the SCL low level. 0: sample SDA data on the SCL high level. (R/W)

I2C_RX_FULL_ACK_LEVEL This register is used to configure the ACK value that need to sent by master when the rx_fifo_cnt has reached the threshold. (R/W)

I2C_MS_MODE Set this bit to configure the module as an I2C Master. Clear this bit to configure the module as an I2C Slave. (R/W)

I2C_TRANS_START Set this bit to start sending the data in TX FIFO. (R/W)

I2C_TX_LSB_FIRST This bit is used to control the sending mode for data needing to be sent. 1: send data from the least significant bit; 0: send data from the most significant bit. (R/W)

I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. 1: receive data from the least significant bit; 0: receive data from the most significant bit. (R/W)

I2C_CLK_EN Reserved (R/W)

I2C_ARBITRATION_EN This is the enable bit for I2C bus arbitration function. (R/W)

I2C_FSM_RST This register is used to reset the SCL_FSM. (R/W)

I2C_REF_ALWAYS_ON This register is used to control the REF_TICK. (R/W)

Register 25.12: I2C_TO_REG (0x000C)

(reserved)	I2C_TIME_OUT_EN	I2C_TIME_OUT_VALUE	
31–25	24	23–0	
0 0 0 0 0 0 0	0	0x0000	Reset

I2C_TIME_OUT_VALUE This register is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

I2C_TIME_OUT_EN This is the enable bit for time out control. (R/W)

Register 25.13: I2C_SLAVE_ADDR_REG (0x0010)

I2C_ADDR_10BIT_EN	(reserved)	I2C_SLAVE_ADDR	
31	30–15	14–0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_SLAVE_ADDR When configured as an I2C Slave, this field is used to configure the slave address. (R/W)

I2C_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode in master mode. (R/W)

Register 25.14: I2C_FIFO_CONF_REG (0x0018)

(reserved)	I2C_FIFO_PRT_EN	I2C_NONFIFO_TX_THRES	I2C_NONFIFO_RX_THRES	I2C_TX_FIFO_RST	I2C_RX_FIFO_RST	I2C_FIFO_ADDR_CFG_EN	I2C_NONFIFO_EN	I2C_TXFIFO_WM_THRHD	I2C_RXFIFO_WM_THRHD	
31 27	26	25 20	19 14	13	12	11	10	9 5	4 0	
0 0 0 0 0	1	0x15	0x15	0	0	0	0	0x4	0xb	Reset

I2C_RXFIFO_WM_THRHD The water mark threshold of RX FIFO in non-FIFO mode. When I2C_FIFO_PRT_EN is 1 and RX FIFO counter is bigger than I2C_RXFIFO_WM_THRHD[4:0], I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_TXFIFO_WM_THRHD The water mark threshold of TX FIFO in non-FIFO mode. When I2C_FIFO_PRT_EN is 1 and TX FIFO counter is smaller than I2C_TXFIFO_WM_THRHD[4:0], I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_NONFIFO_EN Set this bit to enable APB non-FIFO mode. (R/W)

I2C_FIFO_ADDR_CFG_EN When this bit is set to 1, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM. (R/W)

I2C_RX_FIFO_RST Set this bit to reset RX FIFO. (R/W)

I2C_TX_FIFO_RST Set this bit to reset TX FIFO. (R/W)

I2C_NONFIFO_RX_THRES When I2C receives more than I2C_NONFIFO_RX_THRES bytes of data, it will generate an I2C_RXFIFO_UDF_INT interrupt and update the current offset address of the received data. (R/W)

I2C_NONFIFO_TX_THRES When I2C sends more than I2C_NONFIFO_TX_THRES bytes of data, it will generate an I2C_TXFIFO_OVF_INT interrupt and update the current offset address of the sent data. (R/W)

I2C_FIFO_PRT_EN The control enable bit of FIFO pointer in non-FIFO mode. This bit controls the valid bits and the interrupts of TX/RX FIFO overflow, underflow, full and empty. (R/W)

Register 25.15: I2C_SCL_SP_CONF_REG (0x00A0)

(reserved)	I2C_SDA_PD_EN	I2C_SCL_PD_EN	I2C_SCL_RST_SLV_NUM	I2C_SCL_RST_SLV_EN	
31 ... 8	7	6	5 ... 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0x0	0	Reset

I2C_SCL_RST_SLV_EN When I2C master is IDLE, set this bit to send out SCL pulses. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0]. (R/W)

I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode. Valid when I2C_SCL_RST_SLV_EN is 1. (R/W)

I2C_SCL_PD_EN The power down enable bit for the I2C output SCL line. 1: Power down. 0: Not power down. Set I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN to 1 to stretch SCL low. (R/W)

I2C_SDA_PD_EN The power down enable bit for the I2C output SDA line. 1: Power down. 0: Not power down. Set I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN to 1 to stretch SDA low. (R/W)

Register 25.16: I2C_SCL_STRETCH_CONF_REG (0x00A4)

(reserved)	I2C_SLAVE_SCL_STRETCH_CLR	I2C_SLAVE_SCL_STRETCH_EN	I2C_STRETCH_PROTECT_NUM	
31 ... 12	11	10	9 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0x0	Reset

I2C_STRETCH_PROTECT_NUM Configure the period of I2C slave stretching SCL line. (R/W)

I2C_SLAVE_SCL_STRETCH_EN The enable bit for slave SCL stretch function. 1: Enable. 0: Disable. The SCL output line will be stretched low when I2C_SLAVE_SCL_STRETCH_EN is 1 and stretch event happens. The stretch cause can be seen in I2C_STRETCH_CAUSE. (R/W)

I2C_SLAVE_SCL_STRETCH_CLR Set this bit to clear the I2C slave SCL stretch function. (WO)

Register 25.17: I2C_SR_REG (0x0008)

31	30 28	27	26 24	23 18	17 16	15 14	13 8	7	6	5	4	3	2	1	0	
(reserved)	I2C_SCL_STATE_LAST	(reserved)	I2C_SCL_MAIN_STATE_LAST	I2C_TXFIFO_CNT	(reserved)	I2C_STRETCH_CAUSE	I2C_RXFIFO_CNT	(reserved)	I2C_BYTE_TRANS	I2C_SLAVE_ADDRESSED	I2C_BUS_BUSY	I2C_ARB_LOST	I2C_TIME_OUT	I2C_SLAVE_RW	I2C_RESP_REC	
0	0x0	0	0x0	0x0	0 0	0x0	0x0	0	0	0	0	0	0	0	0	Reset

I2C_RESP_REC The received ACK value in master mode or slave mode. 0: ACK; 1: NACK. (RO)

I2C_SLAVE_RW When in slave mode, 1: master reads from slave; 0: master writes to slave. (RO)

I2C_TIME_OUT When the I2C controller takes more than I2C_TIME_OUT clocks to receive a data bit, this field changes to 1. (RO)

I2C_ARB_LOST When the I2C controller loses control of SCL line, this register changes to 1. (RO)

I2C_BUS_BUSY 1: the I2C bus is busy transferring data; 0: the I2C bus is in idle state. (RO)

I2C_SLAVE_ADDRESSED When configured as an I2C Slave, and the address sent by the master is equal to the address of the slave, then this bit will be of high level. (RO)

I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

I2C_RXFIFO_CNT This field represents the amount of data needed to be sent. (RO)

I2C_STRETCH_CAUSE The cause of stretching SCL low in slave mode. 0: stretching SCL low at the beginning of I2C read data state. 1: stretching SCL low when I2C TX FIFO is empty in slave mode. 2: stretching SCL low when I2C RX FIFO is full in slave mode. (RO)

I2C_TXFIFO_CNT This field stores the amount of received data in RAM. (RO)

I2C_SCL_MAIN_STATE_LAST This field indicates the states of the I2C module state machine. 0: Idle; 1: Address shift; 2: ACK address; 3: RX data; 4: TX data; 5: Send ACK; 6: Wait ACK (RO)

I2C_SCL_STATE_LAST This field indicates the states of the state machine used to produce SCL. 0: Idle; 1: Start; 2: Negative edge; 3: Low; 4: Positive edge; 5: High; 6: Stop (RO)

Register 25.18: I2C_FIFO_ST_REG (0x0014)

(reserved)		I2C_SLAVE_RW_POINT	I2C_TX_UPDATE	I2C_RX_UPDATE	I2C_TXFIFO_END_ADDR	I2C_TXFIFO_START_ADDR	I2C_RXFIFO_END_ADDR	I2C_RXFIFO_START_ADDR	
31	30	29 22	21	20	19 15	14 10	9 5	4 0	
0	0	0x0	0	0	0x0	0x0	0x0	0x0	Reset

I2C_RXFIFO_START_ADDR This is the offset address of the last received data, as described in I2C_NONFIFO_RX_THRES. (RO)

I2C_RXFIFO_END_ADDR This is the offset address of the last received data, as described in I2C_NONFIFO_RX_THRES. This value refreshes when an I2C_RXFIFO_UDF_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_TXFIFO_START_ADDR This is the offset address of the first sent data, as described in I2C_NONFIFO_TX_THRES. (RO)

I2C_TXFIFO_END_ADDR This is the offset address of the last sent data, as described in I2C_NONFIFO_TX_THRES. The value refreshes when an I2C_TXFIFO_OVF_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_RX_UPDATE Write 0 or 1 to I2C_RX_UPDATE to update the value of I2C_RXFIFO_END_ADDR and I2C_RXFIFO_START_ADDR. (WO)

I2C_TX_UPDATE Write 0 or 1 to I2C_TX_UPDATE to update the value of I2C_TXFIFO_END_ADDR and I2C_TXFIFO_START_ADDR. (WO)

I2C_SLAVE_RW_POINT The received data in I2C slave mode. (RO)

Register 25.19: I2C_DATA_REG (0x001C)

(reserved)	I2C_FIFO_RDATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

I2C_FIFO_RDATA The value of RX FIFO read data. (RO)

Register 25.20: I2C_INT_RAW_REG (0x0020)

31..17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2C_SLAVE_STRETCH_INT_RAW	I2C_DET_START_INT_RAW	I2C_SCL_MAIN_ST_TO_INT_RAW	I2C_SCL_ST_TO_INT_RAW	I2C_RXFIFO_UDF_INT_RAW	I2C_TXFIFO_OVF_INT_RAW	I2C_NACK_INT_RAW	I2C_TRANS_START_INT_RAW	I2C_TIME_OUT_INT_RAW	I2C_TRANS_COMPLETE_INT_RAW	I2C_MST_TXFIFO_UDF_INT_RAW	I2C_ARBITRATION_LOST_INT_RAW	I2C_BYTE_TRANS_DONE_INT_RAW	I2C_END_DETECT_INT_RAW	I2C_RXFIFO_OVF_INT_RAW	I2C_TXFIFO_WM_INT_RAW	I2C_RXFIFO_WM_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2C_RXFIFO_WM_INT_RAW The raw interrupt bit for I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_RAW The raw interrupt bit for I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt bit for I2C_RXFIFO_OVF_INT interrupt. (RO)

I2C_END_DETECT_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt bit for I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TIME_OUT_INT_RAW The raw interrupt bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_START_INT_RAW The raw interrupt bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_NACK_INT_RAW The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt bit for I2C_TXFIFO_OVF_INT interrupt. (RO)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt bit for I2C_RXFIFO_UDF_INT interrupt. (RO)

I2C_SCL_ST_TO_INT_RAW The raw interrupt bit for I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

I2C_DET_START_INT_RAW The raw interrupt bit for I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_RAW The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

Register 25.21: I2C_INT_CLR_REG (0x0024)

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2C_SLAVE_STRETCH_INT_CLR	I2C_DET_START_INT_CLR	I2C_SCL_MAIN_ST_TO_INT_CLR	I2C_SCL_ST_TO_INT_CLR	I2C_RXFIFO_UDF_INT_CLR	I2C_TXFIFO_OVF_INT_CLR	I2C_NACK_INT_CLR	I2C_TRANS_START_INT_CLR	I2C_TIME_OUT_INT_CLR	I2C_TRANS_COMPLETE_INT_CLR	I2C_MST_TXFIFO_UDF_INT_CLR	I2C_ARBITRATION_LOST_INT_CLR	I2C_BYTE_TRANS_DONE_INT_CLR	I2C_END_DETECT_INT_CLR	I2C_RXFIFO_OVF_INT_CLR	I2C_TXFIFO_WM_INT_CLR	I2C_RXFIFO_WM_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2C_RXFIFO_WM_INT_CLR Set this bit to clear I2C_RXFIFO_WM_INT interrupt. (WO)

I2C_TXFIFO_WM_INT_CLR Set this bit to clear I2C_TXFIFO_WM_INT interrupt. (WO)

I2C_RXFIFO_OVF_INT_CLR Set this bit to clear I2C_RXFIFO_OVF_INT interrupt. (WO)

I2C_END_DETECT_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

I2C_BYTE_TRANS_DONE_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

I2C_ARBITRATION_LOST_INT_CLR Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WO)

I2C_MST_TXFIFO_UDF_INT_CLR Set this bit to clear I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_TRANS_COMPLETE_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_TIME_OUT_INT_CLR Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WO)

I2C_TRANS_START_INT_CLR Set this bit to clear the I2C_TRANS_START_INT interrupt. (WO)

I2C_NACK_INT_CLR Set this bit to clear I2C_SLAVE_STRETCH_INT interrupt. (WO)

I2C_TXFIFO_OVF_INT_CLR Set this bit to clear I2C_TXFIFO_OVF_INT interrupt. (WO)

I2C_RXFIFO_UDF_INT_CLR Set this bit to clear I2C_RXFIFO_UDF_INT interrupt. (WO)

I2C_SCL_ST_TO_INT_CLR Set this bit to clear I2C_SCL_ST_TO_INT interrupt. (WO)

I2C_SCL_MAIN_ST_TO_INT_CLR Set this bit to clear I2C_SCL_MAIN_ST_TO_INT interrupt. (WO)

I2C_DET_START_INT_CLR Set this bit to clear I2C_DET_START_INT interrupt. (WO)

I2C_SLAVE_STRETCH_INT_CLR Set this bit to clear I2C_SLAVE_STRETCH_INT interrupt. (WO)

Register 25.22: I2C_INT_ENA_REG (0x0028)

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2C_SLAVE_STRETCH_INT_ENA	I2C_DET_START_INT_ENA	I2C_SCL_MAIN_ST_TO_INT_ENA	I2C_SCL_ST_TO_INT_ENA	I2C_RXFIFO_UDF_INT_ENA	I2C_TXFIFO_OVF_INT_ENA	I2C_NACK_INT_ENA	I2C_TRANS_START_INT_ENA	I2C_TIME_OUT_INT_ENA	I2C_TRANS_COMPLETE_INT_ENA	I2C_MST_TXFIFO_UDF_INT_ENA	I2C_ARBITRATION_LOST_INT_ENA	I2C_BYTE_TRANS_DONE_INT_ENA	I2C_END_DETECT_INT_ENA	I2C_RXFIFO_OVF_INT_ENA	I2C_TXFIFO_WM_INT_ENA	I2C_RXFIFO_WM_INT_ENA	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2C_RXFIFO_WM_INT_ENA The raw interrupt bit for I2C_RXFIFO_WM_INT interrupt. (R/W)

I2C_TXFIFO_WM_INT_ENA The raw interrupt bit for I2C_TXFIFO_WM_INT interrupt. (R/W)

I2C_RXFIFO_OVF_INT_ENA The raw interrupt bit for I2C_RXFIFO_OVF_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/W)

I2C_BYTE_TRANS_DONE_INT_ENA The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA The raw interrupt bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_MST_TXFIFO_UDF_INT_ENA The raw interrupt bit for I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TRANS_COMPLETE_INT_ENA The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA The raw interrupt bit for the I2C_TIME_OUT_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA The raw interrupt bit for the I2C_TRANS_START_INT interrupt. (R/W)

I2C_NACK_INT_ENA The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (R/W)

I2C_TXFIFO_OVF_INT_ENA The raw interrupt bit for I2C_TXFIFO_OVF_INT interrupt. (R/W)

I2C_RXFIFO_UDF_INT_ENA The raw interrupt bit for I2C_RXFIFO_UDF_INT interrupt. (R/W)

I2C_SCL_ST_TO_INT_ENA The raw interrupt bit for I2C_SCL_ST_TO_INT interrupt. (R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA The raw interrupt bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

I2C_DET_START_INT_ENA The raw interrupt bit for I2C_DET_START_INT interrupt. (R/W)

I2C_SLAVE_STRETCH_INT_ENA The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (R/W)

Register 25.23: I2C_INT_STATUS_REG (0x002C)

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2C_SLAVE_STRETCH_INT_ST	I2C_DET_START_INT_ST	I2C_SCL_MAIN_ST_TO_INT_ST	I2C_SCL_ST_TO_INT_ST	I2C_RXFIFO_UDF_INT_ST	I2C_TXFIFO_OVF_INT_ST	I2C_NACK_INT_ST	I2C_TRANS_START_INT_ST	I2C_TIME_OUT_INT_ST	I2C_TRANS_COMPLETE_INT_ST	I2C_MST_TXFIFO_UDF_INT_ST	I2C_ARBITRATION_LOST_INT_ST	I2C_BYTE_TRANS_DONE_INT_ST	I2C_END_DETECT_INT_ST	I2C_RXFIFO_OVF_INT_ST	I2C_TXFIFO_WM_INT_ST	I2C_RXFIFO_WM_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2C_RXFIFO_WM_INT_ST The masked interrupt status bit for I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_ST The masked interrupt status bit for I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_ST The masked interrupt status bit for I2C_RXFIFO_OVF_INT interrupt. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status bit for I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_ST The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_NACK_INT_ST The masked interrupt status bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_ST The masked interrupt status bit for I2C_TXFIFO_OVF_INT interrupt. (RO)

I2C_RXFIFO_UDF_INT_ST The masked interrupt status bit for I2C_RXFIFO_UDF_INT interrupt. (RO)

I2C_SCL_ST_TO_INT_ST The masked interrupt status bit for I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

I2C_DET_START_INT_ST The masked interrupt status bit for I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_ST The masked interrupt status bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

Register 25.24: I2C_SCL_FILTER_CFG_REG (0x0050)

(reserved)	I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	
31 … 5	4	3 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0x0	Reset

I2C_SCL_FILTER_THRES When a pulse on the SCL input has smaller width than this register value in I2C module clock cycles, the I2C controller will ignore that pulse. (R/W)

I2C_SCL_FILTER_EN This is the filter enable bit for SCL. (R/W)

Register 25.25: I2C_SDA_FILTER_CFG_REG (0x0054)

(reserved)	I2C_SDA_FILTER_EN	I2C_SDA_FILTER_THRES	
31 … 5	4	3 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0x0	Reset

I2C_SDA_FILTER_THRES When a pulse on the SDA input has smaller width than this register value in I2C module clock cycles, the I2C controller will ignore that pulse. (R/W)

I2C_SDA_FILTER_EN This is the filter enable bit for SDA. (R/W)

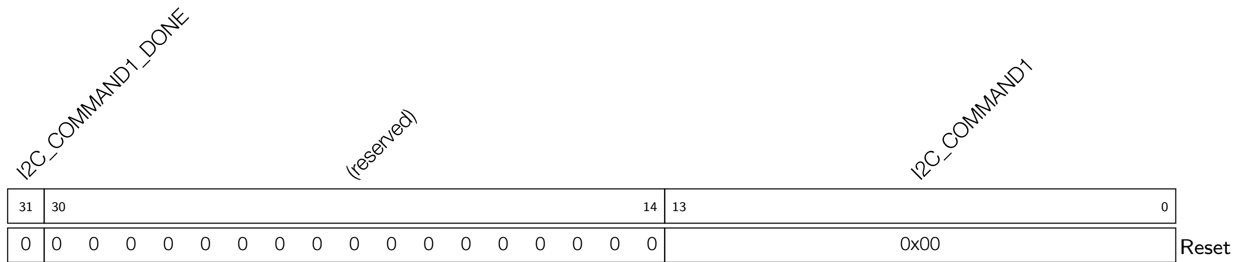
Register 25.26: I2C_COMD0_REG (0x0058)

I2C_COMMAND0_DONE	(reserved)	I2C_COMMAND0	
31	30 … 14	13 … 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_COMMAND0 This is the content of command 0. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND0_DONE When command 0 is done in I2C Master mode, this bit changes to high level. (R/W)

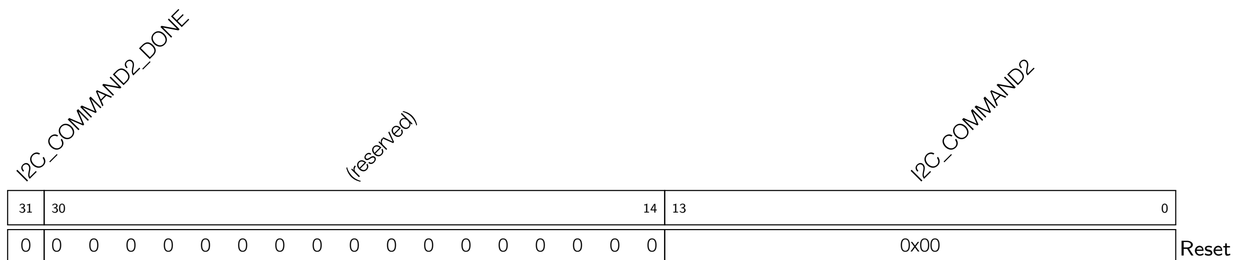
Register 25.27: I2C_COMD1_REG (0x005C)



I2C_COMMAND1 This is the content of command 1. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND1_DONE When command 1 is done in I2C Master mode, this bit changes to high level. (R/W)

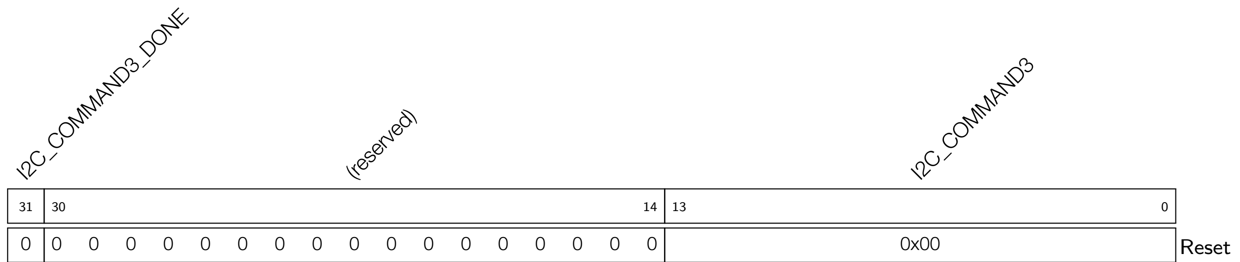
Register 25.28: I2C_COMD2_REG (0x0060)



I2C_COMMAND2 This is the content of command 2. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND2_DONE When command 2 is done in I2C Master mode, this bit changes to high Level. (R/W)

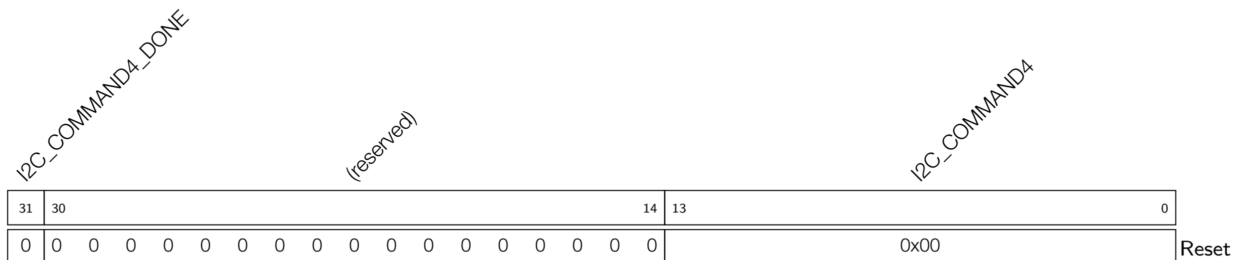
Register 25.29: I2C_COMD3_REG (0x0064)



I2C_COMMAND3 This is the content of command 3. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND3_DONE When command 3 is done in I2C Master mode, this bit changes to high level. (R/W)

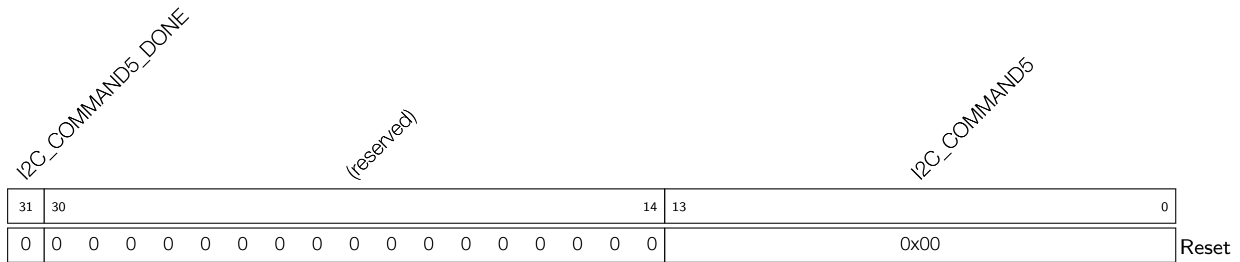
Register 25.30: I2C_COMD4_REG (0x0068)



I2C_COMMAND4 This is the content of command 4. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND4_DONE When command 4 is done in I2C Master mode, this bit changes to high level. (R/W)

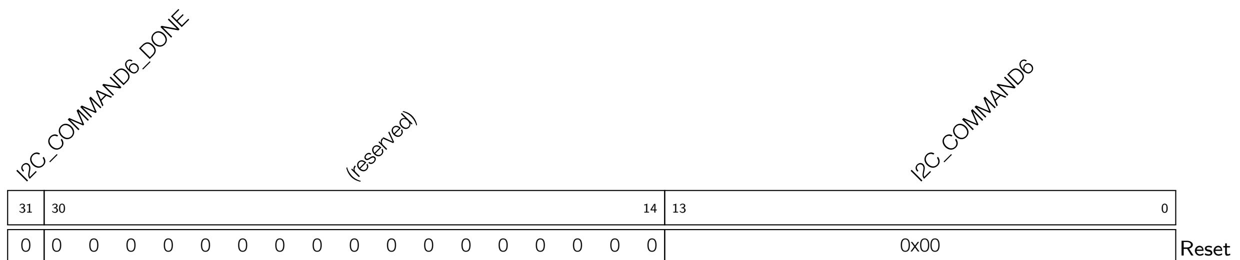
Register 25.31: I2C_COMD5_REG (0x006C)



I2C_COMMAND5 This is the content of command 5. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND5_DONE When command 5 is done in I2C Master mode, this bit changes to high level. (R/W)

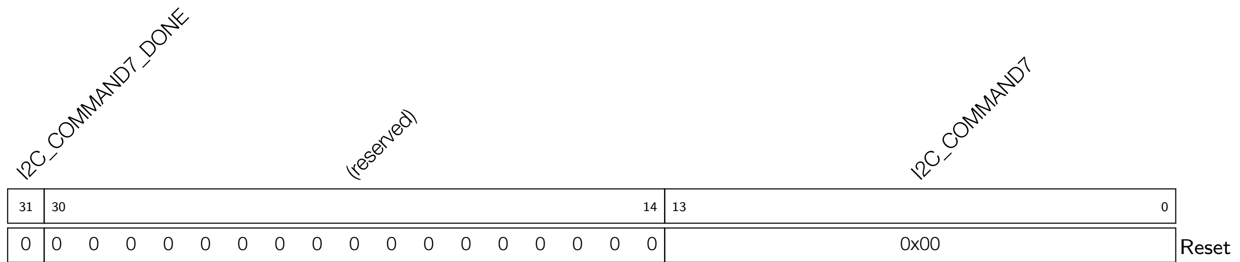
Register 25.32: I2C_COMD6_REG (0x0070)



I2C_COMMAND6 This is the content of command 6. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND6_DONE When command 6 is done in I2C Master mode, this bit changes to high level. (R/W)

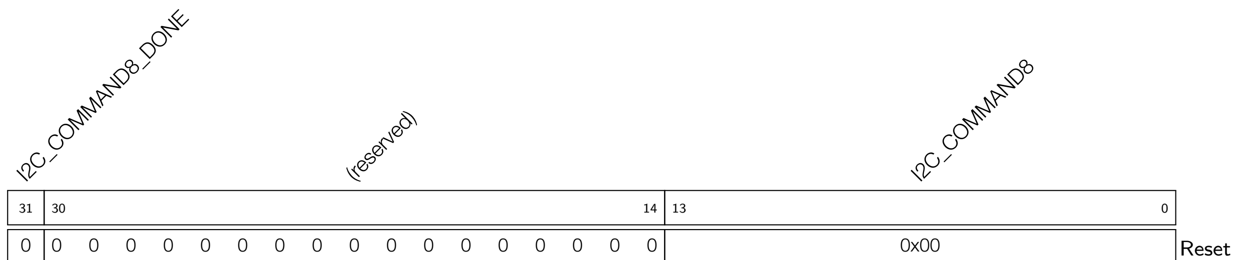
Register 25.33: I2C_COMD7_REG (0x0074)



I2C_COMMAND7 This is the content of command 7. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND7_DONE When command 7 is done in I2C Master mode, this bit changes to high level. (R/W)

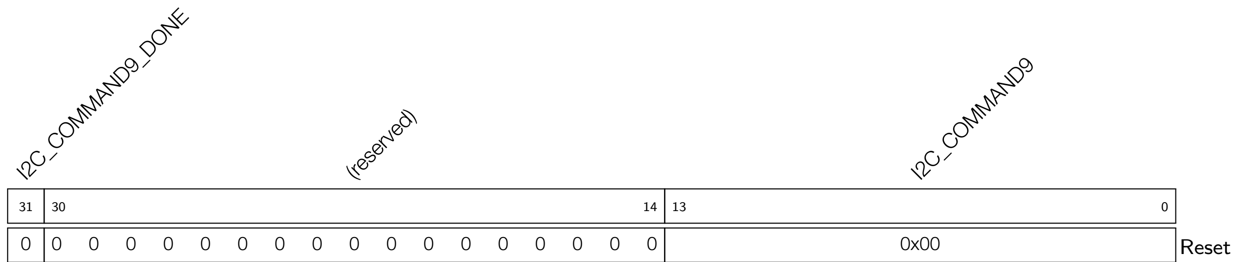
Register 25.34: I2C_COMD8_REG (0x0078)



I2C_COMMAND8 This is the content of command 8. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND8_DONE When command 8 is done in I2C Master mode, this bit changes to high level. (R/W)

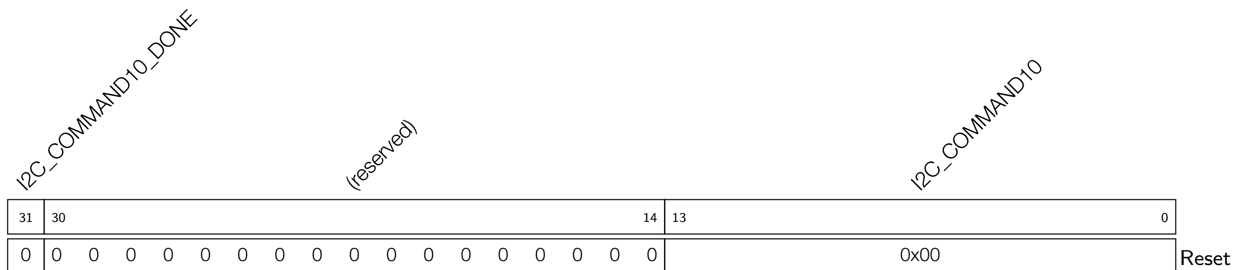
Register 25.35: I2C_COMD9_REG (0x007C)



I2C_COMMAND9 This is the content of command 9. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND9_DONE When command 9 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 25.36: I2C_COMD10_REG (0x0080)



I2C_COMMAND10 This is the content of command 10. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND10_DONE When command 10 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 25.37: I2C_COMD11_REG (0x0084)

I2C_COMMAND11_DONE	(reserved)	I2C_COMMAND11	
31	30 ... 14	13 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_COMMAND11 This is the content of command 11. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND11_DONE When command 11 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 25.38: I2C_COMD12_REG (0x0088)

I2C_COMMAND12_DONE	(reserved)	I2C_COMMAND12	
31	30 ... 14	13 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_COMMAND12 This is the content of command 12. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND12_DONE When command 12 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 25.39: I2C_COMD13_REG (0x008C)

I2C_COMMAND13_DONE	(reserved)	I2C_COMMAND13	
31	30 ... 14	13 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_COMMAND13 This is the content of command 13. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND13_DONE When command 13 is done in I2C Master mode, this bit changes to high level. (R/W)

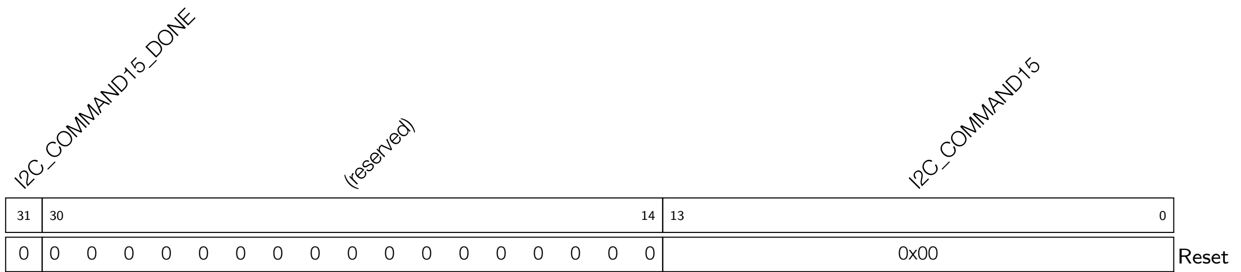
Register 25.40: I2C_COMD14_REG (0x0090)

I2C_COMMAND14_DONE	(reserved)	I2C_COMMAND14	
31	30 ... 14	13 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

I2C_COMMAND14 This is the content of command 14. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND14_DONE When command 14 is done in I2C Master mode, this bit changes to high level. (R/W)

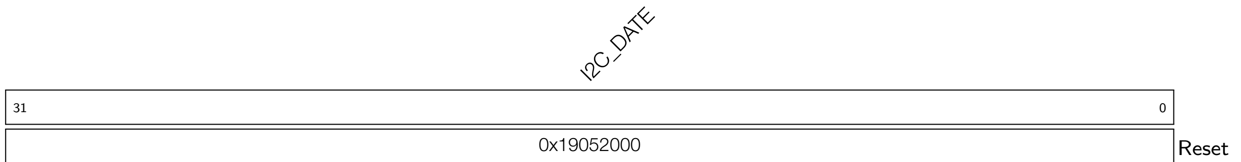
Register 25.41: I2C_COMD15_REG (0x0094)



I2C_COMMAND15 This is the content of command 15. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND15_DONE When command 15 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 25.42: I2C_DATE_REG (0x00F8)



I2C_DATE This is the the version control register. (R/W)

26. I2S Controller (I2S)

26.1 Overview

I2S bus provides a flexible communication interface for streaming digital data in multimedia applications, especially in digital audio applications. ESP32-S2 has a built-in I2S interface, i.e. I2S*0*.

I2S standard bus defines three signals: a bit clock signal (BCK), a channel/word select signal (WS), and a serial data signal (SD). A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication. The I2S module on ESP32-S2 provides separate transmit (TX) and receive (RX) units for high performance.

In addition to standard I2S, the I2S module supports LCD and camera operation modes which transfer data over a parallel bus.

26.2 System Diagram

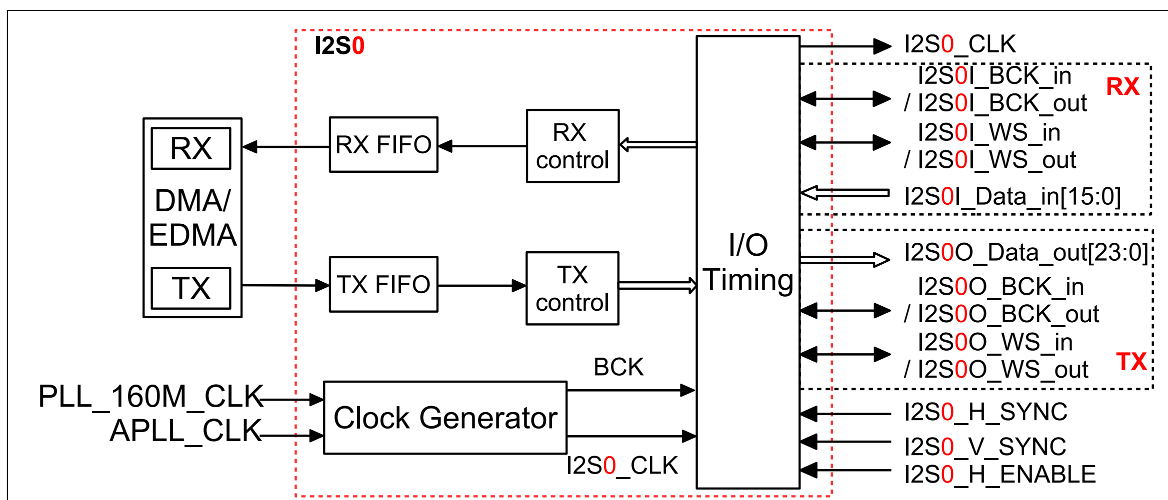


Figure 26-1. ESP32-S2 I2S System Diagram

Figure 26-1 shows the structure of ESP32-S2 I2S module, consisting of a TX unit (TX control), an RX unit (RX control), an input and output timing unit (I/O Timing), a clock divider (Clock Generator), a TX FIFO, and an RX FIFO. Both the TX unit and the RX unit have a 64 x 32-bit FIFO. I2S module supports direct access (DMA) to internal memory and external RAM, see Chapter DMA Controller. Please ensure the consistency between cache and DMA when accessing external RAM. I2S module can work in multiple modes: I2S master/slave transmitter and receiver, LCD master transmitter, and camera slave receiver.

In I2S mode, both the TX unit and the RX unit have a three-line interface, including a bit clock line (BCK), a word select line (WS), and a serial data line (SD). The SD line of TX unit is fixed as output, and the SD line of RX unit as input. BCK and WS signal lines for TX unit and RX unit can be configured as master output mode and slave input mode.

In LCD master transmitting mode, I2S*0*O_BCK_out works as LCD pixel clock output, and I2S*0*O_Data_out [23:0] as LCD parallel output data bus. A wide variety of LCD formats and bit widths are supported in this mode.

In camera slave receiving mode, the lines perform the following roles:

- I2S*0*I_V_SYNC: camera frame rate input

- I2S0I_H_SYNC: horizontal scanning frequency input
- I2S0I_H_ENABLE: horizontal scanning enable line
- I2S0I_WS_in: pixel clock input
- I2S0I_Data_in[15:0]: camera parallel input data bus

The signal bus of I2S module is shown at the right part of Figure 26-1. The naming of these signals in RX and TX units follows the pattern: I2S0A_B_C, such as I2S0I_BCK_in.

- "A": direction of data bus signal
  - "I": input, receiving
  - "O": output, transmitting
- "B": signal function
  - BCK
  - WS
  - SD
- "C": signal direction
  - "in": input signal into I2S module
  - "out": output signal from I2S module

For a detailed description of I2S signal bus, please refer to Table 149.

Table 149: I2S Signal Description

<table>
<tr><th>Signal</th><th>Direction</th><th>Function</th></tr>
<tr><td>I2S0I_BCK_in</td><td>Input</td><td>In I2S slave mode, input BCK signal for RX unit.</td></tr>
<tr><td>I2S0I_BCK_out</td><td>Output</td><td>In I2S master mode, output BCK signal for RX unit.</td></tr>
<tr><td>I2S0I_WS_in</td><td>Input</td><td>In I2S slave mode, input WS signal for RX unit.</td></tr>
<tr><td>I2S0I_WS_out</td><td>Output</td><td>In I2S master mode, output WS signal for RX unit.</td></tr>
<tr><td>I2S0I_Data_in</td><td>Input</td><td>In I2S mode, I2S0I_Data_in works as the serial input bus. In camera mode, I2S0I_Data_in works as the parallel input bus, and the bit width (up to 16) of data line is configurable according to its data transfer format.</td></tr>
<tr><td>I2S0O_Data_out</td><td>Output</td><td>In I2S mode, I2S0O_Data_out is the serial output line for I2S. In LCD mode, works as a parallel output data line. Its bit width (up to 24) is configurable according to the data transfer format.</td></tr>
<tr><td>I2S0O_BCK_in</td><td>Input</td><td>In I2S slave mode, input BCK signal for TX unit.</td></tr>
<tr><td>I2S0O_BCK_out</td><td>Output</td><td>In I2S master mode, output BCK signal for TX unit.</td></tr>
<tr><td>I2S0O_WS_in</td><td>Input</td><td>In I2S slave mode, input WS signal for TX unit.</td></tr>
<tr><td>I2S0O_WS_out</td><td>Output</td><td>In I2S master mode, output WS signal for TX unit.</td></tr>
<tr><td>I2S0_CLK</td><td>Output</td><td>Work as a clock source for peripherals. Note: this signal corresponds to clk_i2s_mux in Table GPIO Matrix.</td></tr>
<tr><td>I2S0I_H_SYNC</td><td rowspan="3">Input</td><td rowspan="3">In camera mode, input HSYNC, VSYNC, and HREF signals, see Figure 26-15.</td></tr>
<tr><td>I2S0I_V_SYNC</td></tr>
<tr><td>I2S0I_H_ENABLE</td></tr>
</table>

Note:

1. All I2S signals must be mapped to the chip's pad via GPIO matrix, see Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*.

2. For input/output signal with a bit width of *N* in LCD/Camera mode, I2S*0*I_Data_in[*N*-1:0] is used as input signal, and I2S*0*O_Data_out[23:23-*N*+1] as output signal. Generally, *N* is 8 ∼ 16 for input signals and 8 ∼ 24 for output signals.

26.3 Features

In I2S mode:

- Support for master mode and slave mode.
- Support for full-duplex and half-duplex communications.
- TX unit and RX unit are independent of each other and can work independently or simultaneously.
- Support for a variety of audio standards:
  - Philips I2S standard
  - PCM standard
  - MSB alignment standard
- High-precision sample clock supports the following frequencies: 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 128 kHz, and 192 kHz. Note that 192 kHz is not supported in 32-bit slave mode.
- Support for 8-/16-/24-/32-bit data communication.

In LCD and camera modes:

- Support to connect to external LCD, and configured as 8- ~ 24-bit parallel output mode.
  - I2S LCD accesses internal memory via DMA.
    - Clock frequency should be less than 40 MHz when LCD data bus is configured as 8- ~ 16-bit parallel output.
    - Clock frequency should be less than 26.7 MHz when LCD data bus is configured as 17- ~ 24-bit parallel output.
  - I2S LCD accesses external RAM via EDMA.
    - Clock frequency should be less than 25 MHz when LCD data bus is configured as 8-bit parallel output.
    - Clock frequency should be less than 12.5 MHz when LCD data bus is configured as 9- ~ 16-bit parallel output.
    - Clock frequency should be less than 6.25 MHz when LCD data bus is configured as 17- ~ 24-bit parallel output.
  - Support for a variety of LCD modes, including MOTO6800, I8080, and etc.
- Support to connect to external camera (i.e. DVP image sensor), and configured as 8- ~ 16-bit parallel input mode.

  - I2S camera accesses internal memory via DMA. Clock frequency should be less than 40 MHz when I2S camera is configured as 8- ~ 16-bit parallel input mode.
  - I2S camera accesses external RAM via EDMA.
    * Clock frequency should be less than 25 MHz when I2S camera is configured as 8-bit parallel input mode.
    * Clock frequency should be less than 12.5 MHz when I2S camera is configured as 9- ~ 16-bit parallel input mode.
- Support to connect to external LCD and camera simultaneously.
  - When accessing internal memory, ensure that the maximum data throughput on the interface is less than DMA total data bandwidth of 80 megabytes per second.
  - When accessing external RAM, ensure that the maximum data throughput on the interface is less than EDMA total data bandwidth of 25 megabytes per second.

I2S interrupts:

- Support for standard I2S interface interrupts.
- Support for I2S DMA interface interrupts.

26.4 Supported Audio Standards

ESP32-S2 I2S supports multiple audio standards.

26.4.1 Philips Standard

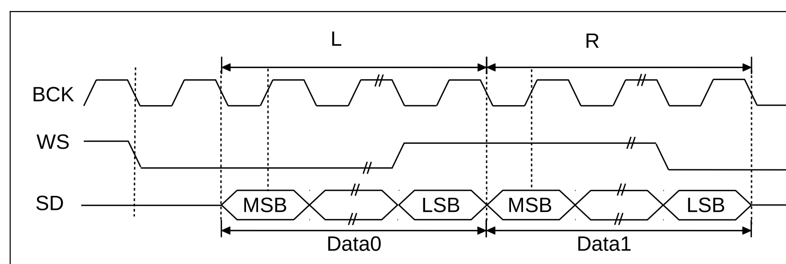


Figure 26-2. Philips Standard

As shown in Figure 26-2, Philips specifications require that WS signal changes one BCK clock cycle earlier than SD signal on BCK falling edge, which means that WS signal keeps valid from a clock cycle earlier before transmitting the first bit of current channel and changes a clock eariler before the end of data transfer in current channel. SD signal line transmits the most significant bit of audio data first. Users can set the bits I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT in register I2S_CONF_REG, respectively, to enable Philips standard when receiving and transmitting data.

26.4.2 MSB Alignment Standard

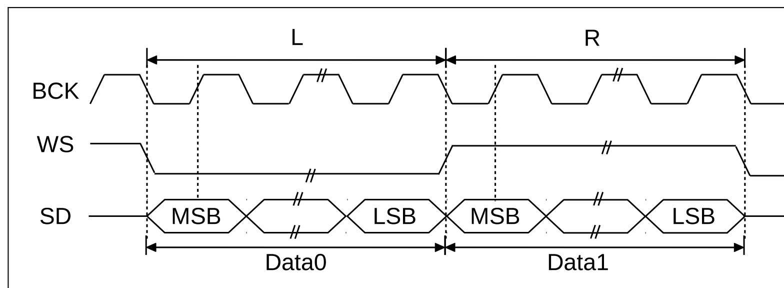


Figure 26-3. MSB Alignment Standard

As shown in Figure 26-3, MSB alignment specifications require WS and SD signals change simultaneously on the falling edge of BCK. The WS signal keeps valid till the end of data transfer in current channel. The SD signal line transmits the most significant bit of audio data first. Users can clear the bits I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT in register I2S_CONF_REG, respectively, to enable MSB alignment standard when receiving and transmitting data.

26.4.3 PCM Standard

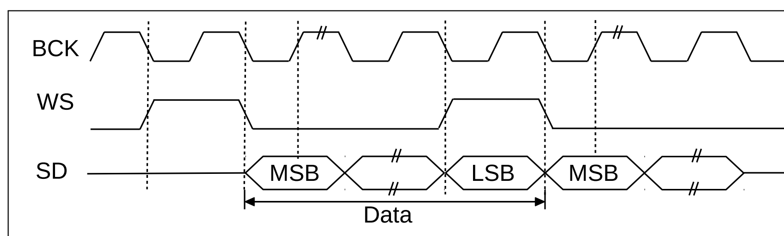


Figure 26-4. PCM Standard

As shown in Figure 26-4, short frame synchronization under PCM standard requires WS signal changes one BCK clock cycle earlier than SD signal on the falling edge of BCK, which means that the WS signal comes valid from a clock cycle earlier before transferring the first bit of current channel and remains unchanged in this BCK clock cycle. The SD signal line transmits the most significant bit of audio data first. Users can set the bits I2S_RX_SHORT_SYNC and I2S_TX_SHORT_SYNC in register I2S_CONF_REG to 1, to enable short frame synchronization mode when receiving and transmitting data.

26.5 I2S Clock

The master clock of I2S module: I2S*0*_CLK is derived from the 160 MHz PLL_160M_CLK or the configurable analog PLL output clock: APLL_CLK. The serial clock (BCK) of I2S module is derived from I2S*0*_CLK, as shown in Figure 26-5. I2S_CLK_SEL[1:0] in register I2S_CLKM_CONF_REG is used to select either PLL_160M_CLK or APLL_CLK as the clock source for I2S*0*, or to disable the clock source of I2S module.

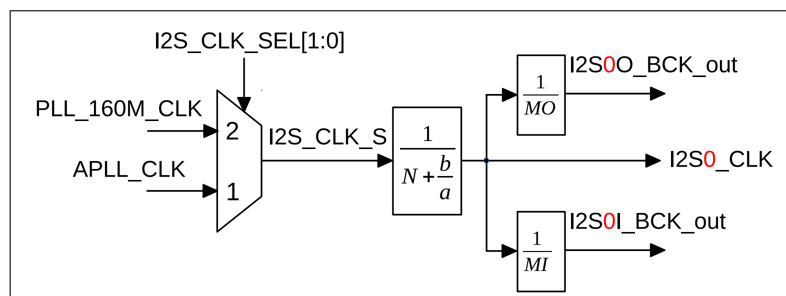


Figure 26-5. I2S Clock

The following formula shows the relation between I2S*0*_CLK frequency f_{i2s} and the divider clock source frequency $f_{\text{I2S_CLK_S}}$:

$$f_{\text{i2s}} = \frac{f_{\text{I2S_CLK_S}}}{\text{N} + \frac{\text{b}}{\text{a}}}$$

N is an integer value between 2 and 256. The value of N corresponds to the value of I2S_CLKM_DIV_NUM[7:0] in register I2S_CLKM_CONF_REG as follows:

- When I2S_CLKM_DIV_NUM[7:0] = 0, N = 256.
- When I2S_CLKM_DIV_NUM[7:0] = 1, N = 2.
- When I2S_CLKM_DIV_NUM[7:0] has any other value, N = I2S_CLKM_DIV_NUM[7:0].

"b" corresponds to the value of I2S_CLKM_DIV_B[5:0], and "a" to the value of I2S_CLKM_DIV_A[5:0]. For integer divider, I2S_CLKM_DIV_A[5:0] and I2S_CLKM_DIV_B[5:0] are cleared. For fractional divider, the value of I2S_CLKM_DIV_B[5:0] should be less than that of I2S_CLKM_DIV_A[5:0].

In master transmitting mode, the serial clock BCK for I2S module is I2S0O_BCK_out, derived from I2S0_CLK. That is:

$$f_{\text{I2S0O_BCK_out}} = \frac{f_{\text{i2s}}}{\text{MO}}$$

"MO" is an integer value between 2 and 128. The value of "MO" corresponds to the value of I2S_TX_BCK_DIV_NUM[5:0] in register I2S_SAMPLE_RATE_CONF_REG as follows:

- When I2S_TX_BCK_DIV_NUM[5:0] = 0, MO = 128.
- When I2S_TX_BCK_DIV_NUM[5:0] is between 2 and 63, MO = I2S_TX_BCK_DIV_NUM[5:0].

Note that I2S_TX_BCK_DIV_NUM[5:0] must not be configured as 1.

In master receiving mode, the serial clock BCK for I2S module is I2S0I_BCK_out, derived from I2S0_CLK. That is:

$$f_{\text{I2S0I_BCK_out}} = \frac{f_{\text{i2s}}}{\text{MI}}$$

"MI" is an integer value between 2 and 128. "MI" corresponds to the value of I2S_RX_BCK_DIV_NUM[5:0] in register I2S_SAMPLE_RATE_CONF_REG as follows:

- When I2S_RX_BCK_DIV_NUM[5:0] = 0, MI = 128.
- When I2S_RX_BCK_DIV_NUM[5:0] is between 2 and 63, MI = I2S_RX_BCK_DIV_NUM[5:0].

Note:

- I2S_RX_BCK_DIV_NUM[5:0] must not be configured as 1.
- Using fractional divider may bring clock jitter. In case that I2S0_CLK and BCK can not be generated from PLL_160_CLK by integer divider, APLL_CLK can be used as clock source. For more information, please refer to Chapter 6 *Reset and Clock*.
- In I2S slave mode, make sure f_{i2s} >= 8 * f_{BCK}.
- I2S module can output I2S0_CLK as the main clock for peripherals.

26.6 I2S Reset

The units in I2S module are reset by different bits in register I2S_CONF_REG.

- I2S TX/RX units: reset by the bits I2S_TX_RESET and I2S_RX_RESET.
- I2S TX/RX FIFO: reset by the bits I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET.

Their reset status are indicated by the status bits: I2S_TX_RESET_ST, I2S_RX_RESET_ST, I2S_TX_FIFO_RESET_ST, and I2S_RX_FIFO_RESET_ST, respectively. Users can read from these status bits to check reset status: 0: reset is completed; 1: reset is not completed.

Reset process is shown as follows:

- Set the bits I2S_TX_RESET, I2S_RX_RESET, I2S_TX_FIFO_RESET, or I2S_RX_FIFO_RESET as needed.
- Wait for I2S_TX_RESET_ST, I2S_RX_RESET_ST, I2S_TX_FIFO_RESET_ST, or I2S_RX_FIFO_RESET_ST to be cleared.

Note: I2S module clock must be configured first before the module and FIFO are reset.

26.7 I2S Master/Slave Mode

ESP32-S2 I2S module can operate as a master or a slave in half-duplex and full-duplex communications, depending on the configuration of the bits I2S_RX_SLAVE_MOD and I2S_TX_SLAVE_MOD in register I2S_CONF_REG.

- Configure transmitting mode via I2S_TX_SLAVE_MOD:
  - 0: master transmitting mode
  - 1: slave transmitting mode
- Configure receiving mode via I2S_RX_SLAVE_MOD:
  - 0: master receiving mode
  - 1: slave receiving mode

26.7.1 Master/Slave Transmitting Mode

- I2S works as master transmitter:
  - Set the bit I2S_TX_START in register I2S_CONF_REG to start transmitting data.
  - I2S keeps driving the clock signal and serial data.
  - If I2S_TX_STOP_EN is set, and all the data in FIFO is transmitted, the master will stop transmitting data.

  - If I2S_TX_STOP_EN is cleared, and all the data in FIFO is transmitted, meanwhile no new data is filled in, the TX unit will keep sending the last data frame.
  - Master stops sending data when the bit I2S_TX_START is cleared.
- I2S works as slave transmitter:
  - Set the bit I2S_TX_START.
  - Wait for the master BCK clock to enable a transmit operation.
  - If I2S_TX_STOP_EN is set, and all the data in FIFO is transmitted, the slave will keep sending 0, till the master stops providing BCK signal.
  - If I2S_TX_STOP_EN is cleared, TX unit will keep sending the last frame when all the data in FIFO is sent and no new data is filled in.
  - Slave stops sending data when the bit I2S_TX_START is cleared or there is no BCK clock in.

26.7.2 Master/Slave Receiving Mode

- I2S works as master receiver:
  - Set the bit I2S_RX_START in register I2S_CONF_REG to start receiving data.
  - RX unit keeps outputting clock signal and sampling input data.
  - Clear the bit I2S_RX_START to stop receiving data.
- I2S works as slave receiver:
  - Set the bit I2S_RX_START.
  - Wait for master BCK signal to start receiving data.

26.8 Transmitting Data

I2S module applies the same way to control TX data format for various audio standards described in Section 26.4. The following part uses MSB alignment standard as an example to illustrate how I2S works in TX mode. I2S transfers data with internal memory via DMA, and with external RAM via EDMA.

I2S module transmits data in three steps:

- Read data from internal memory and write it to TX FIFO.
- Read data to send from TX FIFO and align the data to 64 bits to get prepared for the data transfer in left and right channels.
- Adjust output data pattern:
  - In I2S mode, clock out the data serially.
  - In LCD mode, convert the data to a bit-width fixed stream, and clock out the stream parallelly.

Users can configure TX data format via I2S_TX_BITS_MOD[5:0], I2S_TX_BIG_ENDIAN, I2S_TX_MSB_RIGHT, I2S_TX_DMA_EQUAL, I2S_TX_CHAN_MOD[2:0], and I2S_TX_RIGHT_FIRST.

TX unit transmits data from left channel by default when WS = 0, and from right channel when WS = 1. For subsequent description, suppose that the first four data to send are D0 (indicating low addresses or low bits), D1, D2, and D3 (indicating high addresses or high bits). The left channel data is stored at low addresses or low bits (e.g. D0, D2), and the right channel data at high addresses or high bits (e.g. D1, D3).

26.8.1 Data Transmitting When I2S_TX_DMA_EQUAL = 0

When I2S_TX_DMA_EQUAL = 0, data in left channel is different from that in right channel as shown in Figure 26-6. In this Figure, $D_n^{'}$ (n: 0∼3) is the result of Dn (n: 0∼3) after processed by I2S_TX_BIG_ENDIAN. $D_n^{''}$ (n: 0∼3) is the result of $D_n^{'}$ (n: 0∼3) after processed by I2S_TX_CHAN_MOD[2:0], corresponding to TX Data2 in the dashed box. “Single” here represents the value of I2S_CONF_SIGLE_DATA_REG [31:0], equal to the low 8, 16, 24, or 32 bits of the register, when I2S_TX_BITS_MOD[5:0] is set to 8, 16, 24, or 32.

I2S_TX_BIG_ENDIAN is used to set the internal endianness of the data to send. Assume the bit width of the TX data in Figure 26-6 is 32/24/16/8 bits, and their corresponding D0 is {B3, B2, B1, B0}, {B2, B1, B0}, {B1, B0}, and B0. “B” here means byte. The relation between $D_n^{'}$ (n: 0∼3) and Dn (n: 0∼3) is shown in Table 150.

Table 150: Endianness Mode of TX Data

I2S_TX_BITS_MOD[5:0]	I2S_TX_BIG_ENDIAN	$D_n^{'}$
32	0	{B3 B2 B1 B0}
	1	{B0 B1 B2 B3}
24	0	{B2 B1 B0}
	1	{B0 B1 B2}
16	0	{B1 B0}
	1	{B0 B1}
8	-	B0

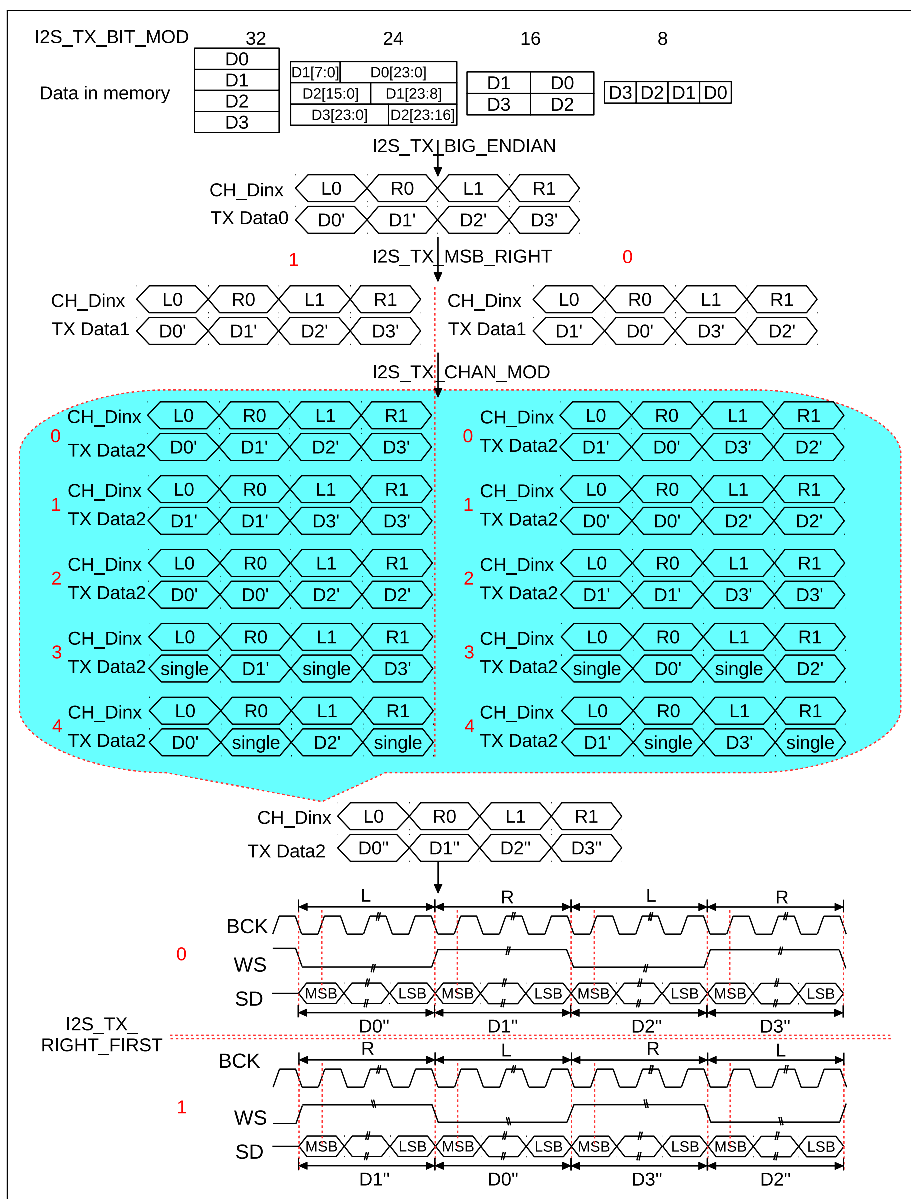


Figure 26-6. ESP32-S2 I2S Data Transmitting Flow When I2S_TX_DMA_EQUAL = 0

Users can configure I2S_TX_MSB_RIGHT to decide data of which channel will be stored in MSB of TX FIFO. I2S_TX_MSB_RIGHT together with I2S_TX_CHAN_MOD[2:0] can configure multiple data formats for right and left channels. I2S_TX_RIGHT_FIRST is used to configure data of which channel, right channel or left channel, should be sent first.

When I2S_TX_DMA_EQUAL = 0, all supported TX channel mode are listed in Table 151. If I2S_TX_CHAN_MOD[2:0] = 0 and I2S_TX_DMA_EQUAL = 0, the output data format is shown in Figure 26-7.

Table 151: TX Channel Mode When I2S_TX_DMA_EQUAL = 0

I2S_TX_CHAN_MOD[2:0]	Channel Mode	Description
0	Dual channel mode	I2S_TX_MSB_RIGHT = 0, left channel transmits high addresses/bits data. Right channel transmits low addresses/bits data. I2S_TX_MSB_RIGHT = 1, left channel transmits low addresses/bits data. Right channel transmits high addresses/bits data.
1	Mono mode	I2S_TX_MSB_RIGHT = 0, both the left and right channels transmit low addresses/bits data. I2S_TX_MSB_RIGHT = 1, both the left and right channels transmit high addresses/bits data.
2	Mono mode	I2S_TX_MSB_RIGHT = 0, both the left and right channels transmit high addresses/bits data. I2S_TX_MSB_RIGHT = 1, both the left and right channels transmit low addresses/bits data.
3	Mono mode	I2S_TX_MSB_RIGHT = 0, left channel transmits the constant: single[31:0]. Right channel transmits low addresses/bits data. I2S_TX_MSB_RIGHT = 1, left channel transmits the constant: single[31:0]. Right channel transmits high addresses/bits data.
4	Mono mode	I2S_TX_MSB_RIGHT = 0, left channel transmits high addresses/bits data. Right channel transmits the constant: single[31:0]. I2S_TX_MSB_RIGHT = 1, left channel transmits low addresses/bits data. Right channel transmits the constant: single[31:0].

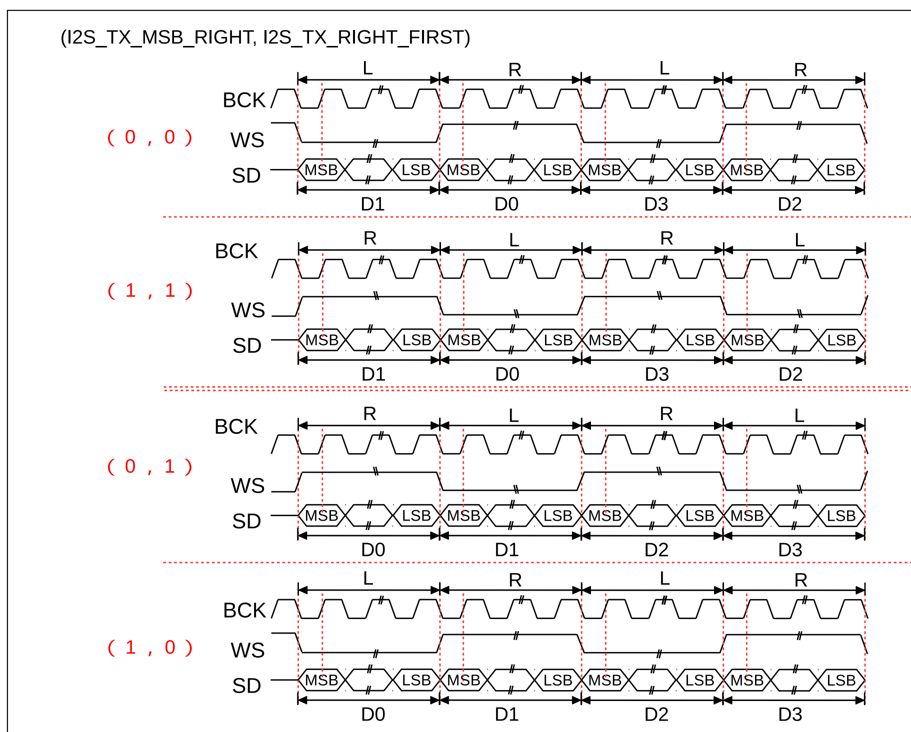


Figure 26-7. I2S Output Format When I2S_TX_CHAN_MOD[2:0] = 0 and I2S_TX_DMA_EQUAL = 0

26.8.2 Data Transmitting When I2S_TX_DMA_EQUAL = 1

When I2S_TX_DMA_EQUAL = 1, data in left channel is equal to that in right channel. In such situation, TX unit only transmits mono data. I2S_TX_MSB_RIGHT can not control data structure. Users can configure I2S_TX_CHAN_MOD[2:0] to output the constant: I2S_CONF_SIGLE_DATA_REG[31:0]. The TX data format is shown in Figure 26-8 and in Table 152.

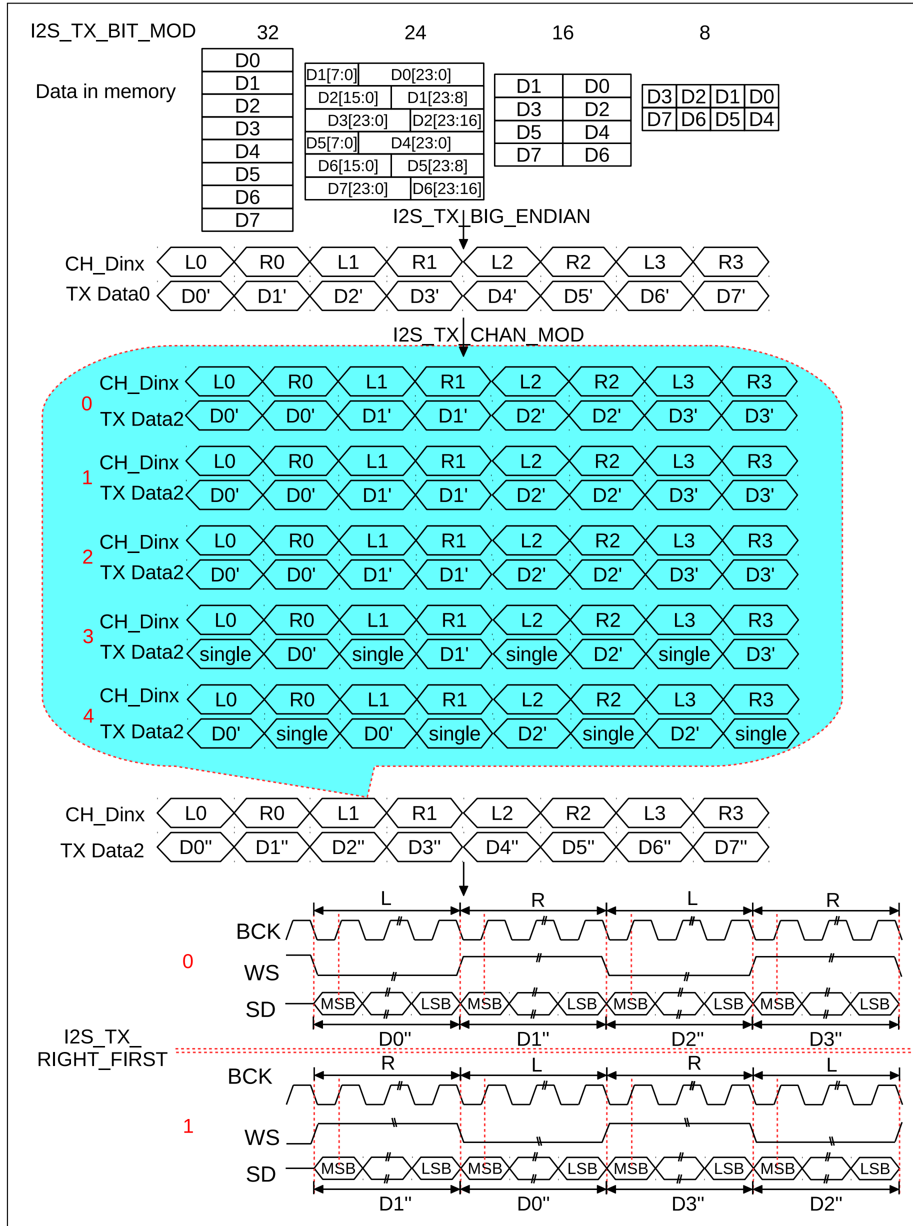


Figure 26-8. I2S TX Data When I2S_TX_DMA_EQUAL = 1

Table 152: TX Channel Mode When I2S_TX_DMA_EQUAL = 1

I2S_TX_CHAN_MOD[2:0]	Channel Mode	Description
0/1/2	Mono mode	Both the left channel and the right channel transmit the same valid data.
3	Mono mode	The right channel transmits valid data, and the left channel transmits the constant: single[31:0].
4	Mono mode	The left channel transmits valid data, and the right channel transmits the constant: single[31:0].

26.8.3 Configuring I2S as TX Mode

Follow the steps below to configure I2S as TX mode via software:

1. Configure clock as described in Section 26.5.
2. Configure signal pins according to Table 149.
3. Clear the bit I2S_LCD_EN in register I2S_CONF2_REG, to enable I2S mode.
4. Configure the bit I2S_TX_SLAVE_MOD in register I2S_CONF_REG, to select the mode needed.
   - 0: master transmitting mode
   - 1: slave transmitting mode
5. Configure I2S_TX_BITS_MOD[5:0], I2S_TX_BIG_ENDIAN, I2S_TX_MSB_RIGHT, I2S_TX_DMA_EQUAL, I2S_TX_CHAN_MOD[2:0], and I2S_TX_RIGHT_FIRST as described in Section 26.8, to set TX data mode correctly.
6. Set the bit I2S_DSCR_EN in register I2S_FIFO_CONF_REG, to enable I2S DMA.
7. Reset TX unit and TX FIFO as described in Section 26.6.
8. Enable corresponding interrupts, see Section 26.12.
9. Configure DMA outlink, and set I2S_OUTLINK_START to start DMA.
10. Set I2S_TX_STOP_EN bit if needed. For more information, please refer to Section 26.7.1.
11. Start transmitting data:
    - In master mode, wait till I2S slave gets ready, then set I2S_TX_START to start transmitting data.
    - In slave mode, set the bit I2S_TX_START. When the I2S master supplies BCK and WS signals, start transmitting data.
12. Wait for the interrupt signals set in Step 8, or check whether the transfer is completed by querying the register I2S_TX_IDLE:
    - 0: transmitter is working.
    - 1: transmitter is in idle.
13. Clear I2S_TX_START to stop data transfer.

26.9 Receiving Data

I2S module applies the same way to control RX data format for various audio standards described in Section 26.4. The following part uses MSB alignment standard as an example to illustrate how I2S works in RX mode.

Suppose that the first four received data are D0 (low addresses or bits), D1, D2, and D3 (high addresses or bits). RX unit receives data according to WS signal:

- WS = 0, receive left channel data.
- WS = 1, receive right channel data.

RX data format is controlled by I2S_RX_BITS_MOD[5:0], I2S_RX_BIG_ENDIAN, I2S_RX_MSB_RIGHT, I2S_RX_DMA_EQUAL, I2S_RX_CHAN_MOD[1:0], and I2S_RX_RIGHT_FIRST.

26.9.1 Data Receiving When I2S_RX_DMA_EQUAL = 0

When I2S_RX_DMA_EQUAL = 0, data in left channel is different from that in right channel. In such situation, I2S RX module enables dual-channel receiving mode. RX data format is shown in Figure 26-9, which is not controlled by I2S_RX_CHAN_MOD[1:0].

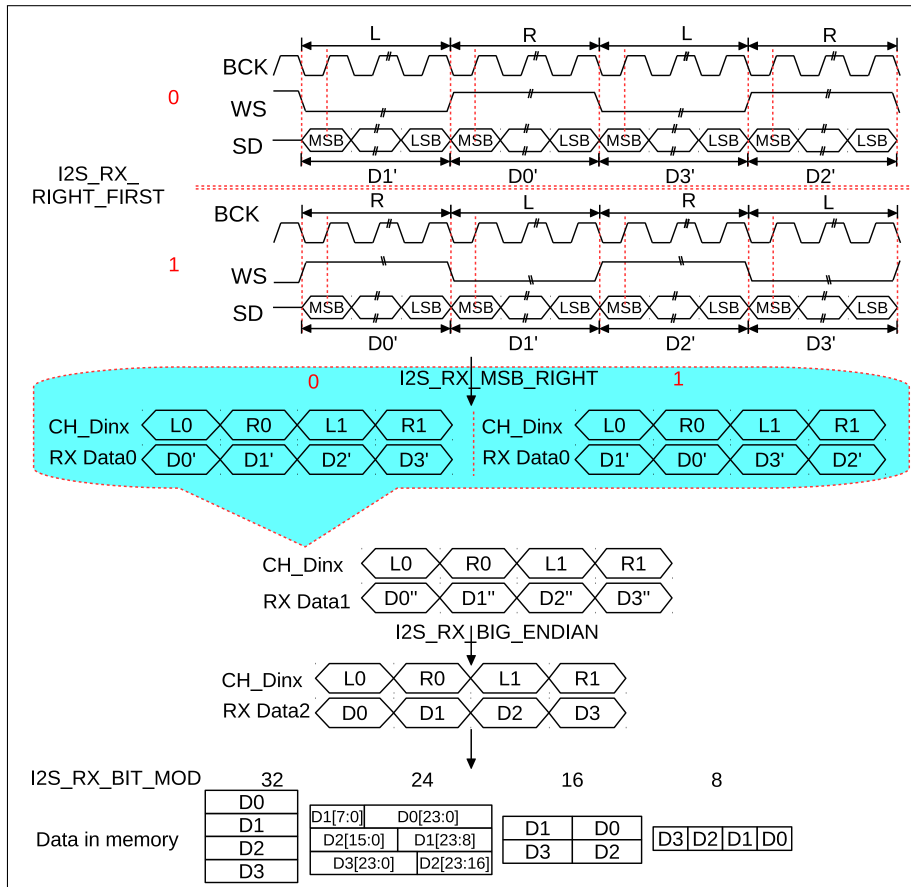


Figure 26-9. I2S RX Data When I2S_RX_DMA_EQUAL = 0

In Figure 26-9, D''_n (n: 0 ~3) is the result of D'_n (n: 0~3) after processed by I2S_RX_MSB_RIGHT. D_n (n: 0 ~3) is the result of D''_n (n: 0~3) after processed by I2S_RX_BIG_ENDIAN. I2S_RX_BIG_ENDIAN is used to modify the endianness of the RX data, see Table 150.

According to Figure 26-9, when I2S_RX_DMA_EQUAL is cleared, the waveform of I2S RX data is shown in Figure

26-10. For the specific control rules, please see Table 153.

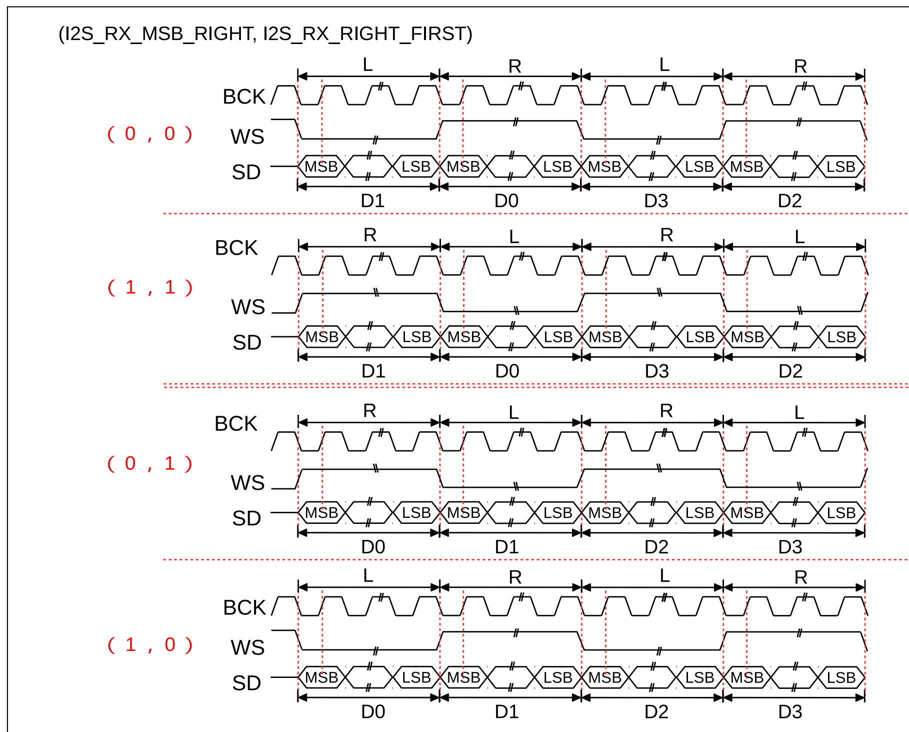


Figure 26-10. I2S RX Data When I2S_RX_DMA_EQUAL = 0

Table 153: RX Channel Mode When I2S_RX_DMA_EQUAL = 0

I2S_RX_MSB_RIGHT	Channel Mode	Description
0	Dual channel mode	I2S_RX_RIGHT_FIRST sets data receiving order in frames. Right channel data is stored to the low address of ESP32-S2 memory, and the left channel data to the high address.
1	Dual channel mode	I2S_RX_RIGHT_FIRST sets data receiving order in frames. Left channel data is stored to the low address of ESP32-S2 memory, and the right channel data to the high address.

26.9.2 Data Receiving When I2S_RX_DMA_EQUAL = 1

Set I2S_RX_DMA_EQUAL to 1, data in left channel is equal to that in right channel. In such situation, RX unit enables single channel mode (mono mode) to receive data. The RX data format is shown in Figure 26-11 and in Table 154.

Table 154: RX Channel Mode When I2S_RX_DMA_EQUAL = 1

I2S_RX_MSB_RIGHT	I2S_RX_CHAN_MOD[2:0]	Channel Mode	Description
0	0/3	Mono mode	Not used
	1	Mono mode	Only store right channel data
	2	Mono mode	Only store left channel data
1	0/3	Mono mode	Not used
	1	Mono mode	Only store left channel data
	2	Mono mode	Only store right channel data

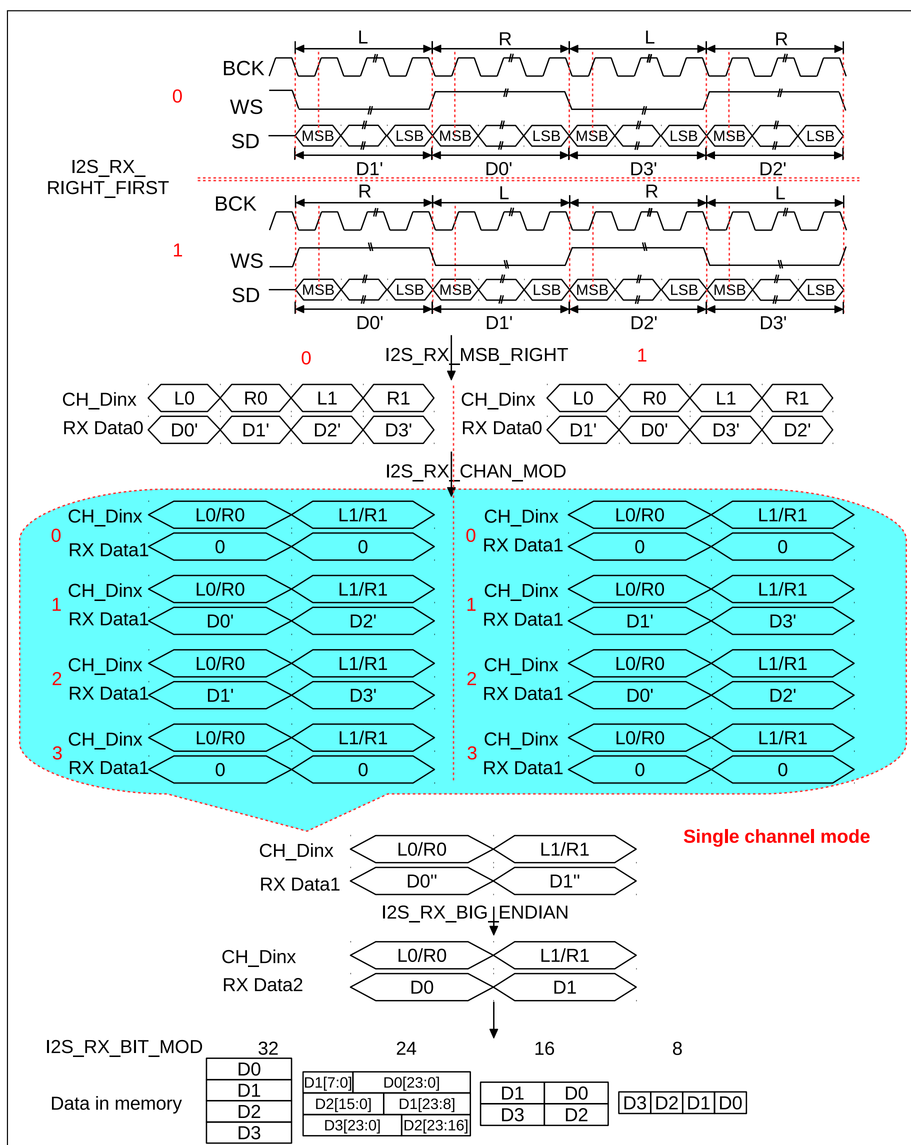


Figure 26-11. ESP32-S2 I2S RX Data When I2S_RX_DMA_EQUAL = 1

26.9.3 Configuring I2S as RX Mode

Follow the steps below to configure I2S as RX mode via software:

1. Configure clock as described in Section 26.5.
2. Configure signal pins according to Table 149.
3. Clear the bits I2S_LCD_EN and I2S_CAMERA_EN in register I2S_CONF2_REG, to enable I2S mode.
4. Configure the bit I2S_RX_SLAVE_MOD in register I2S_CONF_REG to select mode needed:
   - 0: master receiving mode
   - 1: slave receiving mode
5. Configure I2S_RX_DMA_EQUAL, I2S_RX_BITS_MOD[5:0], I2S_RX_BIG_ENDIAN, I2S_RX_MSB_RIGHT, I2S_RX_CHAN_MOD[2:0], and I2S_RX_RIGHT_FIRST to select desired RX data mode as described in Section 26.9.
6. Set the bit I2S_DSCR_EN in register I2S_FIFO_CONF_REG, to enable I2S DMA.

7. Reset RX unit and its FIFO as described in Section 26.6.
8. Enable corresponding interrupts as described in Section 26.12.
9. Configure DMA inlink, and set the length of RX data in register I2S_RXEOF_NUM_REG. Then set I2S_INLINK_START to start DMA.
10. Start receiving data:
    - In master mode, when the slave is ready, set I2S_RX_START to start receiving data.
    - In slave mode, set I2S_RX_START to start receiving data when get BCK and WS signals from the master.
11. Receive data and store the data to the specified address of ESP32-S2 memory. Then corresponding interrupts set in Step 8 will be generated.

26.10 LCD Master Transmitting Mode

26.10.1 Overview

As shown in Figure 26-12, LCD WR signal is connected to I2S WS signal. The data width can be set to 8/16/24 bits (parallel), depending on the configuration of I2S_TX_BITS_MOD[5:0].

In LCD mode, WS clock frequency is:

$$f_{\mathrm{WS}} = \frac{f_{\mathrm{i2s}}}{\mathrm{W} * 2}$$

W is an integer in the range 1 to 64. W corresponds to the value of I2S_TX_BCK_DIV_NUM[5:0] in register I2S_SAMPLE_RATE_CONF_REG as follows.

- When I2S_TX_BCK_DIV_NUM[5:0] = 0, W = 64.
- When I2S_TX_BCK_DIV_NUM[5:0] is set to other values, W = I2S_TX_BCK_DIV_NUM[5:0].

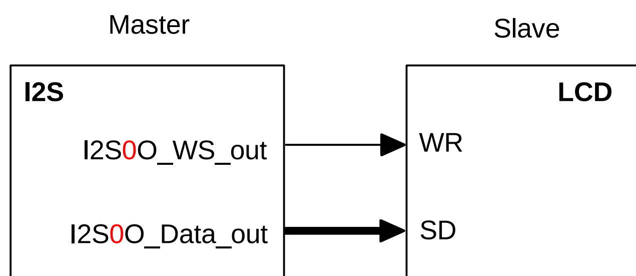


Figure 26-12. LCD Master Transmitting Mode

26.10.2 Configure I2S as LCD Master Transmitting Mode

In LCD master transmitting mode, I2S supports the data frame format as shown in 26-13. Follow the steps

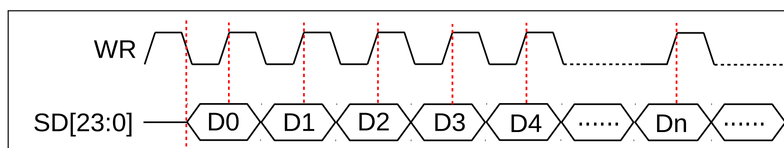


Figure 26-13. Data Frame Format 1 in LCD Master Transmitting Mode

below to configure I2S module as LCD master transmitter via software:

1. Configure clock as described in Section 26.10.1.
2. Configure signal pins according to Table 149 and Figure 26-12.
3. Set the bit I2S_LCD_EN in register I2S_CONF2_REG, and clear the bit I2S_TX_SLAVE_MOD in register I2S_CONF_REG, to enable LCD master transmitting mode.
4. Set I2S_TX_DMA_EQUAL, then clear I2S_TX_RIGHT_FIRST, I2S_LCD_TX_WRX2_EN, I2S_LCD_TX_SDX2_EN, and I2S_TX_CHAN_MOD[2:0].
5. Configure I2S_TX_BITS_MOD[5:0] and I2S_TX_BIG_ENDIAN as described in Section 26.8 to select desired TX data mode.
6. Set the bit I2S_DSCR_EN in register I2S_FIFO_CONF_REG, to enable I2S DMA.
7. Reset TX unit and TX FIFO according to Section 26.6.
8. Enable corresponding interrupts, see Section 26.12.
9. Configure DMA outlink, and set I2S_OUTLINK_START to start DMA.
10. Set I2S_TX_STOP_EN bit.
11. Wait till LCD slave gets ready, then set I2S_TX_START to start transmitting data.
12. Wait for the interrupt signals set in Step 8, or check whether the transfer is completed by querying the register I2S_TX_IDLE:
    - 0: transmitter is working.
    - 1: transmitter is in idle.
13. Clear I2S_TX_START, to stop transmitting data.

In LCD master transmitting mode, the data frame format shown in 26-14 is also supported. The only difference in software configuration with the format as shown in Figure 26-13 is that I2S_LCD_TX_SDX2_EN is set.

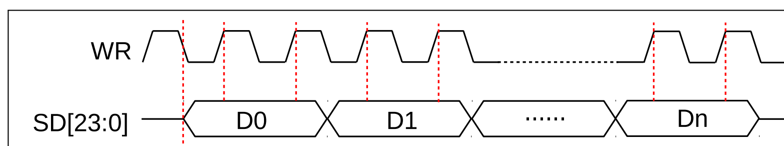


Figure 26-14. Data Frame Format 2 in LCD Master Transmitting Mode

26.11 Camera Slave Receiving Mode

26.11.1 Overview

ESP32-S2 I2S can be configured as camera slave receiving mode for high-speed data transfer with external camera modules. In this mode, I2S module works as slave receiver with a 16-channel data signal bus I2S*0*I_Data_in[15:0], and three more signals: I2S*0*I_H_SYNC, I2S*0*I_V_SYNC, and I2S*0*I_H_ENABLE, see Figure 26-15.

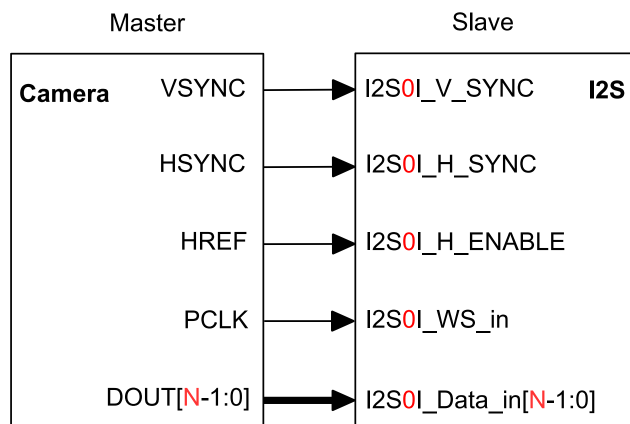


Figure 26-15. Camera Slave Receiving Mode

Note:
The bit width of input data is N, which can be 8 ~ 16 bits.

26.11.2 Configure I2S as Camera Slave Receiving Mode

Follow the steps below to configure I2S as camera slave receiver via software:

1. Configure clock as described in Section 26.5. Note that the clock frequency of I2S module: f_{i2s} should be equal to or larger than twice of the frequency of PCLK clock (input from I2S0I_WS_in pin).
2. Configure signal pins according to Table 149 and Figure 26-15.
3. Set the bits I2S_LCD_EN and I2S_CAMERA_EN in register I2S_CONF2_REG, to enable camera mode.
4. Set the bit I2S_RX_SLAVE_MOD in register I2S_CONF_REG, to enable slave mode.
5. Set the bit I2S_RX_DMA_EQUAL, then configure I2S_RX_BITS_MOD[5:0], I2S_RX_BIG_ENDIAN, I2S_RX_MSB_RIGHT, I2S_RX_CHAN_MOD[2:0], and I2S_RX_RIGHT_FIRST according to Section 26.9, to select desired RX data format.
6. Set the bit I2S_DSCR_EN in register I2S_FIFO_CONF_REG.
7. Reset RX unit and its FIFO according to Section 26.6. Set I2S_CAM_SYNC_FIFO_RESET bit and then clear it.
8. Enable corresponding interrupt as described in Section 26.12.
9. Configure DMA inlink, and set the length of RX data in register I2S_RXEOF_NUM_REG. Then set I2S_INLINK_START to start DMA.
10. Set I2S_RX_START, and wait for $transmission_start$ signal.
11. Receive data and store the data to the specified address of ESP32-S2 memory. Then corresponding interrupts set in Step 8 will be generated.

When I2S0I_V_SYNC is detected on rising edge, I2S0I_H_SYNC, I2S0I_H_ENABLE are held high, and I2S0I_V_SYNC is held low, I2S module starts receiving data:

$$transmission_start = ((I2S0I_V_SYNC == 0)\&\&(I2S0I_H_SYNC == 1)\&\&(I2S0I_H_ENABLE == 1)$$

In case that the signals do not satisfy the requirements defined above, users may reverse the signals through GPIO matrix. For more information, please refer to 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)*.

To avoid noise on I2S*0*I_V_SYNC signal, set I2S_VSYNC_FILTER_EN to enable filter for I2S*0*I_V_SYNC signal. The filter samples input signals continuously, and will detect the signals which remain unchanged for a continuous I2S_VSYNC_FILTER_THRES PCLK clock cycles as valid, otherwise the signals will be detected as invalid. Only the valid signals can pass through this filter. Therefore, the filter will remove pulses with a length of less than I2S_VSYNC_FILTER_THRES PCLK cycles from I2S*0*I_V_SYNC signal line.

Camera module provides at least 8 PCLK rising-edge clock signals when I2S*0*I_V_SYNC is held high, to ensure that I2S samples the rising-edge of I2S*0*I_V_SYNC signal correctly.

26.12 I2S Interrupts

26.12.1 FIFO Interrupts

- I2S_TX_HUNG_INT: Triggered when transmitting data is timed out.
- I2S_RX_HUNG_INT: Triggered when receiving data is timed out.
- I2S_TX_REMPTY_INT: Triggered when TX FIFO is empty.
- I2S_TX_WFULL_INT: Triggered when TX FIFO is full.
- I2S_RX_REMPTY_INT: Triggered when RX FIFO is empty.
- I2S_RX_WFULL_INT: Triggered when RX FIFO is full.
- I2S_TX_PUT_DATA_INT: Triggered when the left and right channel data number in TX FIFO is smaller than the value of I2S_TX_DATA_NUM[5:0]. (TX FIFO is almost empty.)
- I2S_RX_TAKE_DATA_INT: Triggered when the left and right channel data number in RX FIFO is larger than the value of I2S_RX_DATA_NUM[5:0]. (RX FIFO is almost full.)
- I2S_V_SYNC_INT: Triggered when I2S camera transmission_start is valid and a new rising edge of I2S*0*I_V_SYNC is detected.

26.12.2 DMA Interrupts

- I2S_OUT_TOTAL_EOF_INT: Triggered when all outlinks are used up.
- I2S_IN_DSCR_EMPTY_INT: Triggered when there are no valid inlinks left.
- I2S_OUT_DSCR_ERR_INT: Triggered when encounter invalid outlink descriptors.
- I2S_IN_DSCR_ERR_INT: Triggered when encounter invalid inlink descriptors.
- I2S_OUT_EOF_INT: Triggered when outlink has finished sending a packet.
- I2S_OUT_DONE_INT: Triggered when all transmitted and buffered data have been read.
- I2S_IN_SUC_EOF_INT: Triggered when all data have been received.
- I2S_IN_DONE_INT: Triggered when current inlink descriptor is handled.

26.13 Base Address

Users can access I2S with two base addresses, which can be seen in Table 155. For more information about accessing peripherals from different buses please see Chapter System and Memory.

Table 155: I2S Register Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F40F000
PeriBUS2	0x6000F000

26.14 Register Summary

Name	Description	Address	Access
Configuration Registers			
I2S_CONF_REG	I2S configuration register	0x0008	varies
I2S_FIFO_CONF_REG	I2S FIFO configuration register	0x0020	R/W
I2S_CONF_SIGLE_DATA_REG	Constant single channel data	0x0028	R/W
I2S_CONF_CHAN_REG	I2S channel configuration register	0x002C	R/W
I2S_LC_HUNG_CONF_REG	I2S Hung configuration register	0x0074	R/W
I2S_CONF1 _REG	I2S configuration register 1	0x00A0	R/W
I2S_PD_CONF_REG	I2S power-down configuration register	0x00A4	R/W
I2S_CONF2_REG	I2S configuration register 2	0x00A8	R/W
Interrupt Registers			
I2S_INT_RAW_REG	Raw interrupt status	0x000C	RO
I2S_INT_ST_REG	Masked interrupt status	0x0010	RO
I2S_INT_ENA_REG	Interrupt enable bits	0x0014	R/W
I2S_INT_CLR_REG	Interrupt clear bits	0x0018	WO
Timing Register			
I2S_TIMING_REG	I2S timing register	0x001C	R/W
DMA Registers			
I2S_RXEOF_NUM_REG	I2S DMA RX EOF data length	0x0024	R/W
I2S_OUT_LINK_REG	I2S DMA TX configuration register	0x0030	R/W
I2S_IN_LINK_REG	I2S DMA RX configuration register	0x0034	R/W
I2S_OUT_EOF_DES_ADDR_REG	Address of outlink descriptor that produces EOF	0x0038	RO
I2S_IN_EOF_DES_ADDR_REG	Address of inlink descriptor that produces EOF	0x003C	RO
I2S_OUT_EOF_BFR_DES_ADDR_REG	Address of buffer relative to the outlink descriptor that produces EOF	0x0040	RO
I2S_INLINK_DSCR_REG	Address of current inlink descriptor	0x0048	RO
I2S_INLINK_DSCR_BF0_REG	Address of next inlink descriptor	0x004C	RO
I2S_INLINK_DSCR_BF1_REG	Address of next inlink data buffer	0x0050	RO
I2S_OUTLINK_DSCR_REG	Address of current outlink descriptor	0x0054	RO
I2S_OUTLINK_DSCR_BF0_REG	Address of next outlink descriptor	0x0058	RO
I2S_OUTLINK_DSCR_BF1_REG	Address of next outlink data buffer	0x005C	RO
I2S_LC_CONF_REG	I2S DMA configuration register	0x0060	R/W
DMA Status Registers			

Name	Description	Address	Access
I2S_LC_STATE0_REG	I2S DMA TX status	0x006C	RO
I2S_LC_STATE1_REG	I2S DMA RX status	0x0070	RO
I2S_STATE_REG	I2S TX status register	0x00BC	RO
Clock and Sample Registers			
I2S_CLKM_CONF_REG	I2S module clock configuration register	0x00AC	R/W
I2S_SAMPLE_RATE_CONF_REG	I2S sample rate register	0x00B0	R/W
Version Register			
I2S_DATE_REG	Version control register	0x00FC	R/W

26.15 Registers

Register 26.1: I2S_CONF_REG (0x0008)

Bit(s)	Field	Reset
31:30	(reserved)	0 0
29	I2S_RX_RESET_ST	0
28	I2S_RX_BIG_ENDIAN	0
27	I2S_TX_BIG_ENDIAN	0
26	I2S_PRE_REQ_EN	0
25	I2S_RX_DMA_EQUAL	0
24	I2S_TX_DMA_EQUAL	0
23	I2S_TX_RESET_ST	0
22	I2S_RX_FIFO_RESET_ST	0
21	I2S_TX_FIFO_RESET_ST	0
20	I2S_SIG_LOOPBACK	0
19	(reserved)	1
18	(reserved)	1
17	I2S_RX_MSB_RIGHT	0
16	I2S_TX_MSB_RIGHT	0
15	I2S_RX_MONO	0
14	I2S_TX_MONO	0
13	I2S_RX_SHORT_SYNC	0
12	I2S_TX_SHORT_SYNC	0
11	I2S_RX_MSB_SHIFT	0
10	I2S_TX_MSB_SHIFT	0
9	I2S_RX_RIGHT_FIRST	1
8	I2S_TX_RIGHT_FIRST	1
7	I2S_RX_SLAVE_MOD	0
6	I2S_TX_SLAVE_MOD	0
5	I2S_RX_START	0
4	I2S_TX_START	0
3	I2S_RX_FIFO_RESET	0
2	I2S_TX_FIFO_RESET	0
1	I2S_RX_RESET	0
0	I2S_TX_RESET	0

I2S_TX_RESET Set this bit to reset transmitter. (WO)

I2S_RX_RESET Set this bit to reset receiver. (WO)

I2S_TX_FIFO_RESET Set this bit to reset TX FIFO. (WO)

I2S_RX_FIFO_RESET Set this bit to reset RX FIFO. (WO)

I2S_TX_START Set this bit to start transmitting data. (R/W)

I2S_RX_START Set this bit to start receiving data. (R/W)

I2S_TX_SLAVE_MOD Set this bit to enable slave transmitter mode. (R/W)

I2S_RX_SLAVE_MOD Set this bit to enable slave receiver mode. (R/W)

I2S_TX_RIGHT_FIRST Set this bit to transmit right channel data first. (R/W)

I2S_RX_RIGHT_FIRST Set this bit to receive right channel data first. (R/W)

I2S_TX_MSB_SHIFT Set this bit to enable transmitter in Phillips standard mode. (R/W)

I2S_RX_MSB_SHIFT Set this bit to enable receiver in Phillips standard mode. (R/W)

I2S_TX_SHORT_SYNC Set this bit to enable transmitter in PCM standard mode. (R/W)

I2S_RX_SHORT_SYNC Set this bit to enable receiver in PCM standard mode. (R/W)

I2S_TX_MONO Set this bit to enable transmitter in mono mode. (R/W)

I2S_RX_MONO Set this bit to enable receiver in mono mode. (R/W)

I2S_TX_MSB_RIGHT Set this bit to place right channel data at the MSB in TX FIFO. (R/W)

I2S_RX_MSB_RIGHT Set this bit to place right channel data at the MSB in RX FIFO. (R/W)

I2S_SIG_LOOPBACK Enable signal loopback mode with transmitter module and receiver module sharing the same WS and BCK signals. (R/W)

Continued on the next page...

Register 26.1: I2S_CONF_REG (0x0008)

Continued from the previous page...

I2S_TX_FIFO_RESET_ST I2S TX FIFO reset status. 1: I2S_TX_FIFO_RESET is not completed. 0: I2S_TX_FIFO_RESET is completed. (RO)

I2S_RX_FIFO_RESET_ST I2S RX FIFO reset status. 1: I2S_RX_FIFO_RESET is not completed. 0: I2S_RX_FIFO_RESET is completed. (RO)

I2S_TX_RESET_ST I2S TX reset status. 1: I2S_TX_RESET is not completed. 0: I2S_TX_RESET is completed. (RO)

I2S_TX_DMA_EQUAL 1: Data in left channel is equal to data in right channel. 1: Data in left channel is not equal to data in right channel. (R/W)

I2S_RX_DMA_EQUAL 1: Data in left channel is equal to data in right channel. 1: Data in left channel is not equal to data in right channel. (R/W)

I2S_PRE_REQ_EN Set this bit to enable I2S to prepare data earlier. (R/W)

I2S_TX_BIG_ENDIAN I2S TX byte endianness. (R/W)

I2S_RX_BIG_ENDIAN I2S RX byte endianness. (R/W)

I2S_RX_RESET_ST I2S RX reset status. 1: I2S_RX_RESET is not completed. 0: I2S_RX_RESET is completed. (RO)

Register 26.2: I2S_FIFO_CONF_REG (0x0020)

	(reserved)	I2S_DSCR_EN	I2S_RX_DATA_NUM	I2S_RX_DATA_NUM
Bits	31 … 13	12	11 … 6	5 … 0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	32	32

I2S_RX_DATA_NUM I2S_RX_TAKE_DATA_INT is triggered when the left and right channel data number in RX FIFO is larger than the value of I2S_RX_DATA_NUM[5:0]. (RX FIFO is almost full threshold.)

I2S_TX_DATA_NUM I2S_TX_PUT_DATA_INT is triggered when the left and right channel data number in TX FIFO is smaller than the value of I2S_TX_DATA_NUM[5:0]. (TX FIFO is almost empty threshold.)

I2S_DSCR_EN Set this bit to enable I2S DMA mode. (R/W)

Register 26.3: I2S_CONF_SIGLE_DATA_REG (0x0028)

I2S_SIGLE_DATA	
31 … 0	
0	Reset

I2S_SIGLE_DATA The right channel or left channel transmits constant value stored in this register according to I2S_TX_CHAN_MOD and I2S_TX_MSB_RIGHT. (R/W)

Register 26.4: I2S_CONF_CHAN_REG (0x002C)

(reserved)	I2S_RX_CHAN_MOD	I2S_TX_CHAN_MOD	
31 … 5	4 … 3	2 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

I2S_TX_CHAN_MOD I2S transmitter channel mode configuration bits. (R/W)

I2S_RX_CHAN_MOD I2S receiver channel mode configuration bits. (R/W)

Register 26.5: I2S_LC_HUNG_CONF_REG (0x0074)

(reserved)	I2S_LC_FIFO_TIMEOUT_ENA	I2S_LC_FIFO_TIMEOUT_SHIFT	I2S_LC_FIFO_TIMEOUT	
31 … 12	11	10 … 8	7 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	0x10	Reset

I2S_LC_FIFO_TIMEOUT I2S_TX_HUNG_INT interrupt or I2S_RX_HUNG_INT interrupt will be triggered when FIFO hung counter is equal to this value. (R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT The bits are used to set the tick counter threshold. The tick counter is clocked by CLK_APB. The tick counter threshold is $88000/2^{I2S_LC_FIFO_TIMEOUT_SHIFT}$. The tick counter is reset when it reaches the threshold. (R/W)

I2S_LC_FIFO_TIMEOUT_ENA The enable bit for FIFO timeout. (R/W)

Register 26.6: I2S_CONF1 _REG (0x00A0)

(reserved)	I2S_TX_STOP_EN	(reserved)	
31 ... 9	8	7 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0	Reset

I2S_TX_STOP_EN Set this bit to stop the output of BCK signal and WS signal when TX FIFO is empty. (R/W)

Register 26.7: I2S_PD_CONF_REG (0x00A4)

(reserved)	I2S_DMA_RAM_CLK_FO	I2S_DMA_RAM_FORCE_PU	I2S_DMA_RAM_FORCE_PD	I2S_PLC_MEM_FORCE_PU	I2S_PLC_MEM_FORCE_PD	I2S_FIFO_FORCE_PU	I2S_FIFO_FORCE_PD	
31 ... 7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	0	1	0	1	0	Reset

I2S_FIFO_FORCE_PD Force FIFO power-down. (R/W)

I2S_FIFO_FORCE_PU Force FIFO power-up. (R/W)

I2S_PLC_MEM_FORCE_PD Force I2S memory power-down. (R/W)

I2S_PLC_MEM_FORCE_PU Force I2S memory power-up. (R/W)

I2S_DMA_RAM_FORCE_PD Force DMA FIFO power-down. (R/W)

I2S_DMA_RAM_FORCE_PU Force DMA FIFO power-up. (R/W)

I2S_DMA_RAM_CLK_FO Set this bit to force on DMA RAM clock. (R/W)

Register 26.8: I2S_CONF2_REG (0x00A8)

(reserved)	I2S_VSYNC_FILTER_THRES	I2S_VSYNC_FILTER_EN	I2S_CAM_CLK_LOOPBACK	I2S_CAM_SYNC_FIFO_RESET	I2S_INTER_VALID_EN	(reserved)	I2S_LCD_EN	(reserved)	I2S_LCD_TX_SDX2_EN	I2S_LCD_TX_WRX2_EN	I2S_CAMERA_EN	
31 ... 14	13 ... 11	10	9	8	7	6	5	4 ... 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	0	0	Reset

I2S_CAMERA_EN Set this bit to enable camera mode. (R/W)

I2S_LCD_TX_WRX2_EN LCD WR double for one datum. (R/W)

I2S_LCD_TX_SDX2_EN Set this bit to duplicate data pairs (Frame Form 2) in LCD mode. (R/W)

I2S_LCD_EN Set this bit to enable LCD mode. (R/W)

I2S_INTER_VALID_EN Set this bit to enable camera VGA reducing-resolution mode: only receive two consecutive cycle data in four consecutive clocks.

I2S_CAM_SYNC_FIFO_RESET Set this bit to reset FIFO in camera mode. (R/W)

I2S_CAM_CLK_LOOPBACK Set this bit to loopback PCLK from I2S0I_WS_out. (R/W)

I2S_VSYNC_FILTER_EN Set this bit to enable I2S VSYNC filter function. (R/W)

I2S_VSYNC_FILTER_THRES Configure the I2S VSYNC filter threshold value. (R/W)

Register 26.9: I2S_INT_RAW_REG (0x000C)

31–18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2S_V_SYNC_INT_RAW	I2S_OUT_TOTAL_EOF_INT_RAW	I2S_IN_DSCR_EMPTY_INT_RAW	I2S_OUT_DSCR_ERR_INT_RAW	I2S_IN_DSCR_ERR_INT_RAW	I2S_OUT_EOF_INT_RAW	I2S_OUT_DONE_INT_RAW	(reserved)	I2S_IN_SUC_EOF_INT_RAW	I2S_IN_DONE_INT_RAW	I2S_TX_HUNG_INT_RAW	I2S_RX_HUNG_INT_RAW	I2S_TX_REMPTY_INT_RAW	I2S_TX_WFULL_INT_RAW	I2S_RX_REMPTY_INT_RAW	I2S_RX_WFULL_INT_RAW	I2S_TX_PUT_DATA_INT_RAW	I2S_RX_TAKE_DATA_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_RX_TAKE_DATA_INT_RAW The raw interrupt status bit for I2S_RX_TAKE_DATA_INT interrupt. (RO)

I2S_TX_PUT_DATA_INT_RAW The raw interrupt status bit for I2S_TX_PUT_DATA_INT interrupt. (RO)

I2S_RX_WFULL_INT_RAW The raw interrupt status bit for I2S_RX_WFULL_INT interrupt. (RO)

I2S_RX_REMPTY_INT_RAW The raw interrupt status bit for I2S_RX_REMPTY_INT interrupt (RO)

I2S_TX_WFULL_INT_RAW The raw interrupt status bit for I2S_TX_WFULL_INT interrupt. (RO)

I2S_TX_REMPTY_INT_RAW The raw interrupt status bit for I2S_TX_REMPTY_INT interrupt. (RO)

I2S_RX_HUNG_INT_RAW The raw interrupt status bit for I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_HUNG_INT_RAW The raw interrupt status bit for I2S_TX_HUNG_INT interrupt. (RO)

I2S_IN_DONE_INT_RAW The raw interrupt status bit for I2S_IN_DONE_INT interrupt. (RO)

I2S_IN_SUC_EOF_INT_RAW The raw interrupt status bit for I2S_IN_SUC_EOF_INT interrupt. (RO)

I2S_OUT_DONE_INT_RAW The raw interrupt status bit for I2S_OUT_DONE_INT interrupt. (RO)

I2S_OUT_EOF_INT_RAW The raw interrupt status bit for I2S_OUT_EOF_INT interrupt. (RO)

I2S_IN_DSCR_ERR_INT_RAW The raw interrupt status bit for I2S_IN_DSCR_ERR_INT interrupt. (RO)

I2S_OUT_DSCR_ERR_INT_RAW The raw interrupt status bit for I2S_OUT_DSCR_ERR_INT interrupt. (RO)

I2S_IN_DSCR_EMPTY_INT_RAW The raw interrupt status bit for I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

I2S_OUT_TOTAL_EOF_INT_RAW The raw interrupt status bit for I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

I2S_V_SYNC_INT_RAW The raw interrupt status bit for I2S_V_SYNC_INT interrupt. (RO)

Register 26.10: I2S_INT_ST_REG (0x0010)

31–18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2S_V_SYNC_INT_ST	I2S_OUT_TOTAL_EOF_INT_ST	I2S_IN_DSCR_EMPTY_INT_ST	I2S_OUT_DSCR_ERR_INT_ST	I2S_IN_DSCR_ERR_INT_ST	I2S_OUT_EOF_INT_ST	I2S_OUT_DONE_INT_ST	(reserved)	I2S_IN_SUC_EOF_INT_ST	I2S_IN_DONE_INT_ST	I2S_TX_HUNG_INT_ST	I2S_RX_HUNG_INT_ST	I2S_TX_REMPTY_INT_ST	I2S_TX_WFULL_INT_ST	I2S_RX_REMPTY_INT_ST	I2S_RX_WFULL_INT_ST	I2S_TX_PUT_DATA_INT_ST	I2S_RX_TAKE_DATA_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_RX_TAKE_DATA_INT_ST The masked interrupt status bit for I2S_RX_TAKE_DATA_INT interrupt. (RO)

I2S_TX_PUT_DATA_INT_ST The masked interrupt status bit for I2S_TX_PUT_DATA_INT interrupt. (RO)

I2S_RX_WFULL_INT_ST The masked interrupt status bit for I2S_RX_WFULL_INT interrupt (RO)

I2S_RX_REMPTY_INT_ST The masked interrupt status bit for I2S_RX_REMPTY_INT interrupt. (RO)

I2S_TX_WFULL_INT_ST The masked interrupt status bit for I2S_TX_WFULL_INT interrupt. (RO)

I2S_TX_REMPTY_INT_ST The masked interrupt status bit for I2S_TX_REMPTY_INT interrupt. (RO)

I2S_RX_HUNG_INT_ST The masked interrupt status bit for I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_HUNG_INT_ST The masked interrupt status bit for I2S_TX_HUNG_INT interrupt. (RO)

I2S_IN_DONE_INT_ST The masked interrupt status bit for I2S_IN_DONE_INT interrupt. (RO)

I2S_IN_SUC_EOF_INT_ST The masked interrupt status bit for I2S_IN_SUC_EOF_INT interrupt. (RO)

I2S_OUT_DONE_INT_ST The masked interrupt status bit for I2S_OUT_DONE_INT interrupt. (RO)

I2S_OUT_EOF_INT_ST The masked interrupt status bit for I2S_OUT_EOF_INT interrupt. (RO)

I2S_IN_DSCR_ERR_INT_ST The masked interrupt status bit for I2S_IN_DSCR_ERR_INT interrupt. (RO)

I2S_OUT_DSCR_ERR_INT_ST The masked interrupt status bit for I2S_OUT_DSCR_ERR_INT interrupt. (RO)

I2S_IN_DSCR_EMPTY_INT_ST The masked interrupt status bit for I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

I2S_OUT_TOTAL_EOF_INT_ST The masked interrupt status bit for I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

I2S_V_SYNC_INT_ST The masked interrupt status bit for I2S_V_SYNC_INT interrupt. (RO)

Register 26.11: I2S_INT_ENA_REG (0x0014)

Bit(s)	Field	Reset
31:18	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0
17	I2S_V_SYNC_INT_ENA	0
16	I2S_OUT_TOTAL_EOF_INT_ENA	0
15	I2S_IN_DSCR_EMPTY_INT_ENA	0
14	I2S_OUT_DSCR_ERR_INT_ENA	0
13	I2S_IN_DSCR_ERR_INT_ENA	0
12	I2S_OUT_EOF_INT_ENA	0
11	I2S_OUT_DONE_INT_ENA	0
10	(reserved)	0
9	I2S_IN_SUC_EOF_INT_ENA	0
8	I2S_IN_DONE_INT_ENA	0
7	I2S_TX_HUNG_INT_ENA	0
6	I2S_RX_HUNG_INT_ENA	0
5	I2S_TX_REMPTY_INT_ENA	0
4	I2S_TX_WFULL_INT_ENA	0
3	I2S_RX_REMPTY_INT_ENA	0
2	I2S_RX_WFULL_INT_ENA	0
1	I2S_TX_PUT_DATA_INT_ENA	0
0	I2S_RX_TAKE_DATA_INT_ENA	0

I2S_RX_TAKE_DATA_INT_ENA The interrupt enable bit for I2S_RX_TAKE_DATA_INT interrupt. (R/W)

I2S_TX_PUT_DATA_INT_ENA The interrupt enable bit for I2S_TX_PUT_DATA_INT interrupt. (R/W)

I2S_RX_WFULL_INT_ENA The interrupt enable bit for the I2S_RX_WFULL_INT interrupt. (R/W)

I2S_RX_REMPTY_INT_ENA The interrupt enable bit for I2S_RX_REMPTY_INT interrupt. (R/W)

I2S_TX_WFULL_INT_ENA The interrupt enable bit for I2S_TX_WFULL_INT interrupt. (R/W)

I2S_TX_REMPTY_INT_ENA The interrupt enable bit for I2S_TX_REMPTY_INT interrupt. (R/W)

I2S_RX_HUNG_INT_ENA The interrupt enable bit for the I2S_RX_HUNG_INT interrupt. (R/W)

I2S_TX_HUNG_INT_ENA The interrupt enable bit for I2S_TX_HUNG_INT interrupt. (R/W)

I2S_IN_DONE_INT_ENA The interrupt enable bit for I2S_IN_DONE_INT interrupt. (R/W)

I2S_IN_SUC_EOF_INT_ENA The interrupt enable bit for I2S_IN_SUC_EOF_INT interrupt. (R/W)

I2S_OUT_DONE_INT_ENA The interrupt enable bit for I2S_OUT_DONE_INT interrupt. (R/W)

I2S_OUT_EOF_INT_ENA The interrupt enable bit for I2S_OUT_EOF_INT interrupt. (R/W)

I2S_IN_DSCR_ERR_INT_ENA The interrupt enable bit for I2S_IN_DSCR_ERR_INT interrupt. (R/W)

I2S_OUT_DSCR_ERR_INT_ENA The interrupt enable bit for I2S_OUT_DSCR_ERR_INT interrupt. (R/W)

I2S_IN_DSCR_EMPTY_INT_ENA The interrupt enable bit for I2S_IN_DSCR_EMPTY_INT interrupt. (R/W)

I2S_OUT_TOTAL_EOF_INT_ENA The interrupt enable bit for I2S_OUT_TOTAL_EOF_INT interrupt. (R/W)

I2S_V_SYNC_INT_ENA The interrupt enable bit for I2S_V_SYNC_INT interrupt. (R/W)

Register 26.12: I2S_INT_CLR_REG (0x0018)

31..18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2S_V_SYNC_INT_CLR	I2S_OUT_TOTAL_EOF_INT_CLR	I2S_IN_DSCR_EMPTY_INT_CLR	I2S_OUT_DSCR_ERR_INT_CLR	I2S_IN_DSCR_ERR_INT_CLR	I2S_OUT_EOF_INT_CLR	I2S_OUT_DONE_INT_CLR	(reserved)	I2S_IN_SUC_EOF_INT_CLR	I2S_IN_DONE_INT_CLR	I2S_TX_HUNG_INT_CLR	I2S_RX_HUNG_INT_CLR	I2S_TX_REMPTY_INT_CLR	I2S_TX_WFULL_INT_CLR	I2S_RX_REMPTY_INT_CLR	I2S_RX_WFULL_INT_CLR	I2S_PUT_DATA_INT_CLR	I2S_TAKE_DATA_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_TAKE_DATA_INT_CLR Set this bit to clear I2S_RX_TAKE_DATA_INT interrupt. (WO)

I2S_PUT_DATA_INT_CLR Set this bit to clear I2S_TX_PUT_DATA_INT interrupt. (WO)

I2S_RX_WFULL_INT_CLR Set this bit to clear I2S_RX_WFULL_INT interrupt. (WO)

I2S_RX_REMPTY_INT_CLR Set this bit to clear I2S_RX_REMPTY_INT interrupt. (WO)

I2S_TX_WFULL_INT_CLR Set this bit to clear I2S_TX_WFULL_INT interrupt. (WO)

I2S_TX_REMPTY_INT_CLR Set this bit to clear I2S_TX_REMPTY_INT interrupt. (WO)

I2S_RX_HUNG_INT_CLR Set this bit to clear I2S_RX_HUNG_INT interrupt. (WO)

I2S_TX_HUNG_INT_CLR Set this bit to clear I2S_TX_HUNG_INT interrupt. (WO)

I2S_IN_DONE_INT_CLR Set this bit to clear I2S_IN_DONE_INT interrupt. (WO)

I2S_IN_SUC_EOF_INT_CLR Set this bit to clear I2S_IN_SUC_EOF_INT interrupt. (WO)

I2S_OUT_DONE_INT_CLR Set this bit to clear I2S_OUT_DONE_INT interrupt. (WO)

I2S_OUT_EOF_INT_CLR Set this bit to clear I2S_OUT_EOF_INT interrupt. (WO)

I2S_IN_DSCR_ERR_INT_CLR Set this bit to clear I2S_IN_DSCR_ERR_INT interrupt. (WO)

I2S_OUT_DSCR_ERR_INT_CLR Set this bit to clear I2S_OUT_DSCR_ERR_INT interrupt. (WO)

I2S_IN_DSCR_EMPTY_INT_CLR Set this bit to clear I2S_IN_DSCR_EMPTY_INT interrupt. (WO)

I2S_OUT_TOTAL_EOF_INT_CLR Set this bit to clear I2S_OUT_TOTAL_EOF_INT interrupt. (WO)

I2S_V_SYNC_INT_CLR Set this bit to clear I2S_V_SYNC_INT interrupt. (WO)

Register 26.13: I2S_TIMING_REG (0x001C)

Bits	Field	Reset
31:25	(reserved)	0 0 0 0 0 0 0
24	I2S_TX_BCK_IN_INV	0
23:22	I2S_DATA_ENABLE_DELAY	0
21	I2S_RX_DSYNC_SW	0
20	I2S_TX_DSYNC_SW	0
19:18	I2S_RX_BCK_OUT_DELAY	0
17:16	I2S_RX_WS_OUT_DELAY	0
15:14	I2S_TX_SD_OUT_DELAY	0
13:12	I2S_TX_WS_OUT_DELAY	0
11:10	I2S_TX_BCK_OUT_DELAY	0
9:8	I2S_RX_SD_IN_DELAY	0
7:6	I2S_RX_WS_IN_DELAY	0
5:4	I2S_RX_BCK_IN_DELAY	0
3:2	I2S_TX_WS_IN_DELAY	0
1:0	I2S_TX_BCK_IN_DELAY	0

I2S_TX_BCK_IN_DELAY Number of delay cycles for BCK signal into the transmitter based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

I2S_TX_WS_IN_DELAY Number of delay cycles for WS signal into the transmitter based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

I2S_RX_BCK_IN_DELAY Number of delay cycles for BCK signal into the receiver based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

Continued on the next page...

Register 26.13: I2S_TIMING_REG (0x001C)

Continued from the previous page...

I2S_RX_WS_IN_DELAY Number of delay cycles for WS signal into the receiver based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

I2S_RX_SD_IN_DELAY Number of delay cycles for SD signal into the receiver based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

I2S_TX_BCK_OUT_DELAY Number of delay cycles for BCK signal out of the transmitter based on I2S0_CLK. (R/W)

- 0: delayed by 0 cycles
- 1: delayed by 1 cycles
- 2: delayed by 2 cycles
- 3: delayed by 3 cycles

I2S_TX_WS_OUT_DELAY Number of delay cycles for WS signal out of the transmitter based on I2S0_CLK. (R/W)

- 0: delayed by 0 cycles
- 1: delayed by 1 cycles
- 2: delayed by 2 cycles
- 3: delayed by 3 cycles

Continued on the next page...

Register 26.13: I2S_TIMING_REG (0x001C)

Continued from the previous page...

I2S_TX_SD_OUT_DELAY Number of delay cycles for SD signal out of the transmitter based on I2S0_CLK. (R/W)

- 0: delayed by 0 cycles
- 1: delayed by 1 cycles
- 2: delayed by 2 cycles
- 3: delayed by 3 cycles

I2S_RX_WS_OUT_DELAY Number of delay cycles for WS signal out of the receiver based on I2S0_CLK. (R/W)

- 0: delayed by 0 cycles
- 1: delayed by 1 cycles
- 2: delayed by 2 cycles
- 3: delayed by 3 cycles

I2S_RX_BCK_OUT_DELAY Number of delay cycles for BCK signal out of the receiver based on I2S0_CLK. (R/W)

- 0: delayed by 0 cycles
- 1: delayed by 1 cycles
- 2: delayed by 2 cycles
- 3: delayed by 3 cycles

I2S_TX_DSYNC_SW Set this bit to synchronize signals into the transmitter by two flip-flop synchronizer. (R/W)

- 0: the signals will be firstly clocked by rising clock edge , then clocked by falling clock edge.
- 1: the signals will be firstly clocked by falling clock edge, then clocked by rising clock edge.

Continued on the next page...

Register 26.13: I2S_TIMING_REG (0x001C)

Continued from the previous page...

I2S_RX_DSYNC_SW Set this bit to synchronize signals into the receiver by two flip-flop synchronizer. (R/W)

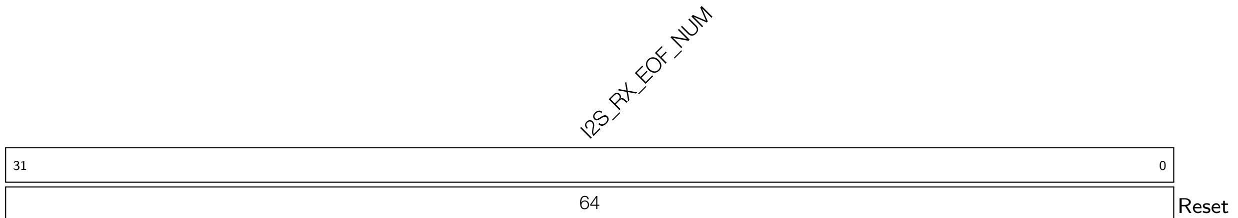
- 0: the signals will be clocked by rising clock edge firstly, then clocked by falling clock edge.
- 1: the signals will be clocked by falling clock edge firstly, then clocked by rising clock edge.

I2S_DATA_ENABLE_DELAY Number of delay cycles for data valid flag based on I2S0_CLK. (R/W)

- 0: delayed by 1.5 cycles
- 1: delayed by 2.5 cycles
- 2: delayed by 3.5 cycles
- 3: delayed by 4.5 cycles

I2S_TX_BCK_IN_INV Set this bit to invert BCK signal input to the slave transmitter. (R/W)

Register 26.14: I2S_RXEOF_NUM_REG (0x0024)



I2S_RX_EOF_NUM The length of data to be received. It will trigger I2S_IN_SUC_EOF_INT. (R/W)

Register 26.15: I2S_OUT_LINK_REG (0x0030)

I2S_OUTLINK_ADDR The address of first outlink descriptor (R/W)

I2S_OUTLINK_STOP Set this bit to stop outlink descriptor. (R/W)

I2S_OUTLINK_START Set this bit to start outlink descriptor. (R/W)

I2S_OUTLINK_RESTART Set this bit to restart outlink descriptor. (R/W)

Register 26.16: I2S_IN_LINK_REG (0x0034)

31	30	29	28	27 … 20	19 … 0	
(reserved)	I2S_INLINK_RESTART	I2S_INLINK_START	I2S_INLINK_STOP	(reserved)	I2S_INLINK_ADDR	
0	0	0	0	0 0 0 0 0 0 0 0	0x000	Reset

I2S_INLINK_ADDR The address of first inlink descriptor (R/W)

I2S_INLINK_STOP Set this bit to stop inlink descriptor. (R/W)

I2S_INLINK_START Set this bit to start inlink descriptor. (R/W)

I2S_INLINK_RESTART Set this bit to restart inlink descriptor. (R/W)

Register 26.17: I2S_OUT_EOF_DES_ADDR_REG (0x0038)

31 … 0	
I2S_OUT_EOF_DES_ADDR	
0x000000	Reset

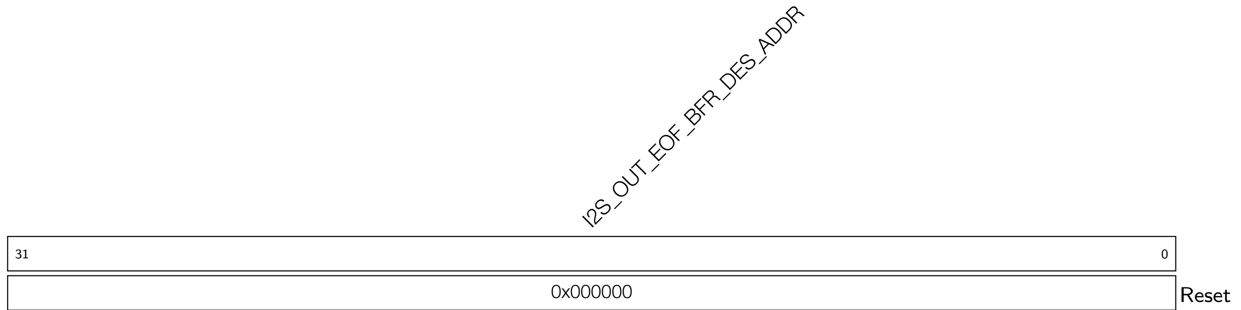
I2S_OUT_EOF_DES_ADDR The address of outlink descriptor that produces EOF (RO)

Register 26.18: I2S_IN_EOF_DES_ADDR_REG (0x003C)

31 … 0	
I2S_IN_SUC_EOF_DES_ADDR	
0x000000	Reset

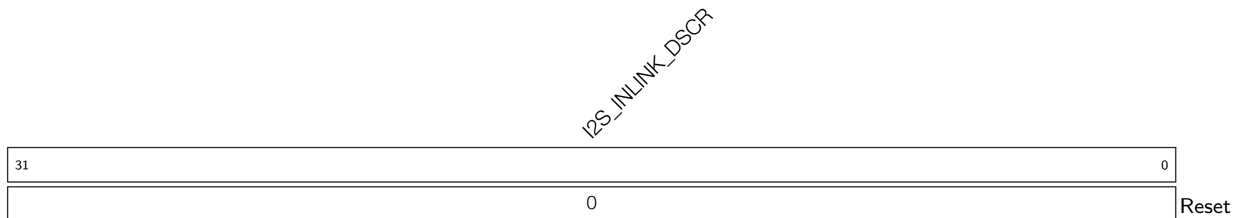
I2S_IN_SUC_EOF_DES_ADDR The address of inlink descriptor that produces EOF (RO)

Register 26.19: I2S_OUT_EOF_BFR_DES_ADDR_REG (0x0040)



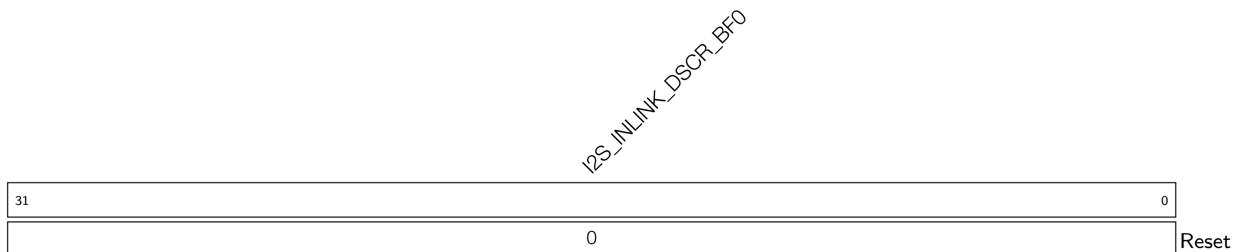
I2S_OUT_EOF_BFR_DES_ADDR The address of buffer relative to the outlink descriptor that produces EOF (RO)

Register 26.20: I2S_INLINK_DSCR_REG (0x0048)



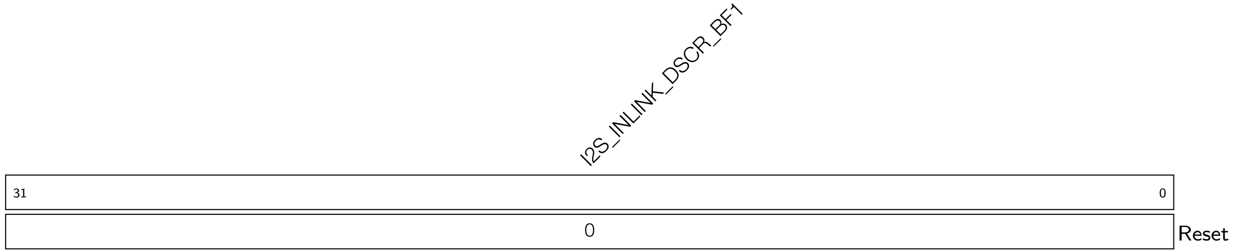
I2S_INLINK_DSCR The address of current inlink descriptor (RO)

Register 26.21: I2S_INLINK_DSCR_BF0_REG (0x004C)



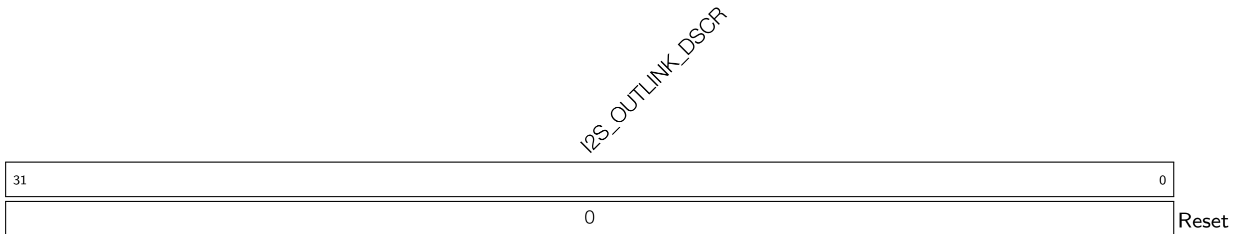
I2S_INLINK_DSCR_BF0 The address of next inlink descriptor (RO)

Register 26.22: I2S_INLINK_DSCR_BF1_REG (0x0050)



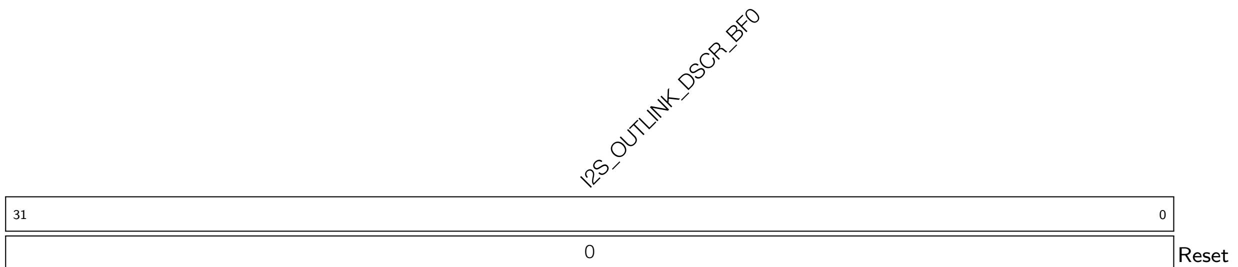
I2S_INLINK_DSCR_BF1 The address of next inlink data buffer (RO)

Register 26.23: I2S_OUTLINK_DSCR_REG (0x0054)



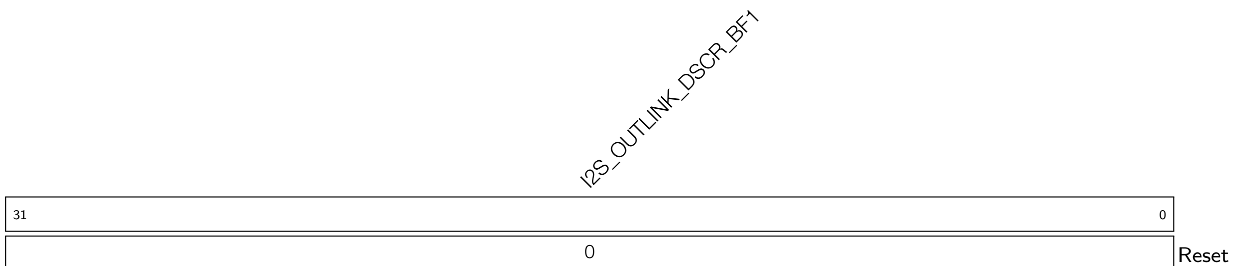
I2S_OUTLINK_DSCR The address of current outlink descriptor (RO)

Register 26.24: I2S_OUTLINK_DSCR_BF0_REG (0x0058)



I2S_OUTLINK_DSCR_BF0 The address of next outlink descriptor (RO)

Register 26.25: I2S_OUTLINK_DSCR_BF1_REG (0x005C)



I2S_OUTLINK_DSCR_BF1 The address of next outlink data buffer (RO)

Register 26.26: I2S_LC_CONF_REG (0x0060)

31–16	15–14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	I2S_EXT_MEM_BK_SIZE	I2S_MEM_TRANS_EN	I2S_CHECK_OWNER	I2S_OUT_DATA_BURST_EN	I2S_INDSCR_BURST_EN	I2S_OUTDSCR_BURST_EN	I2S_OUT_EOF_MODE	(reserved)	I2S_OUT_AUTO_WRBACK	I2S_IN_LOOP_TEST	I2S_OUT_LOOP_TEST	I2S_AHBM_RST	I2S_AHBM_FIFO_RST	I2S_OUT_RST	I2S_IN_RST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Reset

I2S_IN_RST Set this bit to reset in-DMA FSM. Set this bit before the DMA configuration. (R/W)

I2S_OUT_RST Set this bit to reset out-DMA FSM. Set this bit before the DMA configuration. (R/W)

I2S_AHBM_FIFO_RST Set this bit to reset AHB interface cmdFIFO of DMA. Set this bit before the DMA configuration. (R/W)

I2S_AHBM_RST Set this bit to reset AHB interface of DMA. Set this bit before the DMA configuration. (R/W)

I2S_OUT_LOOP_TEST Set this bit to loop test inlink. (R/W)

I2S_IN_LOOP_TEST Set this bit to loop test outlink. (R/W)

I2S_OUT_AUTO_WRBACK Set this bit to enable outlink-written-back automatically when out buffer is transmitted done. (R/W)

I2S_OUT_EOF_MODE DMA out EOF flag generation mode. 1: When DMA has popped all data from the FIFO. 0: When AHB has pushed all data to the FIFO. (R/W)

I2S_OUTDSCR_BURST_EN DMA outlink descriptor transfer mode configuration bit. 1: Prepare outlink descriptor with burst mode. 0: Prepare outlink descriptor with byte mode. (R/W)

I2S_INDSCR_BURST_EN DMA inlink descriptor transfer mode configuration bit. 1: Prepare inlink descriptor with burst mode. 0: Prepare inlink descriptor with byte mode. (R/W)

I2S_OUT_DATA_BURST_EN Transmitter data transfer mode configuration bit. 1: Prepare out data with burst mode. 0: Prepare out data with byte mode. (R/W)

I2S_CHECK_OWNER Set this bit to enable check owner bit by hardware. (R/W)

I2S_MEM_TRANS_EN Reserved. (R/W)

I2S_EXT_MEM_BK_SIZE DMA access external memory block size. 0: 16 bytes. 1: 32 bytes. 2: 64 bytes. 3: reserved. (R/W)

Register 26.27: I2S_LC_STATE0_REG (0x006C)

I2S_OUT_EMPTY	I2S_OUT_FULL	I2S_OUTFIFO_CNT	I2S_OUT_STATE	I2S_OUT_DSCR_STATE	I2S_OUTLINK_DSCR_ADDR	
31	30	29 23	22 20	19 18	17 0	
0	0	0	0	0	0x000	Reset

I2S_OUTLINK_DSCR_ADDR I2S DMA out descriptor address. (RO)

I2S_OUT_DSCR_STATE I2S DMA out descriptor state. (RO)

I2S_OUT_STATE I2S DMA out data state. (RO)

I2S_OUTFIFO_CNT The remains of I2S DMA outfifo data. (RO)

I2S_OUT_FULL I2S DMA outfifo is full. (RO)

I2S_OUT_EMPTY I2S DMA outfifo is empty. (RO)

Register 26.28: I2S_LC_STATE1_REG (0x0070)

I2S_IN_EMPTY	I2S_IN_FULL	I2S_INFIFO_CNT_DEBUG	I2S_IN_STATE	I2S_IN_DSCR_STATE	I2S_INLINK_DSCR_ADDR	
31	30	29 23	22 20	19 18	17 0	
0	0	0	0	0	0x000	Reset

I2S_INLINK_DSCR_ADDR I2S DMA in descriptor address. (RO)

I2S_IN_DSCR_STATE I2S DMA in descriptor state. (RO)

I2S_IN_STATE I2S DMA in data state. (RO)

I2S_INFIFO_CNT_DEBUG The remains of I2S DMA infifo data. (RO)

I2S_IN_FULL I2S DMA infifo is full. (RO)

I2S_IN_EMPTY I2S DMA infifo is empty. (RO)

Register 26.29: I2S_STATE_REG (0x00BC)

(reserved)	I2S_TX_IDLE	
31 … 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

I2S_TX_IDLE 1: I2S TX is in idle state. 0: I2S TX is at work. (RO)

Register 26.30: I2S_CLKM_CONF_REG (0x00AC)

(reserved)	I2S_CLK_SEL	I2S_CLK_EN	I2S_CLKM_DIV_A	I2S_CLKM_DIV_B	I2S_CLKM_DIV_NUM	
31 … 23	22 … 21	20	19 … 14	13 … 8	7 … 0	
0 0 0 0 0 0 0 0 0	0	0	0x0	0x0	4	Reset

I2S_CLKM_DIV_NUM Integral I2S clock divider value (R/W)

I2S_CLKM_DIV_B Fractional clock divider numerator value (R/W)

I2S_CLKM_DIV_A Fractional clock divider denominator value (R/W)

I2S_CLK_EN Set this bit to enable clock gate (R/W)

I2S_CLK_SEL Set this bit to select I2S module clock source. 0: No clock. 1: APLL_CLK. 2: PLL_160M_CLK. 3: No clock. (R/W)

Register 26.31: I2S_SAMPLE_RATE_CONF_REG (0x00B0)

(reserved)	I2S_RX_BITS_MOD	I2S_TX_BITS_MOD	I2S_RX_BCK_DIV_NUM	I2S_TX_BCK_DIV_NUM	
31 … 24	23 … 18	17 … 12	11 … 6	5 … 0	
0 0 0 0 0 0 0 0	16	16	6	6	Reset

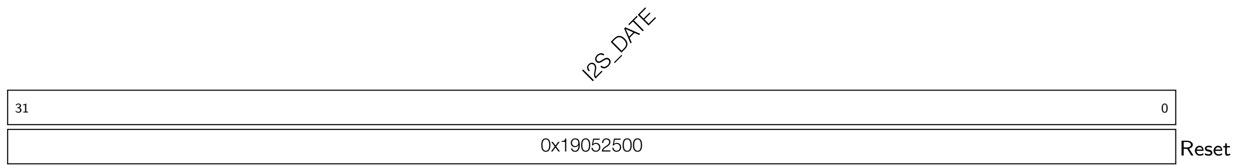
I2S_TX_BCK_DIV_NUM Bit clock configuration bits in transmitter mode. (R/W)

I2S_RX_BCK_DIV_NUM Bit clock configuration bits in receiver mode. (R/W)

I2S_TX_BITS_MOD Set the bits to configure bit length of I2S transmitter channel, the value of which can only be 8, 16, 24 and 32. (R/W)

I2S_RX_BITS_MOD Set the bits to configure bit length of I2S receiver channel, the value of which can only be 8, 16, 24 and 32. (R/W)

Register 26.32: I2S_DATE_REG (0x00FC)



I2S_DATE Version control register (R/W)

27. Pulse Count Controller (PCNT)

The pulse count controller (PCNT) is designed to count input pulses and generate interrupts. It can increment or decrement a pulse counter value by keeping track of rising (positive) or falling (negative) edges of the input pulse signal. The PCNT has four independent pulse counters, called units which have their groups of registers. In this chapter, *n* denotes the number of a unit from 0 ~ 3.

Each unit includes two channels (ch0 and ch1) which can independently increment or decrement its pulse counter value. The remainder of the chapter will mostly focus on channel 0 (ch0) as the functionality of the two channels is identical.

As shown in Figure 27-1, each channel has two input signals:

1. One control signal (e.g. ctrl_ch0_u*n*, the control signal for ch0 of unit n)
2. One input pulse signal (e.g. sig_ch0_u*n*, the input pulse signal for ch0 of unit n)

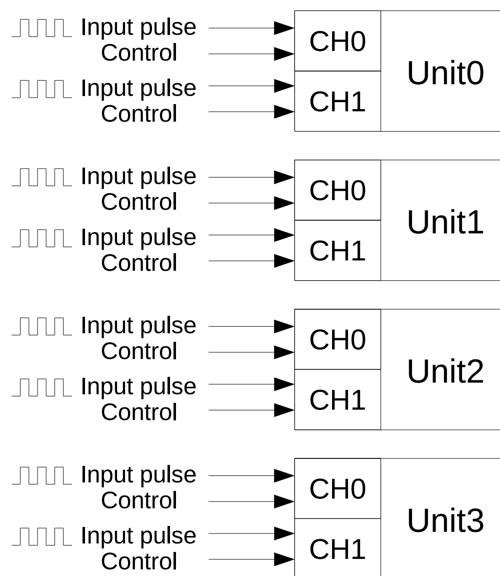


Figure 27-1. PCNT Block Diagram

27.1 Features

A PCNT has the following features:

- Four independent pulse counters (units)
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g. sig_ch0_u*n*) with their corresponding control signals (e.g. ctrl_ch0_u*n*)
- Independent filtering of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit
- Each channel has the following parameters:
    1. Selection between counting on positive or negative edges of the input pulse signal

2. Configuration to Increment, Decrement, or Disable counter mode for control signal's high and low states

27.2 Functional Description

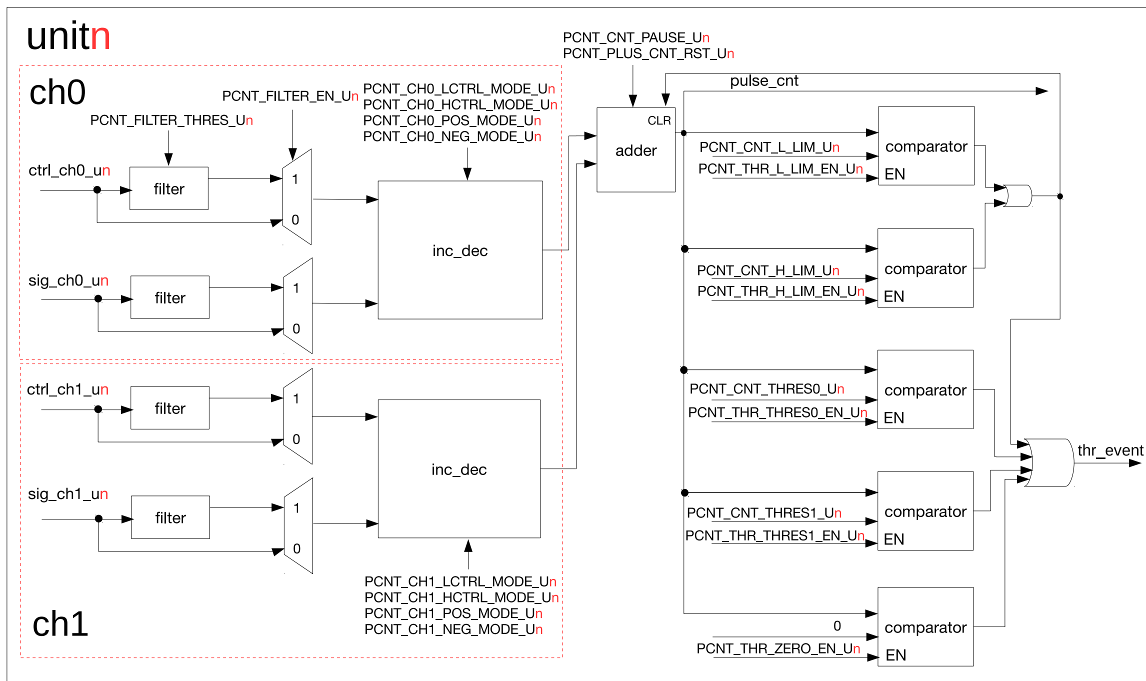


Figure 27-2. PCNT Unit Architecture

Figure 27-2 shows PCNT's architecture. As stated above, ctrl_ch0_u*n* is the control signal for ch0 of unit n. Its high and low states can be assigned different counter modes and used for pulse counting of the channel's input pulse signal sig_ch0_u*n* on negative or positive edges. The available counter modes are as follows:

- Increment mode: When a channel detects an active edge of sig_ch0_u*n* (the one configured for counting), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if PCNT_CNT_PAUSE_U*n* is set before pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.
- Decrement mode: When a channel detects an active edge of sig_ch0_u*n* (the one configured for counting), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if PCNT_CNT_PAUSE_U*n* is set before pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.
- Disable mode: Counting is disabled, and the counter value pulse_cnt freezes.

Table 157 to Table 160 provide information on how to configure the counter mode for channel 0.

Each unit has one filter for all its control and input pulse signals. A filter can be enabled with the bit PCNT_FILTER_EN_U*n*. The filter monitors the signals and ignores all the noise, i.e. the glitches with pulse widths shorter than
PCNT_FILTER_THRES_U*n* APB clock cycles in length.

As previously mentioned, each unit has two channels which process different input pulse signals and increase or decrease values via their respective inc_dec modules, then the two channels send these values to the adder

Table 157: Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_POS_MODE_Un	PCNT_CH0_LCTRL_MODE_Un	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 158: Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_POS_MODE_Un	PCNT_CH0_HCTRL_MODE_Un	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 159: Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_NEG_MODE_Un	PCNT_CH0_LCTRL_MODE_Un	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 160: Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_NEG_MODE_Un	PCNT_CH0_HCTRL_MODE_Un	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

module that is 16-bit wide with a sign bit. This adder can be suspended by setting PCNT_CNT_PAUSE_Un, and cleared by setting PCNT_PULSE_CNT_RST_Un.

The PCNT has five watchpoints that share one interrupt. The interrupt can be enabled or disabled by interrupt enable signals of each individual watchpoint.

- Maximum count value: When pulse_cnt reaches PCNT_CNT_H_LIM_Un, an interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_Un is high.
- Minimum count value: When pulse_cnt reaches PCNT_CNT_L_LIM_Un, an interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_Un is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_THRES0_Un or PCNT_CNT_THRES1_Un, an interrupt is triggered and either PCNT_CNT_THR_THRES0_LAT_Un or PCNT_CNT_THR_THRES1_LAT_Un is high respectively.
- Zero: When pulse_cnt is 0, an interrupt is triggered and PCNT_CNT_THR_ZERO_LAT_Un is valid.

27.3 Applications

In each unit, channel 0 and channel 1 can be configured to work independently or together. The three subsections below provide details of channel 0 incrementing independently, channel 0 decrementing independently, and channel 0 and channel 1 incrementing together. For other working modes not elaborated in this section (e.g. channel 1 incrementing/decremeting independently, or one channel incrementing while the other decrementing), reference can be made to these three subsections.

27.3.1 Channel 0 Incrementing Independently

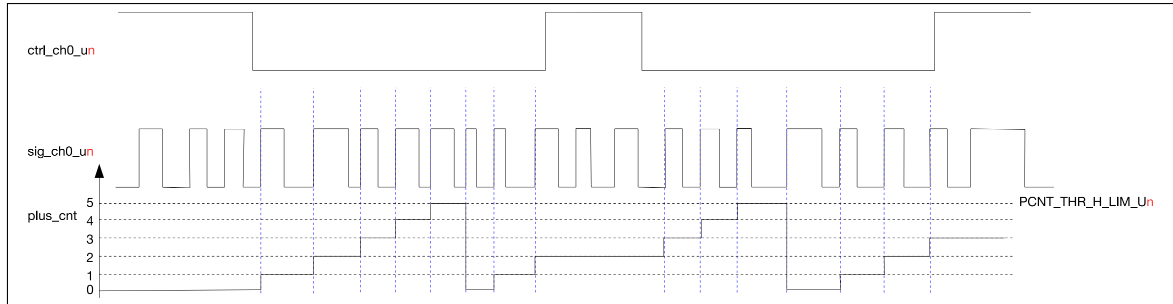


Figure 27-3. Channel 0 Up Counting Diagram

Figure 27-3 illustrates how channel 0 is configured to increment independently on the positive edge of sig_ch0_un while channel 1 is disabled (see subsection 27.2 for how to disable channel 1). The configuration of channel 0 is shown below.

- PCNT_CH0_LCTRL_MODE_Un=0: When ctrl_ch0_un is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
- PCNT_CH0_HCTRL_MODE_Un=2: When ctrl_ch0_un is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
- PCNT_CH0_POS_MODE_Un=1: The counter increments on the positive edge of sig_ch0_un.
- PCNT_CH0_NEG_MODE_Un=0: The counter idles on the negative edge of sig_ch0_un.
- PCNT_CNT_H_LIM_Un=5: When pulse_cnt counts up to PCNT_CNT_H_LIM_Un, it is cleared.

27.3.2 Channel 0 Decrementing Independently

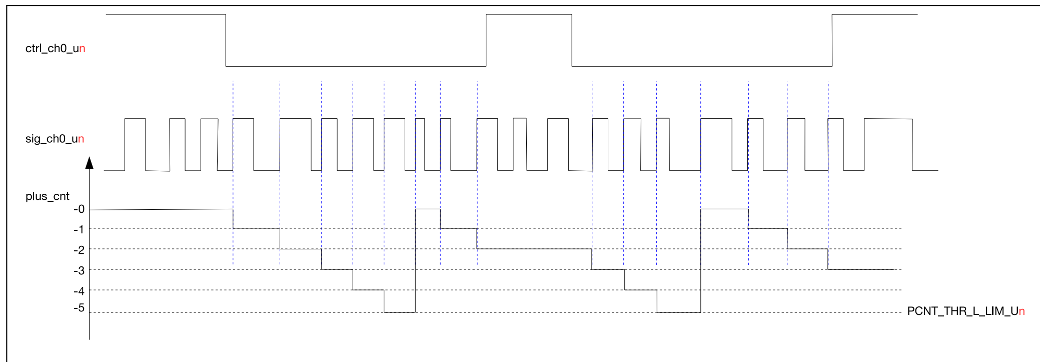


Figure 27-4. Channel 0 Down Counting Diagram

Figure 27-4 illustrates how channel 0 is configured to decrement independently on the positive edge of sig_ch0_un while channel 1 is disabled. The configuration of channel 0 in this case differs from that in Figure 27-3 in the following aspects:

- PCNT_CH0_POS_MODE_Un=2: the counter decrements on the positive edge of sig_ch0_un.
- PCNT_CNT_L_LIM_Un=-5: when pulse_cnt counts down to PCNT_CNT_L_LIM_Un, it is cleared.

27.3.3 Channel 0 and Channel 1 Incrementing Together

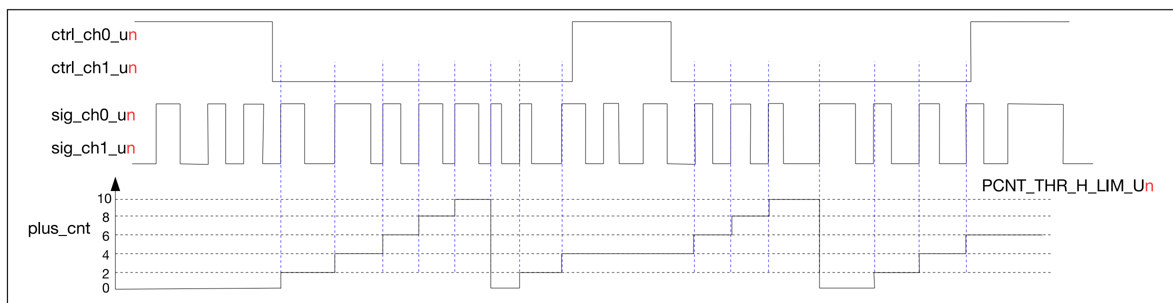


Figure 27-5. Two Channels Up Counting Diagram

Figure 27-5 illustrates how channel 0 and channel 1 are configured to increment on the positive edge of sig_ch0_un and sig_ch1_un respectively at the same time. It can be seen in Figure 27-5 that control signal ctrl_ch0_un and ctrl_ch1_un have the same waveform, so as input pulse signal sig_ch0_un and sig_ch1_un. The configuration procedure is shown below.

- For channel 0:
  - PCNT_CH0_LCTRL_MODE_Un=0: When ctrl_ch0_un is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH0_HCTRL_MODE_Un=2: When ctrl_ch0_un is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH0_POS_MODE_Un=1: The counter increments on the positive edge of sig_ch0_un.
  - PCNT_CH0_NEG_MODE_Un=0: The counter idles on the negative edge of sig_ch0_un.
- For channel 1:

  - PCNT_CH1_LCTRL_MODE_Un=0: When ctrl_ch1_un is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH1_HCTRL_MODE_Un=2: When ctrl_ch1_un is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH1_POS_MODE_Un=1: The counter increments on the positive edge of sig_ch1_un.
  - PCNT_CH1_NEG_MODE_Un=0: The counter idles on the negative edge of sig_ch1_un.
- PCNT_CNT_H_LIM_Un=10: When pulse_cnt counts up to PCNT_CNT_H_LIM_Un, it is cleared.

27.4 Base Address

Users can access the PCNT registers with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 161: PCNT Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F417000
PeriBUS2	0x60017000

27.5 Register Summary

The addresses in the following table are relative to the PCNT base addresses provided in Section 27.4.

Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT UNIT0 status register	0x0050	RO
PCNT_U1_STATUS_REG	PNCT UNIT1 status register	0x0054	RO

Name	Description	Address	Access
PCNT_U2_STATUS_REG	PNCT UNIT2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT UNIT3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

27.6 Registers

Register 27.1: PCNT_Un_CONF0_REG (n: 0-3) (0x0000+0xC*n)

PCNT_CH1_LCTRL_MODE_U0	PCNT_CH1_HCTRL_MODE_U0	PCNT_CH1_POS_MODE_U0	PCNT_CH1_NEG_MODE_U0	PCNT_CH0_LCTRL_MODE_U0	PCNT_CH0_HCTRL_MODE_U0	PCNT_CH0_POS_MODE_U0	PCNT_CH0_NEG_MODE_U0	PCNT_THR_THRES1_EN_U0	PCNT_THR_THRES0_EN_U0	PCNT_THR_L_LIM_EN_U0	PCNT_THR_H_LIM_EN_U0	PCNT_THR_ZERO_EN_U0	PCNT_FILTER_EN_U0	PCNT_FILTER_THRES_U0	
31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15	14	13	12	11	10	9 0	
0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0	0	1	1	1	1	0x10	Reset

PCNT_FILTER_THRES_Un This sets the maximum threshold, in APB_CLK cycles, for the filter.

Any pulses with width less than this will be ignored when the filter is enabled. (R/W)

PCNT_FILTER_EN_Un This is the enable bit for unit n's input filter. (R/W)

PCNT_THR_ZERO_EN_Un This is the enable bit for unit n's zero comparator. (R/W)

PCNT_THR_H_LIM_EN_Un This is the enable bit for unit n's thr_h_lim comparator. (R/W)

PCNT_THR_L_LIM_EN_Un This is the enable bit for unit n's thr_l_lim comparator. (R/W)

PCNT_THR_THRES0_EN_Un This is the enable bit for unit n's thres0 comparator. (R/W)

PCNT_THR_THRES1_EN_Un This is the enable bit for unit n's thres1 comparator. (R/W)

PCNT_CH0_NEG_MODE_Un This register sets the behavior when the signal input of channel 0 detects a negative edge.

1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter (R/W)

PCNT_CH0_POS_MODE_Un This register sets the behavior when the signal input of channel 0 detects a positive edge.

1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter (R/W)

PCNT_CH0_HCTRL_MODE_Un This register configures how the CHn_POS_MODE/CHn_NEG_MODE settings will be modified when the control signal is high.

0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

PCNT_CH0_LCTRL_MODE_Un This register configures how the CHn_POS_MODE/CHn_NEG_MODE settings will be modified when the control signal is low.

0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

Continued on the next page...

Register 27.1: PCNT_Un_CONF0_REG (n: 0-3) (0x0000+0xC*n)

Continued from the previous page...

PCNT_CH1_NEG_MODE_Un This register sets the behavior when the signal input of channel 1 detects a negative edge.

1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter (R/W)

PCNT_CH1_POS_MODE_Un This register sets the behavior when the signal input of channel 1 detects a positive edge.

1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter (R/W)

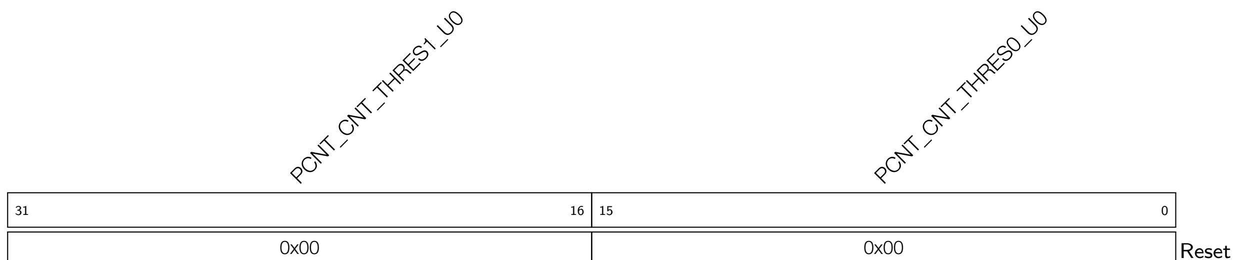
PCNT_CH1_HCTRL_MODE_Un This register configures how the CHn_POS_MODE/CHn_NEG_MODE settings will be modified when the control signal is high.

0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

PCNT_CH1_LCTRL_MODE_Un This register configures how the CHn_POS_MODE/CHn_NEG_MODE settings will be modified when the control signal is low.

0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

Register 27.2: PCNT_Un_CONF1_REG (n: 0-3) (0x0004+0xC*n)



PCNT_CNT_THRES0_Un This register is used to configure the thres0 value for unit n. (R/W)

PCNT_CNT_THRES1_Un This register is used to configure the thres1 value for unit n. (R/W)

Register 27.3: PCNT_Un_CONF2_REG (n: 0-3) (0x0008+0xC*n)

PCNT_CNT_L_LIM_U0 (31..16)	PCNT_CNT_H_LIM_U0 (15..0)	
0x00	0x00	Reset

PCNT_CNT_H_LIM_Un This register is used to configure the thr_h_lim value for unit n. (R/W)

PCNT_CNT_L_LIM_Un This register is used to configure the thr_l_lim value for unit n. (R/W)

Register 27.4: PCNT_CTRL_REG (0x0060)

(reserved) (31..17)	PCNT_CLK_EN (16)	(reserved) (15..8)	PCNT_CNT_PAUSE_U3 (7)	PCNT_PULSE_CNT_RST_U3 (6)	PCNT_CNT_PAUSE_U2 (5)	PCNT_PULSE_CNT_RST_U2 (4)	PCNT_CNT_PAUSE_U1 (3)	PCNT_PULSE_CNT_RST_U1 (2)	PCNT_CNT_PAUSE_U0 (1)	PCNT_PULSE_CNT_RST_U0 (0)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0	0	1	0	1	0	1	0	1	Reset

PCNT_PULSE_CNT_RST_Un Set this bit to clear unit n's counter. (R/W)

PCNT_CNT_PAUSE_Un Set this bit to freeze unit n's counter. (R/W)

PCNT_CLK_EN The registers clock gate enable signal of PCNT module. 1: the registers can be read and written by application. 0: the registers can not be read or written by application (R/W)

Register 27.5: PCNT_Un_CNT_REG (n: 0-3) (0x0030+0x4*n)

(reserved) (31..16)	PCNT_PULSE_CNT_U0 (15..0)	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

PCNT_PULSE_CNT_Un This register stores the current pulse count value for unit n. (RO)

Register 27.6: PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4*n)

31 … 7	6	5	4	3	2	1 … 0	
(reserved)	PCNT_CNT_THR_ZERO_LAT_U0	PCNT_CNT_THR_H_LIM_LAT_U0	PCNT_CNT_THR_L_LIM_LAT_U0	PCNT_CNT_THR_THRES0_LAT_U0	PCNT_CNT_THR_THRES1_LAT_U0	PCNT_CNT_THR_ZERO_MODE_U0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0x0	Reset

PCNT_CNT_THR_ZERO_MODE_U*n* The pulse counter status of PCNT_U*n* corresponding to 0. 0: pulse counter decreases from positive to 0. 1: pulse counter increases from negative to 0. 2: pulse counter is negative. 3: pulse counter is positive. (RO)

PCNT_CNT_THR_THRES1_LAT_U*n* The latched value of thres1 event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thres1 and thres1 event is valid. 0: others. (RO)

PCNT_CNT_THR_THRES0_LAT_U*n* The latched value of thres0 event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thres0 and thres0 event is valid. 0: others. (RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* The latched value of low limit event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thr_l_lim and low limit event is valid. 0: others. (RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* The latched value of high limit event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thr_h_lim and high limit event is valid. 0: others. (RO)

PCNT_CNT_THR_ZERO_LAT_U*n* The latched value of zero threshold event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to 0 and zero threshold event is valid. 0: others. (RO)

Register 27.7: PCNT_INT_RAW_REG (0x0040)

(reserved)	PCNT_CNT_THR_EVENT_U3_INT_RAW	PCNT_CNT_THR_EVENT_U2_INT_RAW	PCNT_CNT_THR_EVENT_U1_INT_RAW	PCNT_CNT_THR_EVENT_U0_INT_RAW	
31 ... 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 27.8: PCNT_INT_ST_REG (0x0044)

(reserved)	PCNT_CNT_THR_EVENT_U3_INT_ST	PCNT_CNT_THR_EVENT_U2_INT_ST	PCNT_CNT_THR_EVENT_U1_INT_ST	PCNT_CNT_THR_EVENT_U0_INT_ST	
31 ... 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

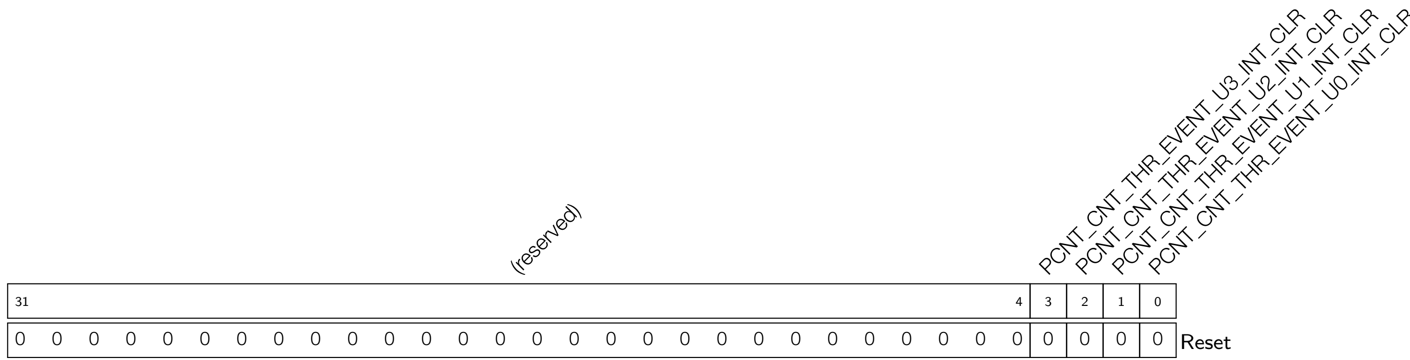
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 27.9: PCNT_INT_ENA_REG (0x0048)

(reserved)	PCNT_CNT_THR_EVENT_U3_INT_ENA	PCNT_CNT_THR_EVENT_U2_INT_ENA	PCNT_CNT_THR_EVENT_U1_INT_ENA	PCNT_CNT_THR_EVENT_U0_INT_ENA	
31 ... 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

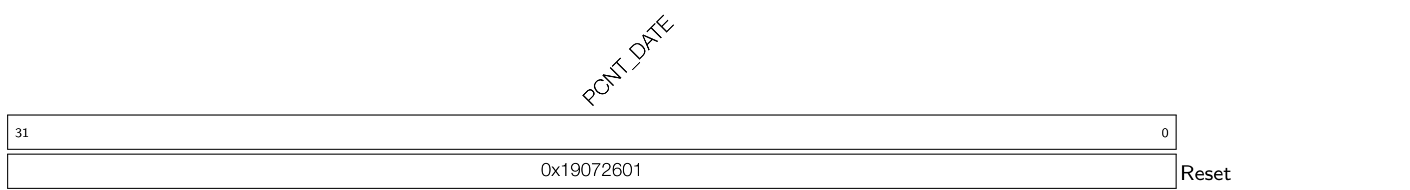
PCNT_CNT_THR_EVENT_U*n*_INT_ENA The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 27.10: PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Set this bit to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 27.11: PCNT_DATE_REG (0x00FC)



PCNT_DATE This is the PCNT version control register. (R/W)

28. USB On-The-Go (USB)

28.1 Overview

The ESP32-S2 features a USB On-The-Go peripheral (henceforth referred to as OTG_FS) along with an integrated transceiver. The OTG_FS can operate as either a USB Host or Device and supports 12 Mbit/s full-speed (FS) and 1.5 Mbit/s low-speed (LS) data rates of the USB1.1 specification. The Host Negotiation Protocol (HNP) and the Session Request Protocol (SRP) are also supported.

28.2 Features

28.2.1 General Features

- FS and LS data rates
- HNP and SRP as A-device or B-device
- Dynamic FIFO (DFIFO) sizing
- Multiple modes of memory access
  - Scatter/Gather DMA mode
  - Buffer DMA mode
  - Slave mode
- Can choose integrated transceiver or external transceiver
- Can be used as a light sleep wake-up source

28.2.2 Device Mode Features

- Endpoint number 0 always present (bi-directional, consisting of EP0 IN and EP0 OUT)
- Six additional endpoints (endpoint numbers 1 to 6), configurable as IN or OUT
- Maximum of five IN endpoints concurrently active at any time (including EP0 IN)
- All OUT endpoints share a single RX FIFO
- Each IN endpoint has a dedicated TX FIFO

28.2.3 Host Mode Features

- Eight channels (pipes)
  - A control pipe consists of two channels (IN and OUT), as IN and OUT transactions must be handled separately. Only Control transfer type is supported.
  - Each of the other seven channels are dynamically configurable to be IN or OUT, and supports Bulk, Isochronous, and Interrupt transfer types.
- All channels share an RX FIFO, non-periodic TX FIFO, and periodic TX FIFO. The size of each FIFO is configurable.

28.3 Functional Description

28.3.1 Controller Core and Interfaces

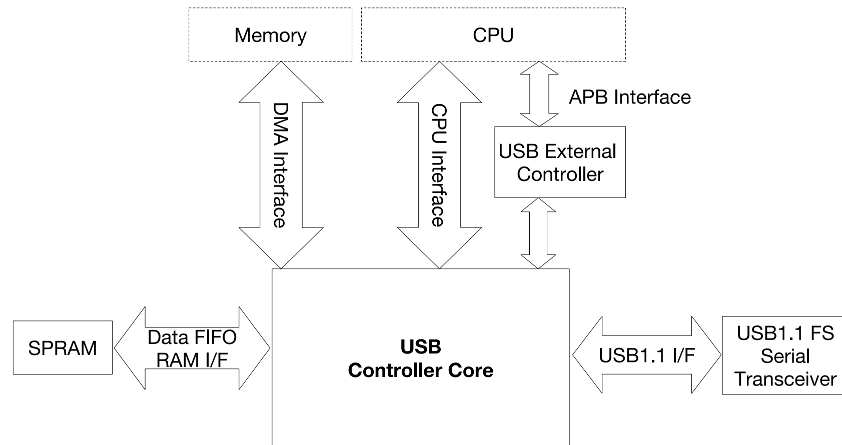


Figure 28-1. OTG_FS System Architecture

The core part of the OTG_FS peripheral is the USB Controller Core. The controller core has the following interfaces (see Figure 28-1):

- **CPU Interface**
  Provides the CPU with read/write access to the controller core's various registers and FIFOs. This interface is internally implemented as an AHB Slave Interface. The way to access the FIFOs through the CPU interface is called Slave mode.
- **APB Interface**
  Allows the CPU to control the USB controller core via the USB external controller.
- **DMA Interface**
  Provides the controller core's internal DMA with read/write access to system memory (e.g., fetching and writing data payloads when operating in DMA mode). This interface is internally implemented as an AHB Master interface.
- **USB1.1 Interface**
  This interface is used to connect the controller core to a USB1.1 FS serial transceiver. The ESP32-S2 contains an internal transceiver, and the USB_PHY_SEL field in the USB external controller register USB_WRAP_OTG_CONF can be configured by software to select connecting the controller core to the ESP32-S2's internal transceiver, or routing through the GPIO matrix to connect to an external transceiver.
- **USB External Controller**
  The USB External Controller is primarily used to control the routing of the USB1.1 FS serial interface to either the internal or external transceiver. The External Controller can also enable a power saving mode by gating the controller core's clock (AHB clock) or putting the connected SPRAM's into a power down. Note that this power saving mode is different for the power savings via SRP.
- **Data FIFO RAM Interface**
  The multiple FIFOs used by the controller core are not actually located within the controller core itself, but on the SPRAM (Single-Port RAM). FIFOs are dynamically sized, thus are allocated at run-time in the SPRAM. When the CPU, DMA, or the controller core attempts to read/write to FIFOs, those accesses are routed through the data FIFO RAM interface.

28.3.2 Memory Layout

The following diagram illustrates the memory layout of the OTG_FS registers which are used to configure and control the USB Controller Core. Note that USB External Controller uses a separate set of registers (called wrap registers).

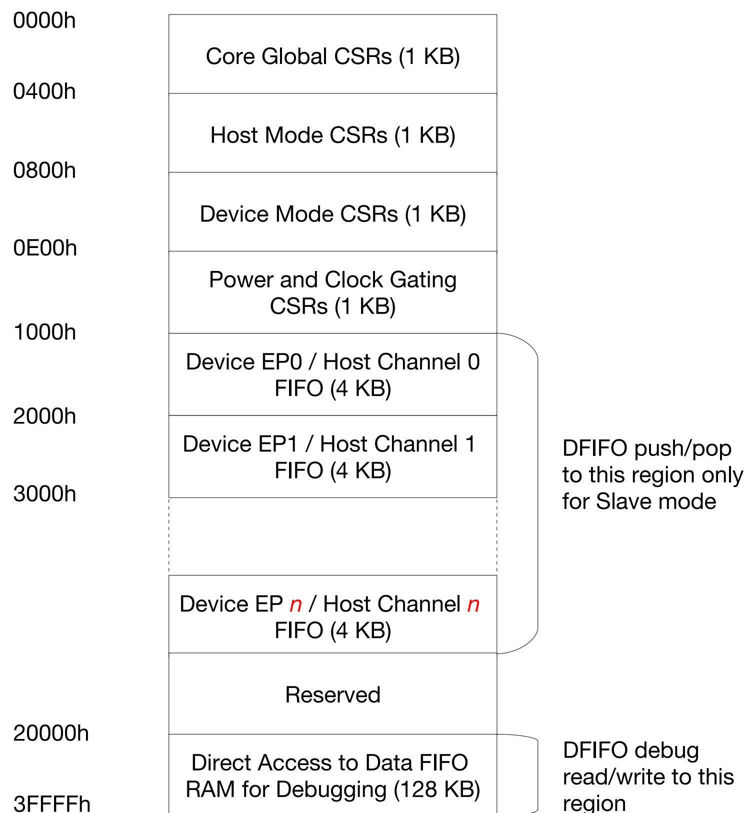


Figure 28-2. OTG_FS Register Layout

28.3.2.1 Control & Status Registers

- **Global CSRs**
  These registers are responsible for the configuration/control/status of the global features of OTG_FS (i.e., features which are common to both Host and Device modes). These features include OTG control (HNP, SRP, and A/B-device detection), USB configuration (selecting Host or Device mode and PHY selection), and system-level interrupts. Software can access these registers whilst in Host or Device modes.
- **Host Mode CSRs**
  These registers are responsible for the configuration/control/status when operating in Host mode, thus should only be accessed when operating in Host mode. Each channel will have its own set of registers within the Host mode CSRs.
- **Device Mode CSRs**
  These registers are responsible for the configuration/control/status when operating in Device mode, thus should only be accessed when operating in Device mode. Each Endpoint will have its own set of registers within the Device mode CSRs.
- **Power and Clock Gating**
  A single register used to control power-down and gate various clocks.

28.3.2.2 FIFO Access

The OTG_FS makes use of multiple FIFOs to buffer transmitted or received data payloads. The number and type of FIFOs are dependent on Host or Device mode, and the number of channels or endpoints used (see Section 28.3.3). There are two ways to access the FIFOs: DMA mode and Slave mode. When using Slave mode, the CPU will need to access to these FIFOs by reading and writing to either the DFIFO push/pop regions or the DFIFO read/write debug region. FIFO access is governed by the following rules:

- Read access to any address in any one of the 4 KB push/pop regions will result in a pop from the shared RX FIFO.
- Write access to a particular 4 KB push/pop region will result in a push to the corresponding endpoint or channel's TX FIFO given that the endpoint is an IN endpoint, or the channel is an OUT channel.
  - In Device mode, data is pushed to the corresponding IN endpoint's dedicated TX FIFO.
  - In Host mode, data is pushed to the non-periodic TX FIFO or the periodic TX FIFO depending on whether the channel is a non-periodic channel, or a periodic channel.
- Access to the 128 KB read/write region will result in direct read/write instead of a push/pop. This is generally used for debugging purposes only.

Note that pushing and popping data to and from the FIFOs by the CPU is only required when operating in Slave mode. When operating in DMA mode, the internal DMA will handle all pushing/popping of data to and from the TX and RX FIFOs.

28.3.3 FIFO and Queue Organization

The FIFOs in OTG_FS are primarily used to hold data packet payloads (the data field of USB Data packets). TX FIFOs are used to store data payloads that will be transmitted by OUT transactions in Host mode or IN transactions in Device mode. RX FIFOs are used to store received data payloads of IN transactions in Host mode or OUT transactions in Device mode. In addition to storing data payloads, RX FIFOs also store a **status entry** for each data payload. Each status entry contains information about a data payload such as channel number, byte count, and validity status. When operating in slave mode, status entries are also used to indicate various channel events.

The portion of SPRAM that can be used for FIFO allocation has a depth of 256 and a width of 35 bits (32 data bits plus 3 control bits). The multiple FIFOs used by each channel (in Host mode) or endpoint (in Device mode) are allocated into the SPRAM and can be dynamically sized.

28.3.3.1 Host Mode FIFOs and Queues

The following FIFOs are used when operating in Host mode (see Figure 28-3):

- **Non-periodic TX FIFO**: Stores data payloads of bulk and control OUT transactions for all channels.
- **Periodic TX FIFO**: Stores data payloads of interrupt or isochronous OUT transactions for all channels.
- **RX FIFO**: Stores data payloads of all IN transactions, and status entries that are used to indicate size of data payloads and transaction/channel events such as transfer complete or channel halted.

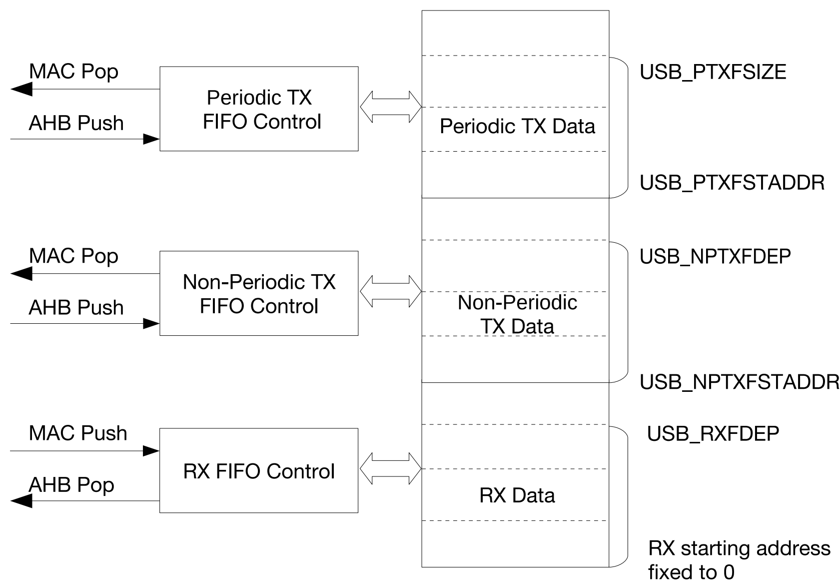


Figure 28-3. Host Mode FIFOs

In addition to FIFOs, Host mode also contains two request queues used to queue up the various transaction request from the multiple channels. Each entry in a request queue holds the IN/OUT channel number along with other information to perform the transaction (such as transaction type). Request queues are also used to queue other types of requests such as a channel halt request.

Unlike FIFOs, request queues are fixed in size and cannot be accessed directly by software. Rather, once a channel is enabled, requests will be automatically written to the request queue by the Host core. The order in which the requests are written into the queue determines the sequence of transactions on the USB.

Host mode contains the following request queues:

- **Non-periodic request queue**: Request queue for all non-periodic channels (bulk and control). The queue has a depth of four entries.
- **Periodic request queue**: Request queue for all periodic channels (interrupt and isochronous). The queue has a depth of eight entries.

When scheduling transactions, hardware will execute all requests on the periodic request queue first before executing requests on the non-periodic request queue.

28.3.3.2 Device Mode FIFOs

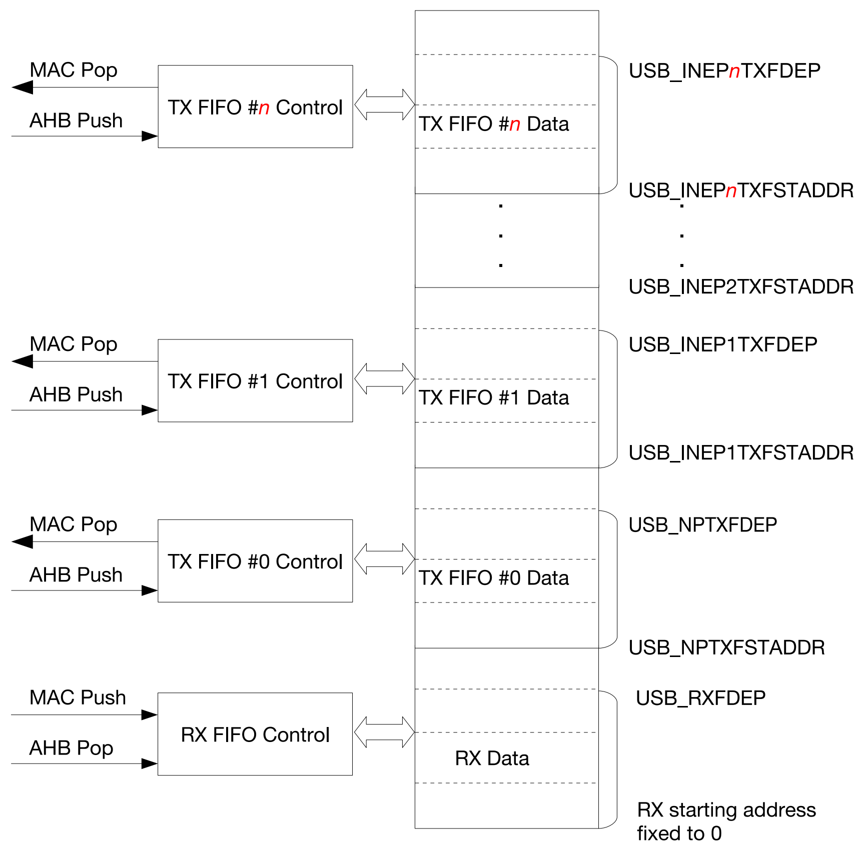


Figure 28-4. Device Mode FIFOs

The following FIFOs are used when operating in Device mode (See Figure 28-4):

- **RX FIFO**: Stores data payloads received in Data packet, and status entries (used to indicate size of those data payloads).
- **Dedicated TX FIFO**: Each active IN endpoint will have a dedicated TX FIFO used to store all IN data payloads of that endpoint, regardless of the transaction type (both periodic and non-periodic IN transactions).

Due to the dedicated FIFOs, Device mode does not use any request queues. Instead, the order of IN transactions are determined by the Host.

28.3.4 Interrupt Hierarchy

OTG_FS provides a single interrupt line which can be routed via the interrupt matrix to one of the CPUs. The interrupt signal can be unmasked by setting USB_GLBLINTRMSK. The OTG_FS interrupt is an OR of all bits in the USB_GINTSTS_REG register, and the bits in USB_GINTSTS_REG can be unmasked by setting the corresponding bits in the USB_GINTMSK_REG register. USB_GINTSTS_REG contains system level interrupts, and also specific bits for Host or Device mode interrupts, and OTG specific interrupts. OTG_FS interrupt sources are organized as Figure 28-5 shows.

The following bits of the USB_GINTSTS_REG register indicate an interrupt source lower in the hierarchy:

- **USB_PRTINT** indicates that the Host port has a pending interrupt. The USB_HPRT_REG register indicates

the interrupt source.

- **USB_HCHINT** indicates that one or more Host channels have a pending interrupt. Read the USB_HAINT_REG register to determine which channel(s) have a pending interrupt, then read the pending channel's USB_HCINT*n*_REG register to determine the interrupt source.
- **USB_OEPINT** indicates that one or more OUT endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which OUT endpoint(s) have a pending interrupt, then read the USB_DOEPINT*n*_REG register to determine the interrupt source.
- **USB_IEPINT** indicates that one or more IN endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which IN endpoint(s) are pending, then read the pending IN endpoint's USB_DIEPINT*n*_REG register to determine the interrupt source.
- **USB_OTGINT** indicates an OTG event has triggered an interrupt. Read the USB_GOTGINT_REG register to determine which OTG event(s) triggered the interrupt.

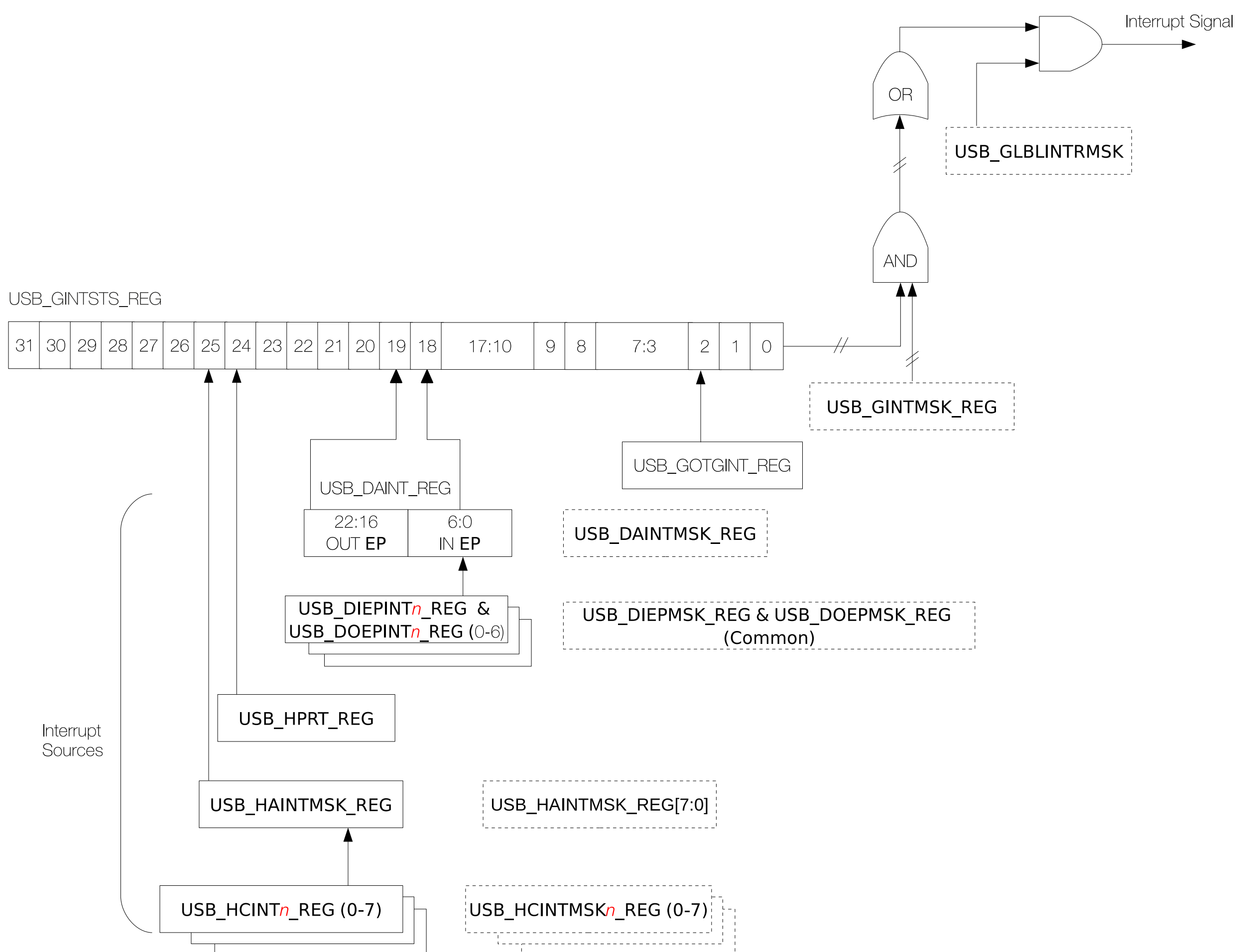


Figure 28-5. OTG_FS Interrupt Hierarchy

28.3.5 DMA Modes and Slave Mode

USB OTG supports three ways to access memory: Scatter/Gather DMA mode, Buffer DMA mode, and Slave mode.

28.3.5.1 Slave Mode

When operating in Slave mode, all data payloads must be pushed/popped to and from the FIFOs by the CPU.

- When transmitting a packet using IN endpoints or OUT channels, the data payload must be pushed into the corresponding endpoint or channel's TX FIFO.
- When receiving a packet, the packet's status entry must first be popped off the RX FIFO by reading USB_GRXSTSP_REG. The status entry should be used to determine the length of the packet's payload (in bytes). The corresponding number of bytes must then be manually popped off the RX FIFO by reading from the RX FIFO's memory region.

28.3.5.2 Buffer DMA Mode

Buffer mode is similar to Slave mode but utilizes the internal DMA to push and pop data payloads to the FIFOs.

- When transmitting a packet using IN endpoints or OUT channels, the data payload's address in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DIEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will push the data payload from memory into the TX FIFO of the channel or endpoint.
- When receiving a packet using OUT endpoints or IN channels, the address of an empty buffer in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DIEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will pop the data payload from RX FIFO into the buffer.

28.3.5.3 Scatter/Gather DMA Mode

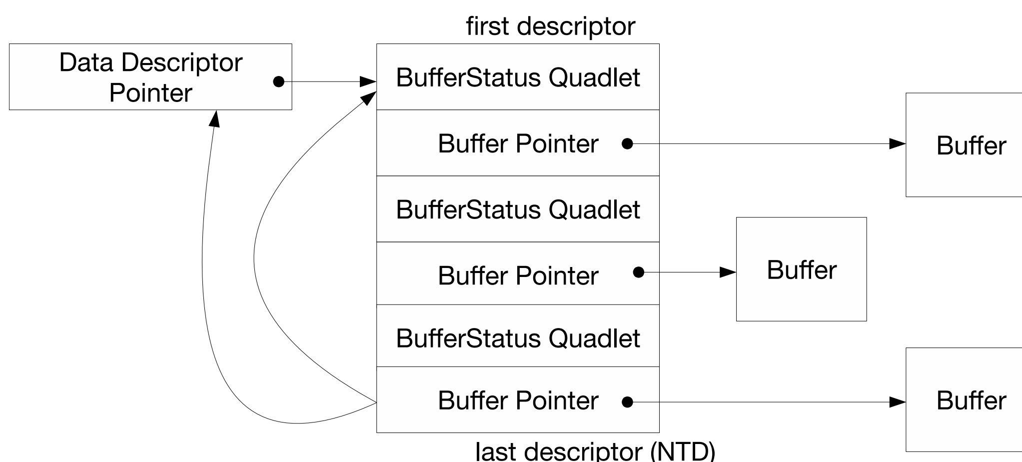


Figure 28-6. Scatter/Gather DMA Descriptor List

When operating in Scatter/Gather DMA mode, buffers containing data payloads can be scattered throughout memory. Each endpoint or channel will have a contiguous DMA descriptor list, where each descriptor contains a 32-bit pointer to the data payload or buffer and a 32-bit buffer descriptor (BufferStatus Quadlet). The data

payloads and buffers can correspond to a single transaction (i.e., < 1 MPS bytes) or an entire transfer (> 1 MPS bytes). (MPS: maximum packet size) The list is implemented as a ring buffer meaning that the DMA will return to the first entry when it encounters the last entry on the list.

- When transmitting a transfer/transaction using IN endpoints or OUT channels, the DMA will gather the data payloads from the multiple buffers and push them into a TX FIFO.
- When receiving a transfer/transaction using OUT endpoints or IN channels, the DMA will pop the received data payloads from the RX FIFO and scatter them to the multiple buffers pointed to by the DMA list entries.

28.3.6 Transaction and Transfer Level Operation

When operating in either Host or Device mode, communication can operate either at the transaction level or the transfer level.

28.3.6.1 Transaction and Transfer Level in DMA Mode

When operating at the transfer level in DMA Host mode, software is interrupted only when a channel has been halted. Channels are halted when their programmed transfer size has completed successfully, has received a STALL, or if there are excessive transaction errors (i.e., 3 consecutive transaction errors). When operating in DMA Device mode, all errors are handled by the controller core itself.

When operating at the transaction level in DMA mode, the transfer size is set to the size of one data packet (either a maximum packet size or a short packet size).

28.3.6.2 Transaction and Transfer Level in Slave Mode

When operating at the transaction level in Slave Mode, transfers are handled one transaction at a time. Each data payload should correspond to a single data packet, and software must determine whether a retry of the transaction is necessary based on the handshake response received on the USB (e.g., ACK or NAK).

The following table describes transaction level operation in Slave mode for both IN and OUT transactions.

Table 163: IN and OUT Transactions in Slave Mode

Host Mode	Device Mode
OUT Transactions	

Host Mode	Device Mode
1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_HCTSIZ*n*_REG register, enables the channel, then copies the packet's data payload into the TX FIFO. 2. When the last DWORD of the data payload has been pushed, the controller core will automatically write a request into the appropriate request queue. 3. If the transaction was successful, the USB_XFERCOMPL interrupt will be generated. If the transaction was unsuccessful, an error interrupt (e.g. USB_H_NACK*n*) will occur.	1. Software specifies the expected size of the data packet (1 MPS) and the number of data packets (1 data packet) in the USB_DOEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to transmit a packet to it. 2. The received packet will be pushed into the RX FIFO along with a packet status entry. 3. If the transaction was unsuccessful (e.g., due to a full RX FIFO), the endpoint will automatically NAK the incoming packet.

Host Mode	Device Mode
IN Transactions	
1. Software specifies the expected size of the data packet and the number of packets (1 data packet) in the USB_HCTSIZ*n*_REG register, then enables the channel. 2. The controller core will automatically write a request into the appropriate request queue. 3. If the transaction was successful, the received data along with a status entry should be written to the RX FIFO. If the transaction was unsuccessful, an error interrupt (e.g., USB_H_NACK*n*) will occur.	1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_DOEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to read the packet. 2. When the packet has been transmitted, the USB_XFERCOMPL interrupt will be generated.

When operating at the transfer level in Slave mode, one or more transaction-level operations can be pipelined thus being analogous to transfer level operation in DMA mode. Within pipelined transactions, multiple packets of the same transfer can be read/written from the FIFOs in single instance, thus preventing the need for interrupting the software on a per-packet basis.

Operating on a transfer level in Slave mode is similar to operating on the transaction-level, except the transfer size and packet count for each transfer in the USB_HCTSIZ*n*_REG or USB_DOEPTSIZ*n*_REG register will need to be set to reflect the entire transfer. After the channel or endpoint is enabled, multiple data packets worth of payloads should be written to or read from the TX or RX FIFOs respectively (given that there is enough space or enough data).

28.4 OTG

USB OTG allows OTG devices to act in the USB Host role or the USB Device role. Thus, OTG devices will typically have a Mini-AB or Micro-AB receptacle so that it can receive an A-plug or B-plug. OTG devices will become either an A-device or a B-device depending on whether an A-plug or a B-plug is connected.

- A-device defaults to the Host role (A-Host) whilst B-device defaults to the Device role (B-Peripheral).
- A-device and B-device may exchange roles by using the Host Negotiation Protocol (HNP), thus becoming A-peripheral and B-Host.
- A-device can turn off Vbus to save power. B-device can then wake up the A-device by requesting it to turn on Vbus and start a new session. This mechanism is called session request protocol (SRP).
- A-device always powers Vbus even if it is an A-peripheral.

OTG devices are able to determine whether they are connected to an A plug or a B plug using the ID pin of the plugs. The ID pin in A-plugs are pulled to ground whilst B-plugs have the ID pin left floating.

28.4.1 ID Pin Detection

Bit USB_CONIDSTS in register USB_GOTGCTL_REG indicates whether the OTG controller is an A-device (1'b0) or a B-device (1'b1). The USB_CONIDSTSCHNG interrupt will trigger whenever there is a change to USB_CONIDSTS (i.e., when a plug is connected or disconnected).

28.4.2 OTG Interface

The OTG_FS supports both the Session Request Protocol (SRP) and Host Negotiation Protocol (HNP) of the OTG Revision 1.3 specification. The OTG_FS controller core interfaces with the transceiver (internal or external) using the UTMI+ OTG interface. The UTMI+ OTG interface allows the controller core to manipulate the transceiver for OTG purposes (e.g., enabling/disabling pull-ups and pull-downs in HNP), and also allows the transceiver to indicate OTG related events. If an external transceiver is used instead, the UTMI+ OTG interface signals will be routed to the ESP32-S2's GPIOs instead. The UTMI+ OTG interface signals are described in Table 164.

Table 164: UTMI OTG Interface

Signal Name	I/O	Description
usb_otg_iddig_in	I	Mini A/B Plug Indicator. Indicates whether the connected plug is mini-A or mini-B. Valid only when usb_otg_idpullup is sampled asserted. 1'b0: Mini-A connected 1'b1: Mini-B connected
usb_otg_avalid_in	I	A-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for an A-peripheral session. The comparator thresholds are: 1'b0: Vbus <0.8 V 1'b1: Vbus = 0.2 V to 2.0 V
usb_otg_bvalid_in	I	B-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for a B-peripheral session. The comparator thresholds are: 1'b0: Vbus <0.8 V 1'b1: Vbus = 0.8 V to 4 V
usb_otg_vbusvalid_in	I	Vbus Valid. Indicates if the voltage Vbus is valid for A/B-device/peripheral operation. The comparator thresholds are: 1'b0: Vbus <4.4 V 1'b1: Vbus >4.75 V
usb_srp_sessend_in	I	B-device Session End. Indicates if the voltage Vbus is below the B-device Session End threshold. The comparator thresholds are: 1'b0: Vbus >0.8 V 1'b1: Vbus <0.2 V
usb_otg_idpullup	O	Analog ID input Sample Enable. Enables sampling the analog ID line. 1'b0: ID pin sampling disabled 1'b1: ID pin sampling enabled
usb_otg_dppulldown	O	D+ Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D+ line.
usb_otg_dmpulldown	O	D- Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D- line.
usb_otg_drvvbus	O	Drive Vbus. Enables driving Vbus to 5 V. 1'b0: Do not drive Vbus 1'b1: Drive Vbus
usb_srp_chrgvbus	O	Vbus Input Charge Enable. Directs the PHY to charge Vbus. 1'b0: Do not charge Vbus through a resistor 1'b1: Charge Vbus through a resistor (must be active for at least 30 ms)

Signal Name	I/O	Description
usb_srp_dischrgvbus	O	Vbus Input Discharge Enable. Directs the PHY to discharge Vbus. 1'b0: Do not discharge Vbus through a resistor. 1'b1: Discharge Vbus through a resistor (must be active for at least 50 ms).

28.4.3 Session Request Protocol (SRP)

28.4.3.1 A-Device SRP

Figure 28-7 illustrates the flow of SRP when the OTG_FS is acting as an A-device (i.e., default host and the device that powers Vbus).

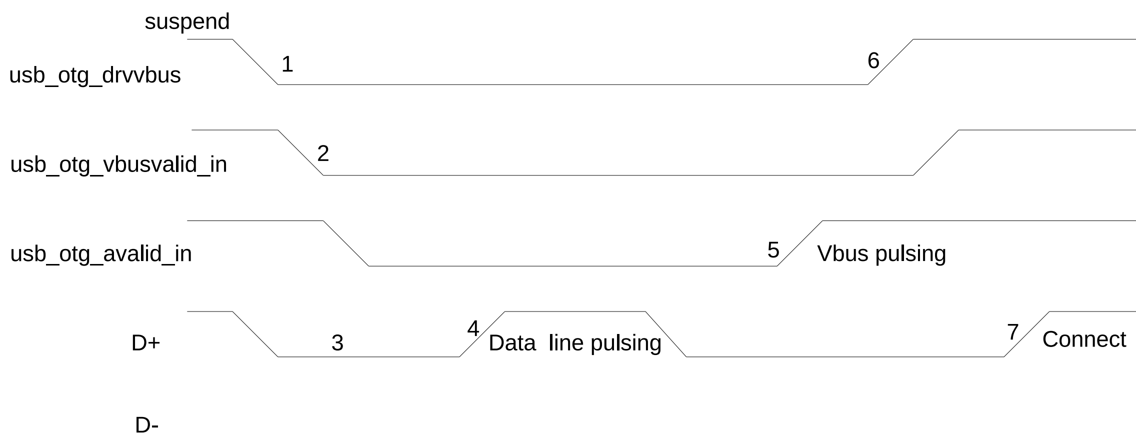


Figure 28-7. A-Device SRP

1. To save power, the application suspends and turns off port power when the bus is idle by writing to the Port Suspend (USB_PRTSUSP to 1'b0) and Port Power (USB_PRTPWR to 1'b0) bits in the Host Port Control and Status register.
2. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal.
3. The A-device must detect SE0 for at least 2 ms to start SRP when Vbus power is off.
4. To initiate SRP, the B-device turns on its data line pull-up resistor for 5 to 10 ms. The OTG_FS core detects data-line pulsing.
5. The device drives Vbus above the A-device session valid (2.0 V minimum) for Vbus pulsing. The OTF_FS core interrupts the application on detecting SRP. The Session Request Detected bit (USB_SESSREQINT) is set in Global Interrupt Status register.
6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by asserting usb_otg_vbusvalid_in signal.
7. When the USB is powered, the B-device connects, completing the SRP process.

28.4.3.2 B-Device SRP

Figure 28-8 illustrates the flow of SRP when the OTG_FS is acting as a B-device (i.e., does not power Vbus).

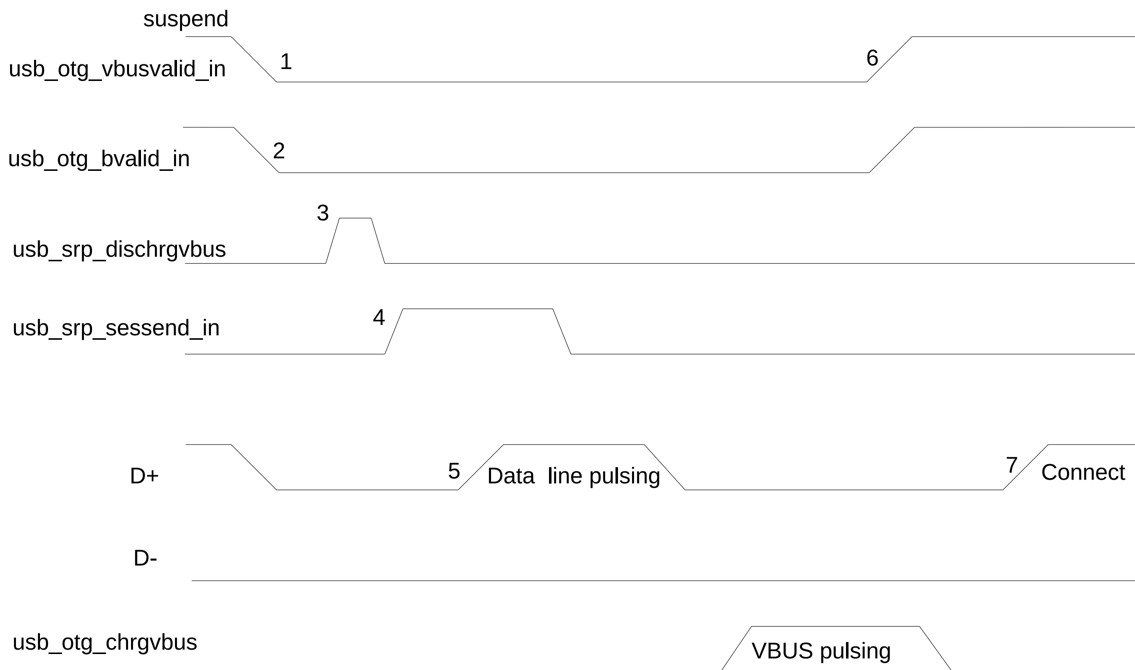


Figure 28-8. B-Device SRP

1. To save power, the host (A-device) suspends and turns off port power when the bus is idle. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal. The OTG_FS core sets the Early Suspend bit in the Core Interrupt register (USB_ERLYSUSP interrupt) after detecting 3 ms of bus idleness. Following this, the OTF_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register. The PHY indicates the end of the B-device session by deasserting the usb_otg_bvalid_in signal.
2. The OTG_FS core asserts the usb_otg_dischrgvbus signal to indicate to the PHY to speed up Vbus discharge.
3. The PHY indicates the session's end by asserting the usb_otg_sessend_in signal. This is the initial condition for SRP. The OTG_FS core requires 2 ms of SE0 before initiating SRP. For a USB 1.1 full-speed serial transceiver, the application must wait until Vbus discharges to 0.2 V after USB_BSESVLD is deasserted.
4. The application waits for 1.5 seconds (TB_SE0_SRP time) before initiating SRP by writing the Session Request bit (USB_SESREQ) in the OTG Control and Status register. The OTG_FS core performs data-line pulsing followed by Vbus pulsing.
5. The host (A-device) detects SRP from either the data-line or Vbus pulsing, and turns on Vbus. The PHY indicates Vbus power-on by asserting usb_otg_vbusvalid_in.
6. The OTG_FS core performs Vbus pulsing by asserting usb_srp_chrgvbus. The host (A-device) starts a new session by turning on Vbus, indicating SRP success. The OTG_FS core interrupts the application by setting the Session Request Success Status Change bit (USB_SESREQSC) in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
7. When the USB is powered, the OTG_FS core connects, completing the SRP process.

28.4.4 Host Negotiation Protocol (HNP)

28.4.4.1 A-Device HNP

Figure 28-9 illustrates the flow of HNP when the OTG_FS is acting as an A-device.

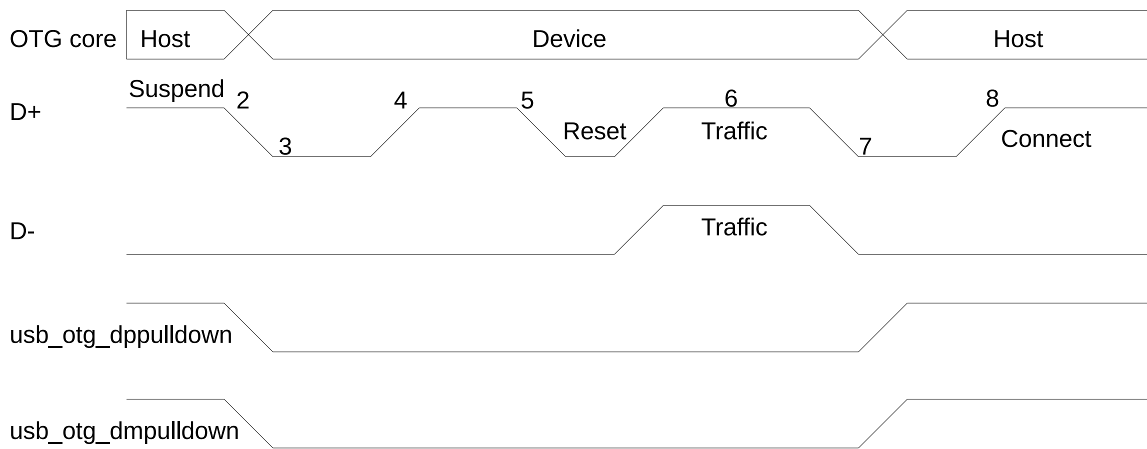


Figure 28-9. A-Device HNP

1. The OTG_FS core sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set Host Set HNP Enable bit (USB_HSTSETHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core that the B-device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be suspended. The OTG_FS core sets the Host Negotiation Detected interrupt (USB_HSTNEGDET) in the OTG Interrupt Status register, indicating the start of HNP. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate a device role. The PHY enables the D+ pull-up resistor, thus indicates a connection for the B-device. The application must read the Current Mode bit (USB_CURMOD_INT) in the OTG Control and Status register to determine Device mode operation.
4. The B-device detects the connection, issues a USB reset, and enumerates the OTG_FS core for data traffic.
5. The B-device continues the host role, initiating traffic, and suspends the bus when done. The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after detecting 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register.
6. In Negotiated mode, the OTG_FS core detects the suspend, disconnects, and switches back to the host role. The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.
7. The OTG_FS core sets the Connector ID Status Change interrupt (USB_CONIDSTS) in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the OTG_FS core's operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current Mode bit in the OTG Control and Status register to determine Host mode operation.
8. The B-device connects, completing the HNP process.

28.4.4.2 B-Device HNP

Figure 28-10 illustrates the flow of HNP when the OTG_FS is acting as an B-device.

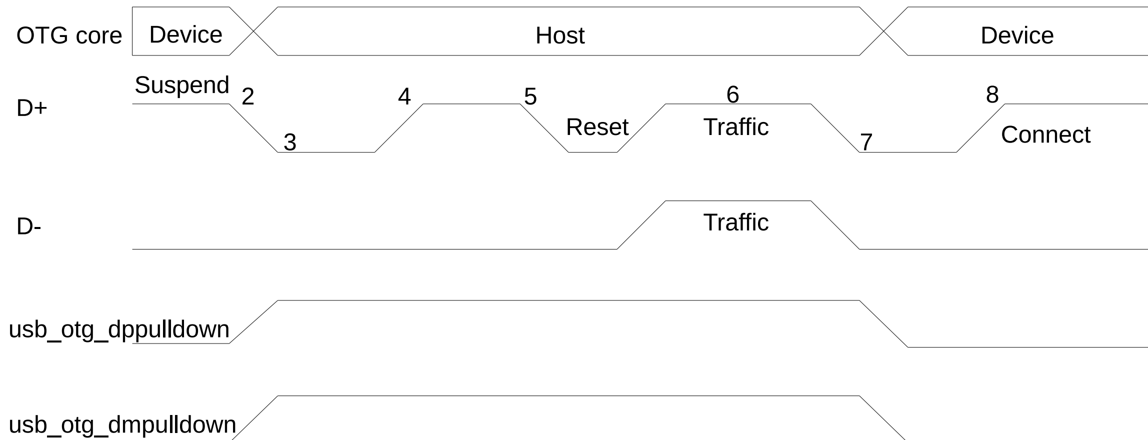


Figure 28-10. B-Device HNP

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The OTG_FS core's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate HNP support. The application sets the HNP Request bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core to initiate HNP.
2. When A-device has finished using the bus, it suspends the bus.
   (a) The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register. The OTG_FS core disconnects and the A-device detects SE0 on the bus, indicating HNP.
   (b) The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.
   (c) The A-device responds by activating its D+ pull-up resistor within 3 ms of detecting SE0. The OTG_FS core detects this as a connect.
   (d) The OTG_FS core sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register (USB_CONIDSTS), indicating the HNP status. The application must read the Host Negotiation Success bit (USB_HSTNEGSCS) in the OTG Control and Status register to determine host negotiation success. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine Host mode operation.
3. Program the USB_PRTPWR bit to 1'b1. This drives Vbus on the USB.
4. Wait for the USB_PRTCONNDET interrupt. This indicates that a device is connected to the port.
5. The application sets the reset bit (USB_PRTRST) and the OTG_FS core issues a USB reset and enumerates the A-device for data traffic.
6. Wait for the USB_PRTENCHNG interrupt.
7. The OTG_FS core continues the host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.

8. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate the assumption of the device role.
9. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine the Host mode operation.
10. The OTG_FS core connects, completing the HNP process.

28.5 Base Address

Users can access the USB core registers and the USB External Controller's registers (i.e., wrap registers) using the base addresses shown in Table 165. Note that PeriBus1 is more efficient but **features speculative reads**. This may cause issues when reading any "clear on read" registers such as USB_GRXSTSP_REG to pop a status entry off the RX FIFO. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 165: USB OTG Base Address

Register	Bus to Access Peripheral	Base Address
USB core registers	PeriBUS1	0x3F480000
	PeriBUS2	0x60080000
USB External Controller registers	PeriBUS1	0x3F439000
	PeriBUS2	0x60039000

29. Two-wire Automotive Interface (TWAI)

29.1 Overview

The Two-wire Automotive Interface (TWAI)® is a multi-master, multi-cast communication protocol with error detection and signaling and inbuilt message priorities and arbitration.The TWAI protocol is suited for automotive and industrial applications (Please see TWAI Protocol Description).

ESP32-S2 contains a TWAI controller that can be connected to a TWAI bus via an external transceiver. The TWAI controller contains numerous advanced features, and can be utilized in a wide range of use cases such as automotive products, industrial automation controls, building automation etc.

29.2 Features

ESP32-S2 TWAI controller supports the following features:

- compatible with ISO 11898-1 protocol
- Supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation
  - Normal
  - Listen Only (no influence on bus)
  - Self Test (transmissions do not require acknowledgment)
- 64-byte Receive FIFO
- Special transmissions
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self Reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports single and dual filter modes)
- Error detection and handling
  - Error counters
  - Configurable Error Warning Limit
  - Error Code Capture
  - Arbitration Lost Capture

29.3 Functional Protocol

29.3.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows for nodes to exchange messages in a latency bounded manner. A TWAI bus will has a following properties.

Single Channel and Non-Return-to-Zero: The bus consists of a single channel to carry bits, thus communication is half-duplex. Synchronization is also derived from this channel, thus extra channels (e.g., clock

or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a Dominant or Recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting a Dominant state will always override a another node transmitting a Recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential wiring).

Bit-Stuffing: Certain fields of TWAI messages are bit-stuffed. A Transmitter that transmits five consecutive bits of the same value should automatically insert a complementary bit. Likewise, a Receiver that receives five consecutive bits should treat the next bit as a stuff bit. Bit stuffing is applied to the following fields: SOF, Arbitration Field, Control Field, Data Field, and CRC Sequence (see Section 29.3.2 for more details).

Multi-cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (See Section 29.3.3).

Multi-master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before beginning its own transmission.

Message-Priorities and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The Arbitration Field of the message transmitted by each node is used to determine which node will win arbitration.

Error Detection and Signaling: Each node will actively monitor the bus for errors, and signal the detection errors by transmitting an Error Frame.

Fault Confinement: Each node will maintain a set of error counts that are incremented/decremented according to a set of rules. When the error counts surpass a certain threshold, a node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes within the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a Transmitter or a Receiver.

- A node originating a message is a Transmitter. The node remains a Transmitter until the bus is idle or until the node loses arbitration. Note that multiple nodes can be Transmitter if they have yet to lose arbitration.
- All nodes that are not Transmitters are Receivers.

29.3.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes. Messages are split into various frame types, and some frame types will have different frame formats. The TWAI protocol has of the following frame types:

- Data Frames
- Remote Frames
- Error Frames
- Overload Frames
- Interframe Space

The TWAI protocol has the following frame formats:

- Standard Frame Format (SFF) that consists of a 11-bit identifier
- Extended Frame Format (EFF) that consists of a 29-bit identifier

29.3.2.1 Data Frames and Remote Frames

Data Frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote Frames are used to nodes to request a Data Frame with the same Identifier from another node, thus does not contain any data bytes. However, Data Frames and Remote Frames share many common fields. Figure 29-1 illustrates the fields and sub fields of the different frames and formats.

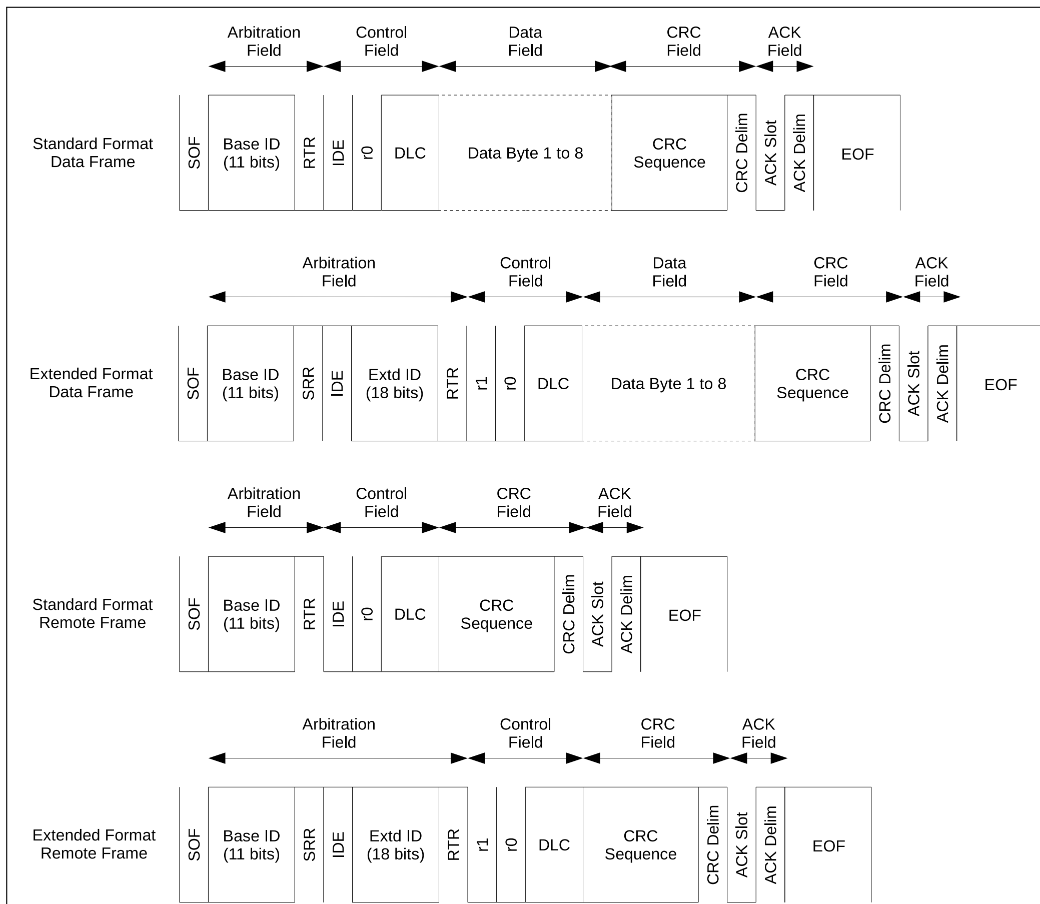


Figure 29-1. The bit fields of Data Frames and Remote Frames

Arbitration Field

When two or more nodes transmits a Data or Remote Frame simultaneously, the Arbitration Field is used to determine which node will win arbitration of the bus. During the Arbitration Field, if a node transmits a Recessive bit but observes a Dominant bit, this indicates that another node has overridden its Recessive bit. Therefore, the node transmitting the Recessive bit has lost arbitration of the bus and should immediately become a Receiver.

The Arbitration Field primarily consists of the Frame Identifier that is transmitted most significant bit first. Given that a Dominant bit represents a logical 0, and a Recessive bit represents a logical 1:

- A frame with the smallest ID value will always win arbitration.
- Given the same ID and format, Data Frames will always prevail over RTR Frames.

- Given the same first 11 bits of ID, a Standard Format Data Frame will prevail over an Extended Format Data Frame due to the SRR being recessive.

Control Field

The control field primarily consists of the DLC (Data Length Code) which indicates the number of payload data bytes for a Data Frame, or the number of requested data bytes for a Remote Frame. The DLC is transmitted most significant bit first.

Data Field

The Data Field contains the actual payload data bytes of a Data Frame. Remote Frames do not contain a Data Field.

CRC Field

The CRC Field primarily consists of a a CRC Sequence. The CRC Sequence is a 15-bit cyclic redundancy code calculated form the de-stuffed contents (everything from the SOF to the end of the Data Field) of a Data or Remote Frame.

ACK Field

The ACK Field primarily consists of an ACK Slot and an ACK Delim. The ACK Field is mainly intended for the receiver to send a message to a transmitter, indicating it has received an effective message.

Table 166: Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single Dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit Identifier for SFF, or the first 11-bits of the 29-bit Identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a Data Frame (Dominant) or a Remote Frame (Recessive). This means that a Remote Frame will always lose arbitration to a Data Frame given they have the same ID.
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The IDE (Identifier Extension) bit indicates whether the message is SFF (Dominant) or EFF (Recessive). This means that a SFF frame will always win arbitration over an EFF frame given they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18-bits of the 29-bit identifier for EFF.
r1	The r1 (reserved bit 1) is always Dominant.
r0	The r0 (reserved bit 0) is always Dominant.
DLC	The DLC (Data Length Code) is 4-bits and should have a value from 0 to 8. Data Frames use the DLC to indicate the number data bytes in the Data Frame. Remote Frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of Data Frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted most significant bit first.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.

Data/Remote Frames	Description
CRC Delim	The CRC Delim (CRC Delimeter) is a single Recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) that intended for Receiver nodes to indicate that the Data or Remote Frame was received without issue. The Transmitter node will send a Recessive bit in the ACK Slot and Receiver nodes should override the ACK Slot with a Dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimeter) is a single Recessive bit.
EOF	The EOF (End of Frame) marks the end of a Data or Remote Frame, and consists of seven Recessive bits.

29.3.2.2 Error and Overload Frames

Error Frames

Error Frames are transmitted when a node detects a Bus Error. Error Frames notably consist of an Error Flag which is made up of 6 consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a Bus Error and transmits an Error Frame, all other nodes will then detect a Stuff Error and transmit their own Error Frames in response. This has the effect of propagating the detection of a Bus Error across all nodes on the bus. When a node detects a Bus Error, it will transmit an Error Frame starting on the next bit. However, if the type of Bus Error was a CRC Error, then the Error Frame will start at the bit following the ACK Delim (see Section 29.3.3). The following Figure 29-2 shows the various fields of an Error Frame:

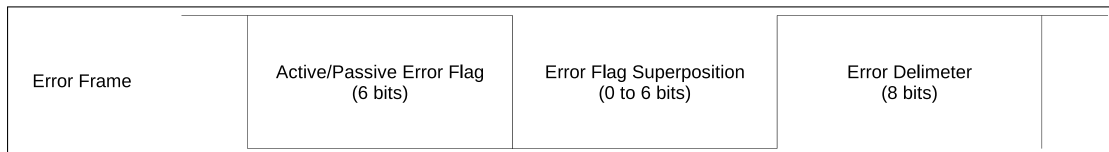


Figure 29-2. Various Fields of an Error Frame

Table 167: Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of 6 Dominant bits and the Passive Error Flag consisting of 6 Recessive bits (unless overridden by Dominant bits of other nodes). Active Error Flags are sent by Error Active nodes, whilst Passive Error Flags are sent by Error Passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first Recessive bit is detected (i.e., the first it of the Delimeter).
Error Delimeter	The Delimeter field marks the end of the Error/Overload Frame, and consists of 8 Recessive bits.

Overload Frames

An Overload Frame has the same bit fields as an Error Frame containing an Active Error Flag. The key difference

is in the conditions that can trigger the transmission of an Overload Frame. Figure 29-3 below shows the bit fields of an Overload Frame.

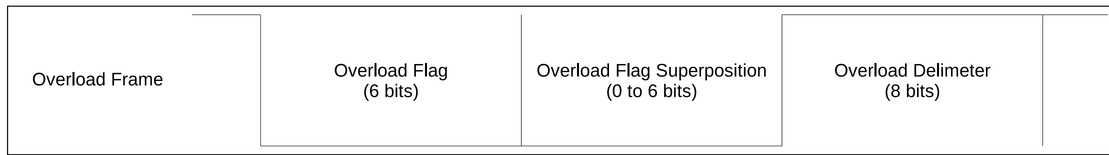


Figure 29-3. The Bit Fields of an Overload Frame

Table 168: Overload Frame

Overload Frame	Description
Overload Flag	Consists of 6 Dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimeter	Consists of 8 Recessive. Same as an Error Delimeter.

Overload Frames will be transmitted under the following conditions:

1. The internal conditions of a Receiver requires a delay of the next Data or Remote Frame.
2. Detection of a Dominant bit at the first and second bit of Intermission.
3. If a Dominant bit is detected at the eighth (last) bit of an Error Delimeter. Note that in this case, TEC and REC will not be incremented (See Section 29.3.3).

Transmitting an overload frame due to one of the conditions must also satisfy the following rules:

- Transmitting an Overload Frame due to condition 1 must only be started at the first bit of Intermission.
- Transmitting an Overload Frame due to condition 2 and 3 must start one bit after the detecting the Dominant bit of the condition.
- A maximum of two Overload frames may be generated in order to delay the next Data or Remote Frame.

29.3.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Data Frames and Remote Frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (Data Frame, Remote Frame, Error Frame, Overload Frame). However, Error Frames and Overload Frames do not need to be separated from preceding frames.

Figure 29-4 shows the fields within an Interframe Space:

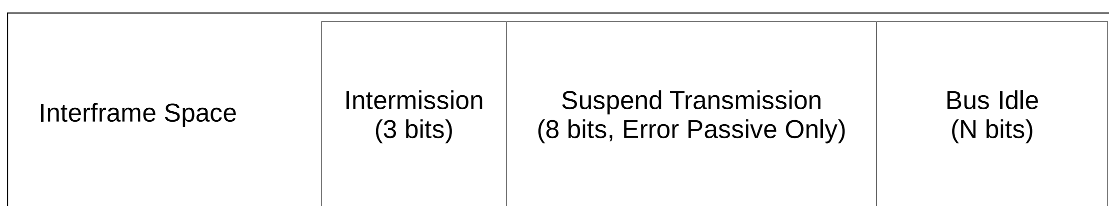


Figure 29-4. The Fields within an Interframe Space

Table 169: Interframe Space

Interframe Space	Description
Intermission	The Intermission consists of 3 Recessive bits.
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of 8 Recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following Intermission.

29.3.3 TWAI Errors

29.3.3.1 Error Types

Bus Errors in TWAI are categorized into one of the following types:

Bit Error
A Bit Error occurs when a node transmits a bit value (i.e., Dominant or Recessive) but the opposite bit is detected (e.g., a Dominant bit is transmitted but a Recessive is detected). However, if the transmitted bit is Recessive and is located in the Arbitration Field or ACK Slot or Passive Error Flag, then detecting a Dominant bit will not be considered a Bit Error.

Stuff Error
A stuff error is detected when 6 consecutive bits of the same value are detected (thus violating the bit-stuffing encoding).

CRC Error
A Receiver of a Data or Remote Frame will calculate a CRC based on the bits it has received. A CRC error occurs when the CRC calculated by the Receiver does not match the CRC sequence in the received Data or Remote Frame.

Form Error
A Form Error is detected when a fixed-form bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be Dominant.

Acknowledgement Error
An Acknowledgment Error occurs when a Transmitter does not detect a Dominant bit at the ACK Slot.

29.3.3.2 Error States

TWAI nodes implement fault confinement by each maintaining two error counters, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states.

Error Active
An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication, but can only transmit an Passive Error Flag when it detects an error. Error Passive nodes that have transmitted a Data or Remote Frame must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit anything).

29.3.3.3 Error Counters

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply for a given message transfer.**

1. When a Receiver detects an error, the REC will be increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a Receiver detects a Dominant bit as the first bit after sending an Error Flag, the REC will be increased by 8.
3. When a Transmitter sends an Error Flag the TEC is increased by 8. However, the following scenarios are exempt form this rule:
    - If a Transmitter is Error Passive that detects an Acknowledgment Error due to not detecting a Dominant bit in the ACK slot, it should send a Passive Error Flag. If no Dominant bit is detected in that Passive Error Flag, the TEC should not be increased.
    - A Transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the offending bit should have been Recessive but was monitored as Dominant, then the TEC should not be increased.
4. If a Transmitter detects a Bit Error whilst sending an Active Error Flag or Overload Flag, the REC is increased by 8.
5. If a Receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
6. Any node tolerates up to 7 consecutive Dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive Dominant bit (when sending an Active Error Flag or Overload Flag), or the 8th consecutive Dominant bit following a Passive Error Flag, a Transmitter will increase its TEC by 8 and a Receiver will increase its REC by 8. Each additional eight consecutive Dominant bits will also increase the TEC (for Transmitters) or REC (for Receivers) by 8 as well.
7. When a Transmitter successfully transmits a message (getting ACK and no errors until the EOF is complete), the TEC is decremented by 1, unless the TEC is already at 0.
8. When a Receiver successfully receives a message (no errors before ACK Slot, and successful sending of ACK), the REC is decremented.
    - If the REC was between 1 and 127, the REC is decremented by 1.
    - If the REC was greater than 127, the REC is set to 127.
    - If the REC was 0, the REC remains 0.
9. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. The error condition that causes a node to become Error Passive will cause the node to send an Active Error Flag.

Note that once the REC has reached to 128, any further increases to its value are irrelevant until the REC returns to a value less than 128.

10. A node becomes Bus Off when its TEC is greater than or equal to 256.
11. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
12. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive Recessive bits on the bus.

29.3.4 TWAI Bit Timing

29.3.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- The **Nominal Bit Rate** is defined as number of bits transmitted per second from an ideal Transmitter and without any synchronization.
- The **Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a fixed unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 29-5 illustrates the segments within a single Nominal Bit Time.

TWAI Controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed at every Time Quanta. If two consecutive Time Quantas have different bus states (i.e., Recessive to Dominant or vice versa), this will be considered an edge. When the bus is analyzed at the intersection of PBS1 and PBS2, this is considered the Sample Point and the sampled bus value is considered the value of that bit.

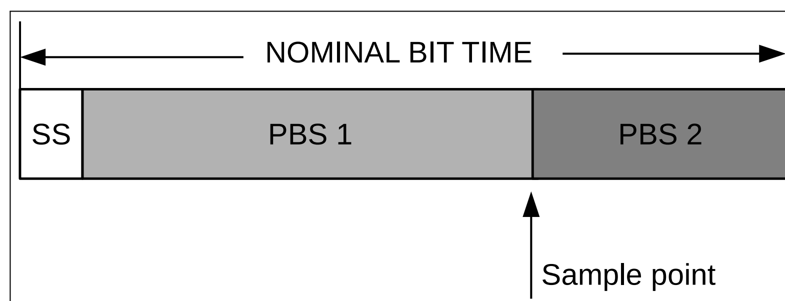


Figure 29-5. Layout of a Bit

Table 170: Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.
PBS2	PBS2 (Phase Buffer Segment 2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

29.3.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct for Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules.

- Only one synchronization may occur in a single bit time.
- Synchronizations only occurs on Recessive to Dominant edges.

Hard Synchronization

Hard Synchronization occurs on the Recessive to Dominant edges during Bus Idle (i.e., the SOF bit). All nodes will restart their internal bit timings such that the Recessive to Dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on Recessive to Dominant edges not during Bus Idle. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is a programmable.

- When the magnitude of the Phase Error is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by **e** number of Time Quanta. This has a same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater to the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

29.4 Architectural Overview

The ESP32-S2 contains a TWAI Controller. Figure 29-6 shows the major functional blocks of the TWAI Controller.

29.4.1 Registers Block

The ESP32-S2 CPU accesses peripherals as 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration options for the TWAI controller such as bit rates, operating mode, Acceptance Filter etc. Configuration registers can only be modified whilst the TWAI controller is in Reset Mode (See Section 29.5.1).

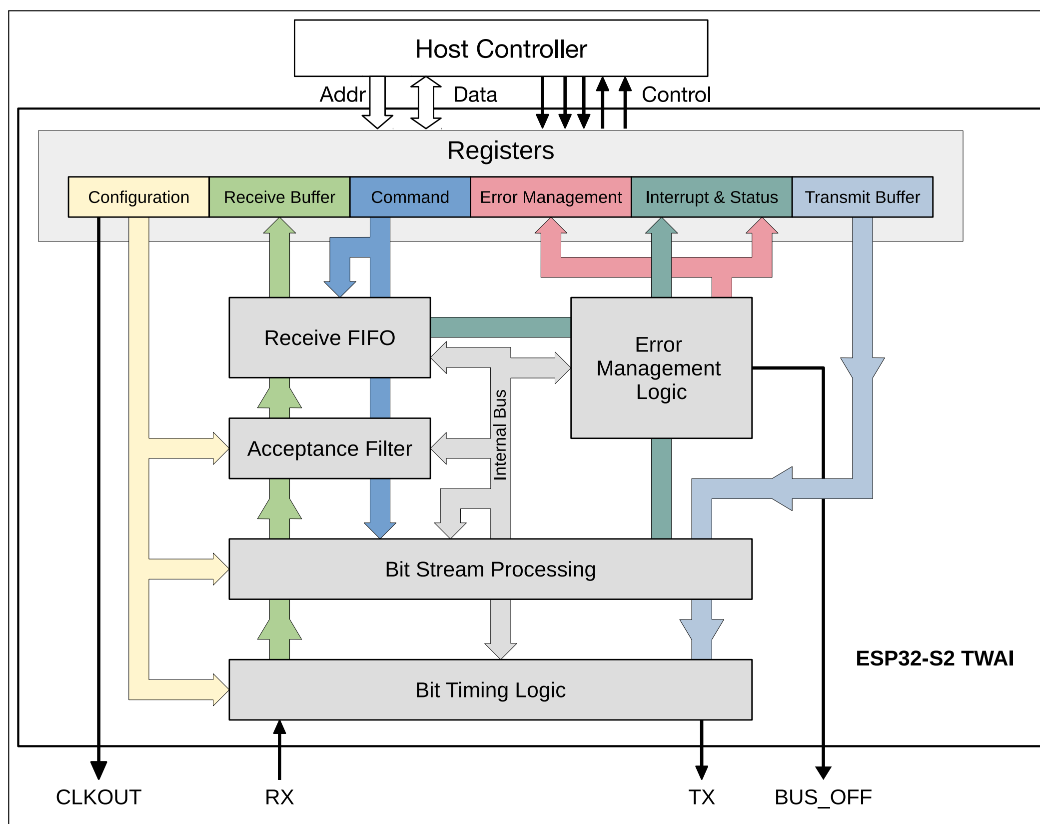


Figure 29-6. TWAI Overview Diagram

Command Register

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation Mode (see section 29.5.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management registers include error counters and capture registers. The error counter registers represent TEC and REC values. The capture registers will record information about instances where TWAI controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window into Receive FIFO mapping to the first received message in the Receive FIFO to the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset Mode, the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation Mode:

- All reads to the address range maps to the Receive Buffer registers.
- All writes to the address range maps to the Transmit Buffer registers.

29.4.2 Bit Stream Processor

The Bit Stream Processing (BSP) module is responsible for framing data from the Transmit Buffer (e.g. bit stuffing and additional CRC fields) and generating a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

29.4.3 Error Management Logic

The Error Management Logic (EML) module is responsible for updating the TEC and REC, recording error information like error types and positions, and updating the error state of the TWAI Controller such that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

29.4.4 Bit Timing Logic

The Bit Timing Logic (BTL) module is responsible for transmitting and receiving messages at the configured bit rate. The BTL module also handles synchronization of out of phase bits such that communication remains stable. A single bit time consists of multiple programmable segments that allows users to set the length of each segment to account for factors such as propagation delay and controller processing time etc.

29.4.5 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter's registers can be programmed to specify a single filter, or specify two separate filters (dual filter mode).

29.4.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (internal to the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13-bytes). When the Receive FIFO is full (or does not have enough space to store the next received message in its entirety), the Overrun Interrupt will be triggered, and any subsequent received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After clearing, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

29.5 Functional Description

29.5.1 Modes

The ESP32-S2 TWAI controller has two working modes: Reset Mode and Operation Mode. Reset Mode and Operation Mode are entered by setting the TWAI_RESET_MODE bit to 1 or 0 respectively.

29.5.1.1 Reset Mode

Entering Reset Mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset Mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset Mode, the TWAI controller will not be able to transmit any messages (including error signaling). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will also not be able to receive any messages.

29.5.1.2 Operation Mode

Entering Operation Mode essentially connects the TWAI controller to the TWAI bus, and write protects the TWAI controller's configuration registers ensuring the configuration stays consistent during operation. When in Operation Mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operating sub-mode the TWAI controller was configured with. The TWAI controller supports the following operating sub-modes:

- **Normal Mode:** The TWAI controller can transmit and receive messages including error signaling (such as Error and Overload Frames).
- **Self Test Mode:** Like Normal Mode, but the TWAI controller will consider the transmission of a Data or RTR Frame successful even if it was not acknowledged. This is commonly used when self testing the TWAI controller.
- **Listen Only Mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitors.

Note that when exiting Reset Mode (i.e., entering Operation Mode), the TWAI controller must wait for 11 consecutive Recessive bits to occur before being able to fully connect the TWAI bus (i.e., be able to transmit or receive).

29.5.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset Mode. The bit rate configuration is located in TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 171 illustrates the bit fields of TWAI_BUS_TIMING_0_REG.

Table 171: Bit Information of TWAI_CLOCK_DIVIDER_REG; TWAI Address 0x18

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	......	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.13	BRP.12	......	BRP.1	BRP.0

Notes:

- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1).
- BRP: The TWAI Time Quanta clock is derived from a prescaled version of the APB clock that is usually 80 MHz. The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock period and t_{CLK} is APB clock period :

$$t_{Tq} = 2 \times t_{CLK} \times (2^{13} \times BRP.13 + 2^{12} \times BRP.12 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$$

The following Table 172 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 172: Bit Information of TWAI_BUS_TIMING_1_REG; TWAI Address 0x1c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined according to the following equation: (8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1).
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined according to the following equation: (4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1).
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses where filtering spikes on the bus line is beneficial.

29.5.3 Interrupt Management

The ESP32-S2 TWAI controller provides seven interrupts, each represented by a single bit in the TWAI_INT_RAW_REG. For a particular interrupt to be triggered (i.e., its bit in TWAI_INT_RAW_REG set to 1), the interrupt's corresponding enable bit in TWAI_INT ENA_REG must be set.

The TWAI controller provides the seven following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt
- Arbitration Lost Interrupt
- Bus Error Interrupt

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INT_RAW_REG, and deasserted when all bits in TWAI_INT_RAW_REG are cleared. The majority of interrupt bits in TWAI_INT_RAW_REG are automatically cleared when the register is read. However, the Receive Interrupt is an exception and can only be cleared the Receive FIFO is empty.

29.5.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_CNT_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUF command bit.

29.5.3.2 Transmit Interrupt (TXI)

The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer to be transmitted. The Transmit Buffer becomes free under the following scenarios:

- A message transmission has completed successfully (i.e., Acknowledged without any errors). Any failed messages will automatically be retried.
- A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_TX_COMPLETE bit).
- A message transmission was aborted using the TWAI_ABORT_TX command bit.

29.5.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST and TWAI_BUS_OFF_ST bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_ERR_ST and TWAI_BUS_OFF_ST at the moment the EWI is triggered.

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
  - If the TWAI controller was previously in the Bus Recovery state, it indicates that Bus Recovery has completed successfully.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).
- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

29.5.3.4 Data Overrun Interrupt (DOI)

The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages.

The DOI is only triggered on the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI will only be able to trigger again when all received messages (valid or overrun) have been cleared.

29.5.3.5 Error Passive Interrupt (TXI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller transitions from Error Active to Error Passive, or vice versa.

29.5.3.6 Arbitration Lost Interrupt (ALI)

The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB LOST CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record new bit location until it is cleared (via a read from the CPU).

29.5.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever TWAI controller observes an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).

29.5.4 Transmit and Receive Buffers

29.5.4.1 Overview of Buffers

Table 173: Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
TWAI address	Content	TWAI address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 173 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation Mode. CPU write operations will access the Transmit Buffer registers, and CPU read operations will access the Receive Buffer registers. However, both buffers share the exact same register layout and fields to represent a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQ bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQ bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQ and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map to the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUF bit in TWAI_CMD_REG so that the next message in the Receive FIFO will be loaded in to the Receive Buffer registers.

29.5.4.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 174.

Table 174: TX/RX Frame Information (SFF/EFF) TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	XDLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

- FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when FF bit is 1, and SFF when FF bit is 0.
- RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a Data Frame or a Remote Frame. The message is a Remote Frame when the RTR bit is 1, and a Data Frame when the RTR bit is 0.
- DLC: The Data Length Code (DLC) field specifies the number of data bytes for a Data Frame, or the number of data bytes to request in a Remote Frame. TWAI Data Frames are limited to a maximum payload of 8 data bytes, thus the DLC should range anywhere from 0 to 8.
- X: Don't care, can be any value.

29.5.4.3 Frame Identifier

The Frame Identifier fields is 2 bytes (11-bits) if the message is SFF, and 4 bytes (29-bits) if the message is EFF.

The Frame Identifier fields for an SFF (11-bits) message is shown in Table 175-176.

Table 175: TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 176: TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	X[1]	X[2]	X[2]	X[2]	X[2]

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 177-180.

Table 177: TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 178: TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 179: TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 180: TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

29.5.4.4 Frame Data

The Frame Data fields contains the payload of transmitted or received a Data Frame, and can range from 0 to 8 bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than 8, the number of valid bytes would still be limited to 8. Remote Frames do not have data payloads, thus the Frame Data fields will be unused.

For example, when transmitting a Data Frame with 5 data bytes, the CPU should write a value of 5 to the DLC field, and then fill in data bytes 1 to 5 in the Frame Data fields. Likewise, when receiving a Data Frame with a DLC of 5, only data bytes 1 to 5 will contain valid payload data for the CPU to read.

29.5.5 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13-bytes of space in the Receive FIFO, and their byte layout is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO. When the TWAI controller receives a message, it will increment the value of TWAI_RX_MESSAGE_COUNTER up to a maximum of 64. If there is adequate space in the Receive FIFO, the message contents will be written into the Receive FIFO. Once a message has been read from the Receive Buffer, the TWAI_RELEASE_BUF bit should be set. This will decrement TWAI_RX_MESSAGE_COUNTER and free the space occupied by the first message in the Receive FIFO. The

Receive Buffer will then map to the next message in the Receive FIFO. A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks the adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs...

- Whatever free space is left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- On the first message that causes the Receive FIFO to overrun, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The RX FIFO will internally mark overrun messages as invalid. The TWAI_MISS_ST bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUF must be called repeatedly called until TWAI_RX_MESSAGE_COUNTER is 0. This has the effect of freeing all valid messages in the Receive FIFO and clearing all overrun messages.

29.5.6 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows for a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) due to the TWAI Controller only needing to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset Mode, due to those registers sharing the same address space as the Transmit Buffer and Receive Buffer registers.

The registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Code value specifies a bit pattern in which each filtered bit of the message must match in order for the message to be accepted. The Mask value is able to mask out certain bits of the Code value (i.e., set as "Don't Care" bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 29-7.

The TWAI Controller Acceptance Filter allows the 32-bit Code and Mask values to either define a single filter (i.e., Single Filter Mode), or two filters (i.e., Dual Filter Mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on whether Single Filter Mode is enabled, and the received message (i.e., SFF or EFF).

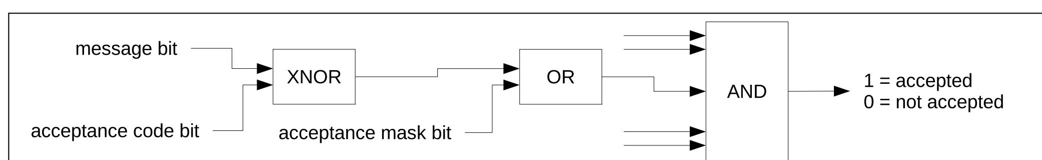


Figure 29-7. Acceptance Filter

29.5.6.1 Single Filter Mode

Single Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of a Data or Remote Frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 29-8 illustrates how the 32-bit code and mask values will be interpreted under Single Filter Mode.

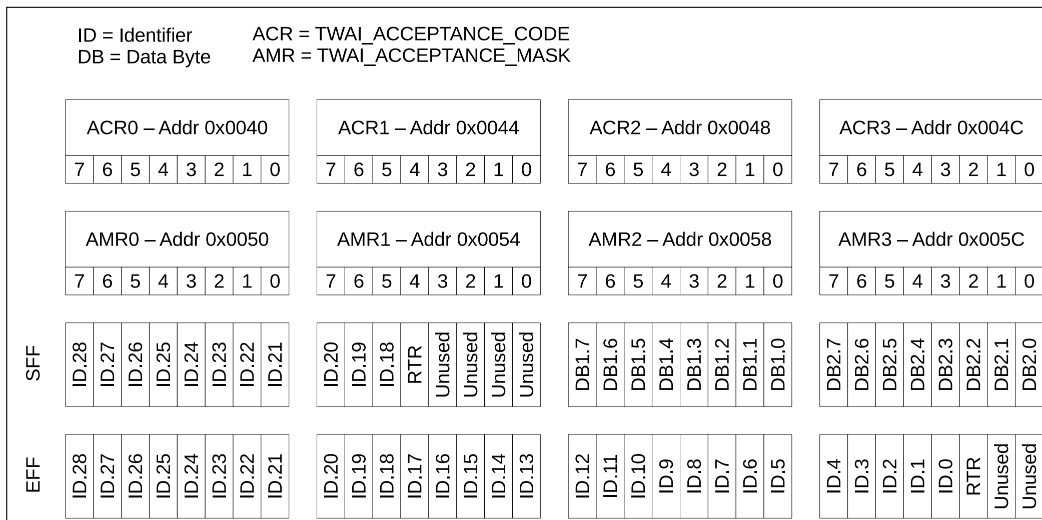


Figure 29-8. Single Filter Mode

29.5.6.2 Dual FIlter Mode

Dual Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define a two separate filters, referred to as filter 1 or two. Under Dual Filter Mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of a Data or Remote Frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)
- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 29-9 illustrates how the 32-bit code and mask values will be interpreted under Dual Filter Mode.

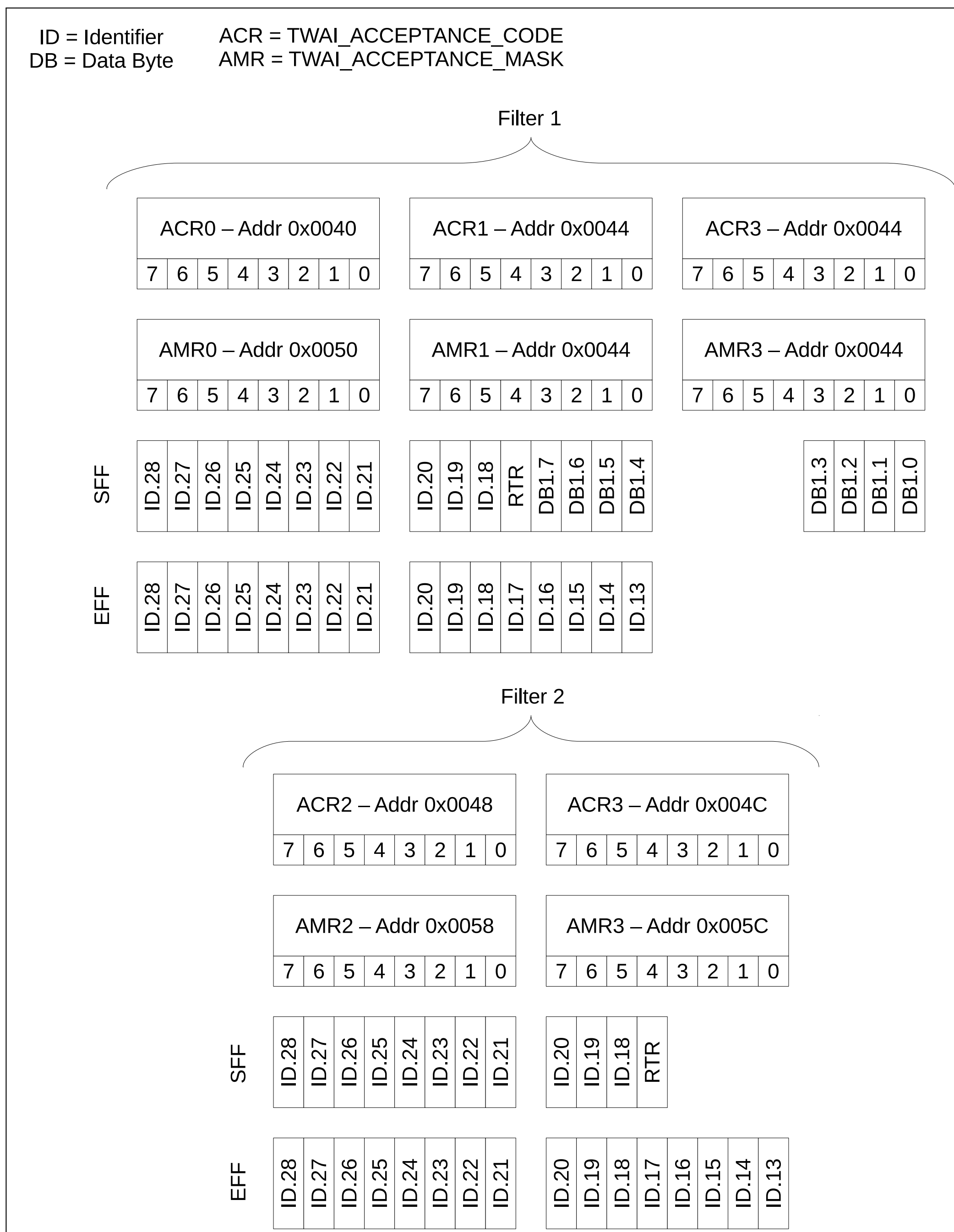


Figure 29-9. Dual Filter Mode

29.5.7 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Count (TEC) and Receive Error Count (REC). The value of both error counts determine the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in the TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and can be read by the CPU at anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn the user regarding the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_ERR_ST, and TWAI_BUS_OFF_ST. Certain changes to these values and bits will also trigger

interrupts, thus allowing the users to be notified of error state transitions (see section 29.5.3). The following figure 29-10 shows the relation between the error states, values and bits, and error state related interrupts.

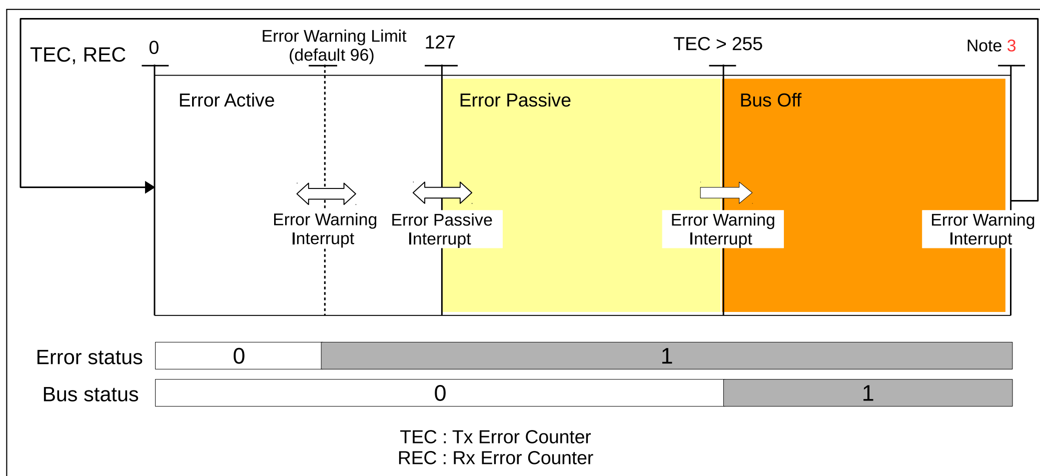


Figure 29-10. Error State Transition

29.5.7.1 Error Warning Limit

The Error Warning Limit (EWL) feature is a configurable threshold value for the TEC and REC, where if exceeded, will trigger an interrupt. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in the TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset Mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96. When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_ERR_ST bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_ERR_ST bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_ERR_ST bit (or the TWAI_BUS_OFF_ST) changes.

29.5.7.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

29.5.7.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_BUS_OFF_ST bit to 1
- Enter Reset Mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_BUS_OFF_ST bit (or the TWAI_ERR_ST bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off recovery. Bus-Off recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive Recessive bits on the bus. To initiate Bus-Off recovery (after entering the Bus-Off state), the TWAI controller should enter Operation Mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off recovery by decrementing the TEC each time the TWAI controller observes 11 consecutive Recessive bits. When Bus-Off recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

29.5.8 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in the TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from the TWAI_ERR_CODE_CAP_REG.

The following Table 181 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 181: Bit Information of TWAI_ERR_CODE_CAP_REG; TWAI Address 0x30

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error: 00 for bit error, 01 for form error, 10 for stuff error, 11 for other type of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error: 0 for Transmitter, 1 for Receiver.
- SEG: The Error Segment (SEG) indicates which segment of the TWAI message (i.e., bit position) the bus error occurred at.

The following Table 182 shows how to interpret the SEG.0 to SEG.4 bits.

Table 182: Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	bit SRTR[1]
0	0	1	0	1	bit IDE[2]
0	0	1	1	1	ID.17 to ID.13
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimeter
1	1	0	0	1	acknowledge slot
1	1	0	1	1	acknowledge delimeter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimeter
1	1	1	0	0	overload flag

Notes:

- Bit RTR: under Standard Frame Format.
- Identifier Extension Bit: 0 for Standard Frame Format.

29.5.9 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in the TWAI_ARB LOST CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent loses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in the TWAI_ARB LOST CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 183 illustrates the bit fields of the TWAI_ERR_CODE_CAP_REG whilst Figure 29-11 illustrates the bit positions of a TWAI message.

Table 183: Bit Information of TWAI_ARB LOST CAP_REG; TWAI Address 0x2c

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

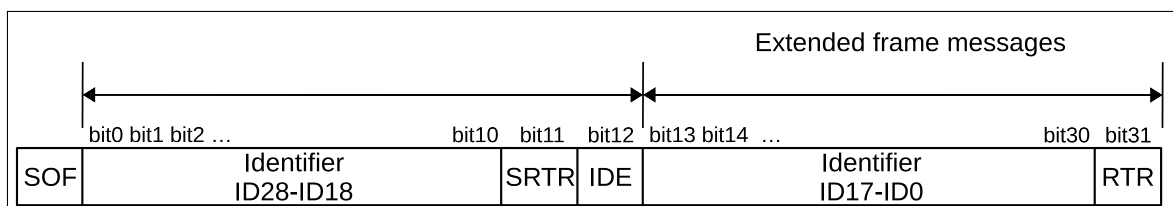


Figure 29-11. Positions of Arbitration Lost Bits

29.6 Base Address

Users can access the TWAI with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter: 3 *System and Memory*.

Table 184: TWAI Base Address

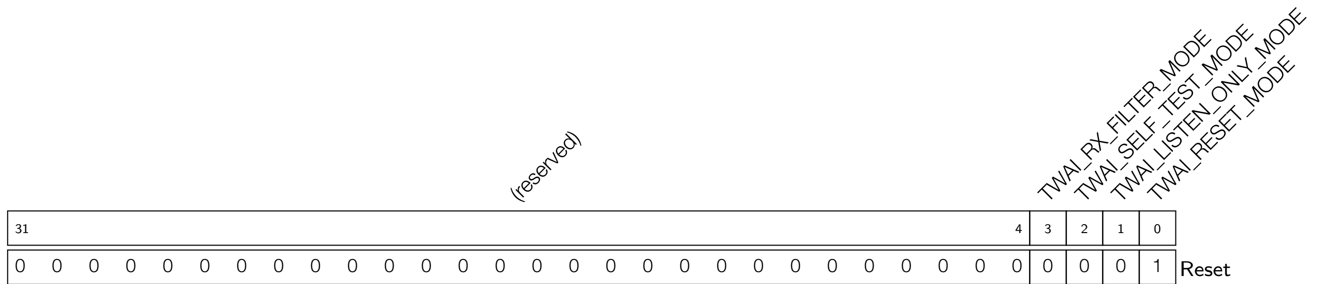
Bus to Access Peripheral	Base Address
PeriBUS1	0x6002B000
PeriBUS2	0x3FC2B000

29.7 Register Summary

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	RO | R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	RO | R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	RO | R/W
TWAI_DATA_0_REG	Data Register 0	0x0040	WO | R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	WO | R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	WO | R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	WO | R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	WO | R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	WO | R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	WO | R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	WO | R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	WO | RO
TWAI_DATA_9_REG	Data Register 9	0x0064	WO | RO
TWAI_DATA_10_REG	Data Register 10	0x0068	WO | RO
TWAI_DATA_11_REG	Data Register 11	0x006C	WO | RO
TWAI_DATA_12_REG	Data Register 12	0x0070	WO | RO
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	varies
Control Registers			
TWAI_CMD_REG	Command Register	0x0004	WO
Status Registers			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB LOST CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	RO | R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	RO | R/W
TWAI_RX_MESSAGE_CNT_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INT_RAW_REG	Interrupt Register	0x000C	RO
TWAI_INT ENA_REG	Interrupt Enable Register	0x0010	R/W

29.8 Register Description

Register 29.1: TWAI_MODE_REG (0x0000)



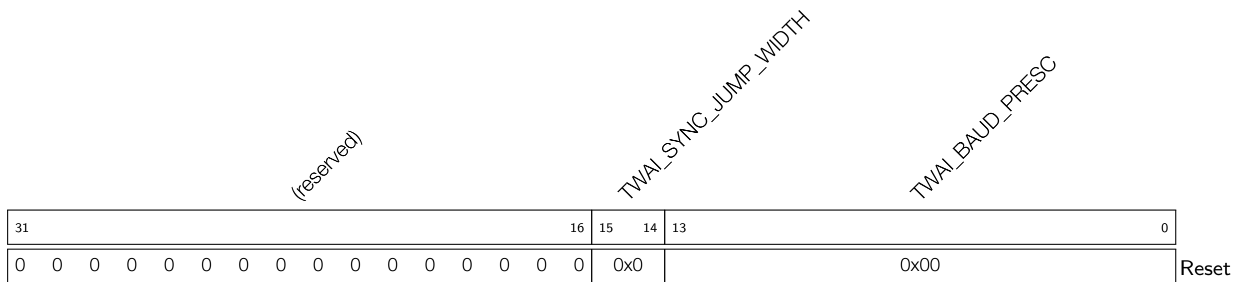
TWAI_RESET_MODE This bit is used to configure the operating mode of the TWAI Controller. 1: Reset mode; 0: Operating mode (R/W)

TWAI_LISTEN_ONLY_MODE 1: Listen only mode. In this mode the nodes will only receive messages from the bus, without generating the acknowledge signal nor updating the RX error counter. (R/W)

TWAI_SELF_TEST_MODE 1: Self test mode. In this mode the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self reception request command. (R/W)

TWAI_RX_FILTER_MODE This bit is used to configure the filter mode. 0: Dual filter mode; 1: Single filter mode (R/W)

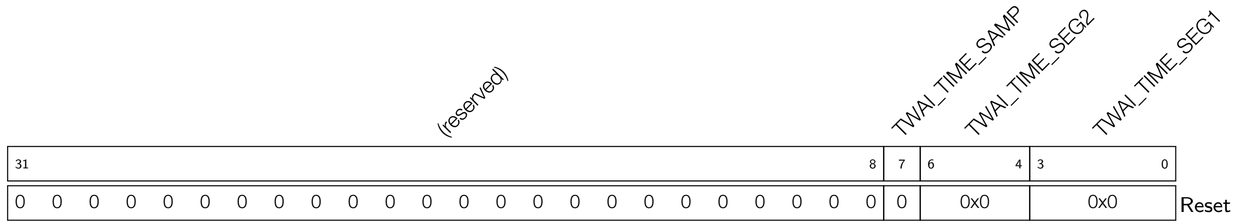
Register 29.2: TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Baud Rate Prescaler, determines the frequency dividing ratio. (RO | R/W)

TWAI_SYNC_JUMP_WIDTH Synchronization Jump Width (SJW), 1 ~ 14 Tq wide. (RO | R/W)

Register 29.3: TWAI_BUS_TIMING_1_REG (0x001C)

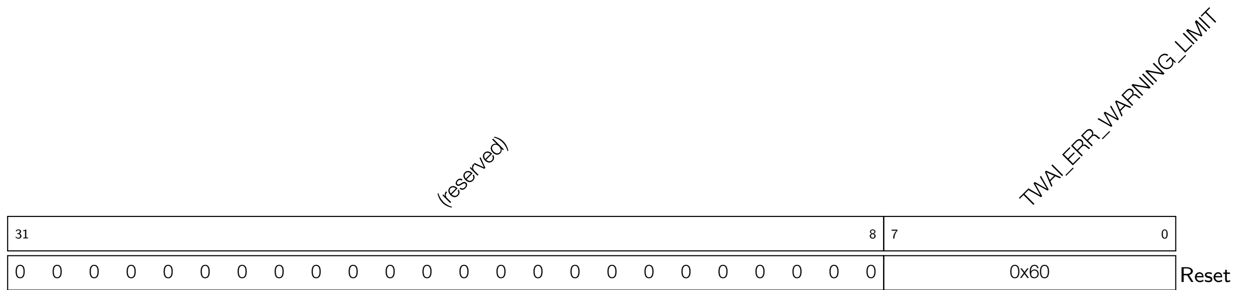


TWAI_TIME_SEG1 The width of PBS1. (RO | R/W)

TWAI_TIME_SEG2 The width of PBS2. (RO | R/W)

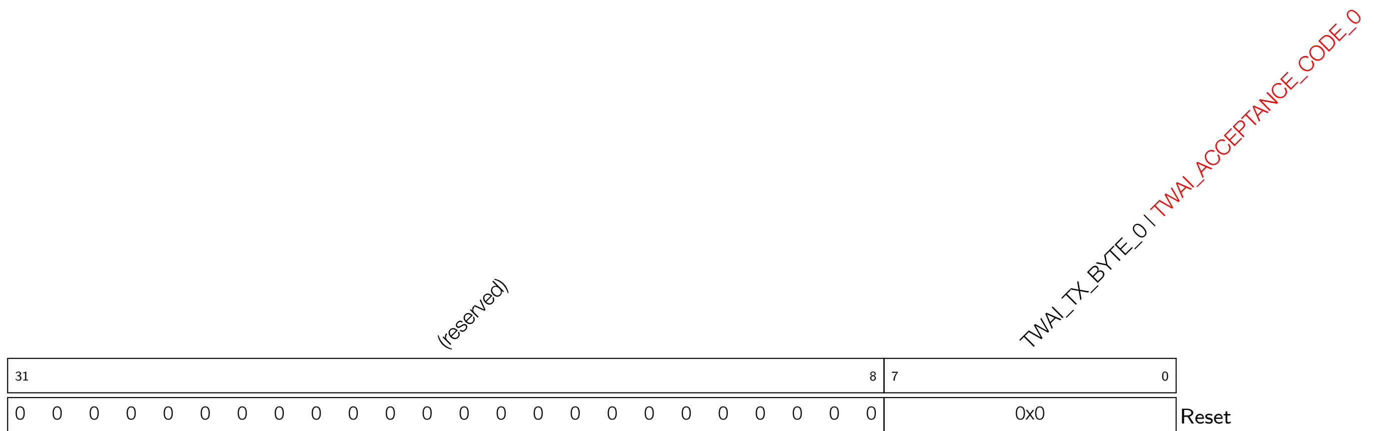
TWAI_TIME_SAMP The number of sample points. 0: the bus is sampled once; 1: the bus is sampled three times (RO | R/W)

Register 29.4: TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Error warning threshold. In the case when any of a error counter value exceeds the threshold, or all the error counter values are below the threshold, an error warning interrupt will be triggered (given the enable signal is valid). (RO | R/W)

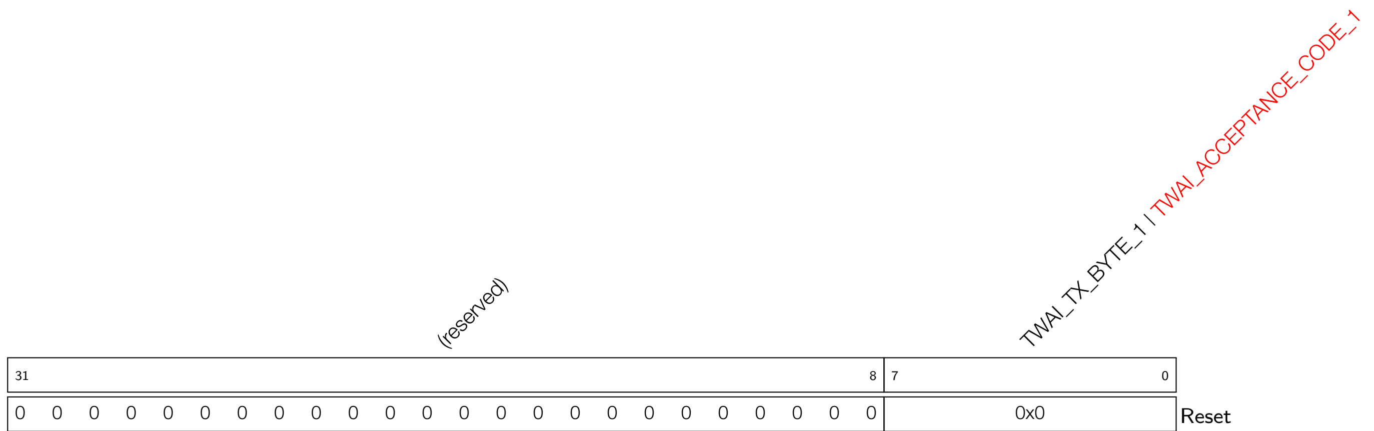
Register 29.5: TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 Stored the 0th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_0 Stored the 0th byte of the filter code under reset mode. (R/W)

Register 29.6: TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 Stored the 1st byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_1 Stored the 1st byte of the filter code under reset mode. (R/W)

Register 29.7: TWAI_DATA_2_REG (0x0048)

(reserved)		TWAI_TX_BYTE_2 | TWAI_ACCEPTANCE_CODE_2	
31	8	7	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0x0	Reset

TWAI_TX_BYTE_2 Stored the 2nd byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_2 Stored the 2nd byte of the filter code under reset mode. (R/W)

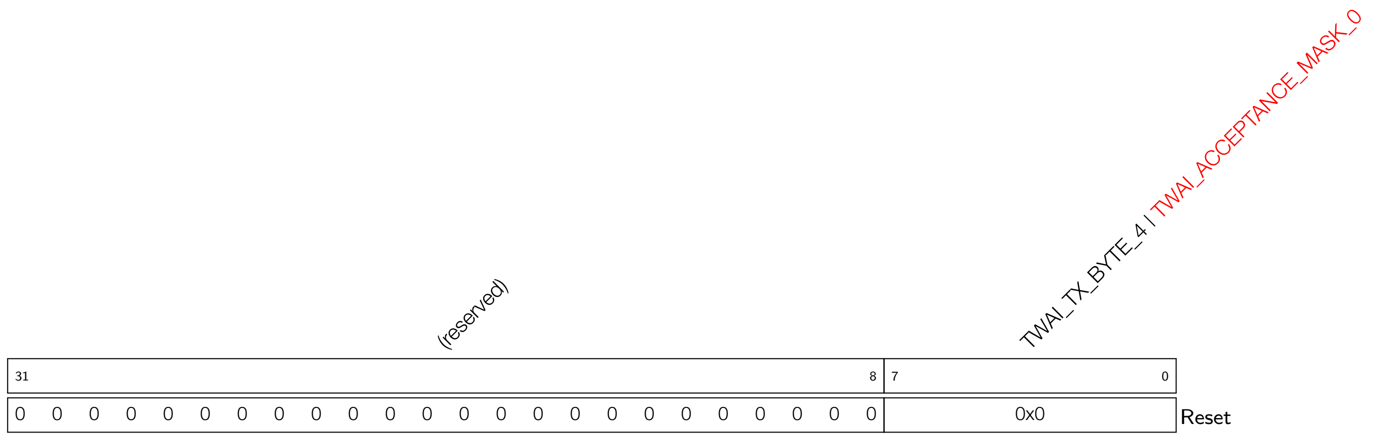
Register 29.8: TWAI_DATA_3_REG (0x004C)

(reserved)		TWAI_TX_BYTE_3 | TWAI_ACCEPTANCE_CODE_3	
31	8	7	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0x0	Reset

TWAI_TX_BYTE_3 Stored the 3rd byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_3 Stored the 3rd byte of the filter code under reset mode. (R/W)

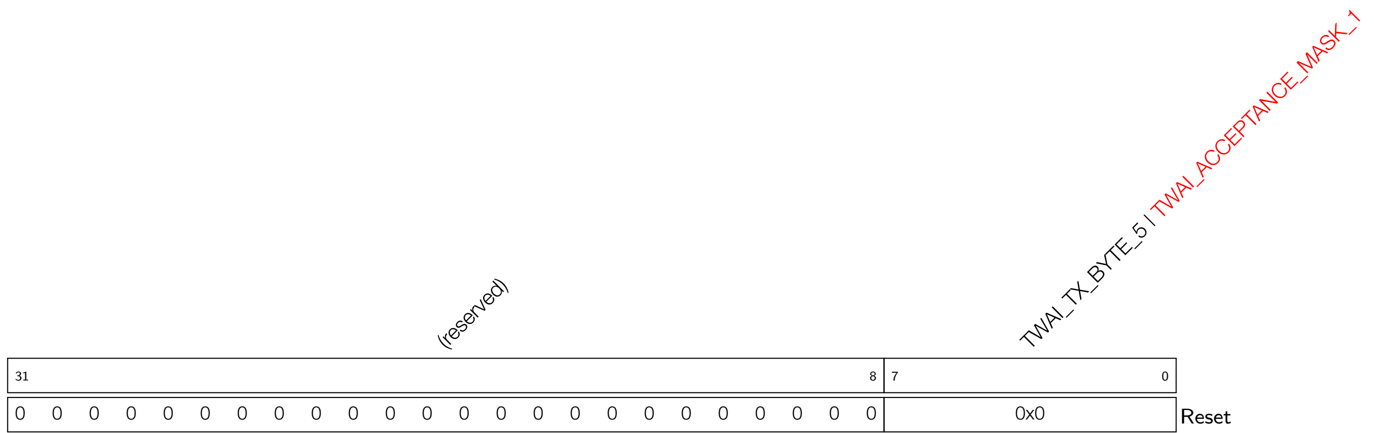
Register 29.9: TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 Stored the 4th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_0 Stored the 0th byte of the filter code under reset mode. (R/W)

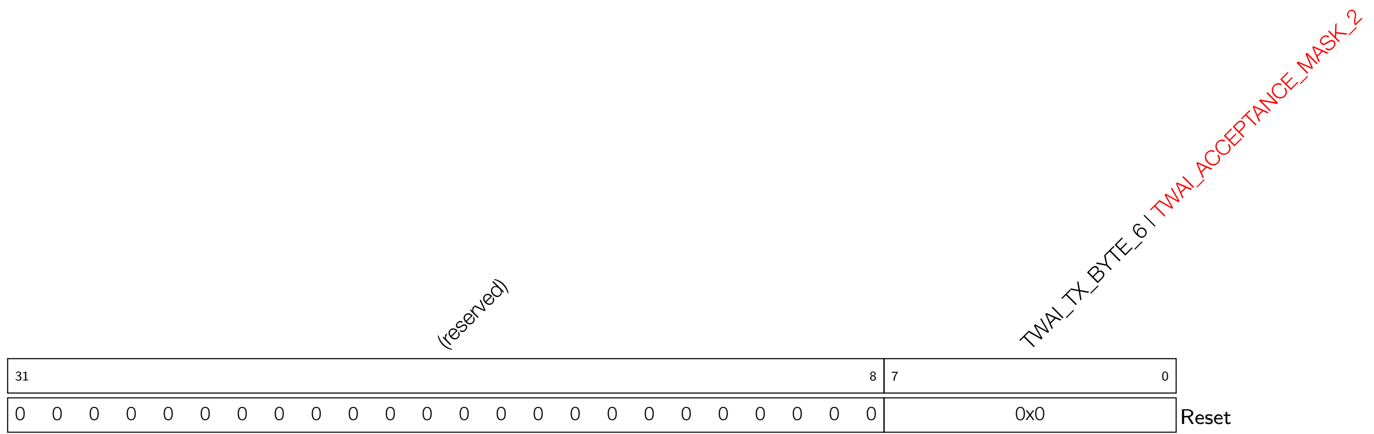
Register 29.10: TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 Stored the 5th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_1 Stored the 1st byte of the filter code under reset mode. (R/W)

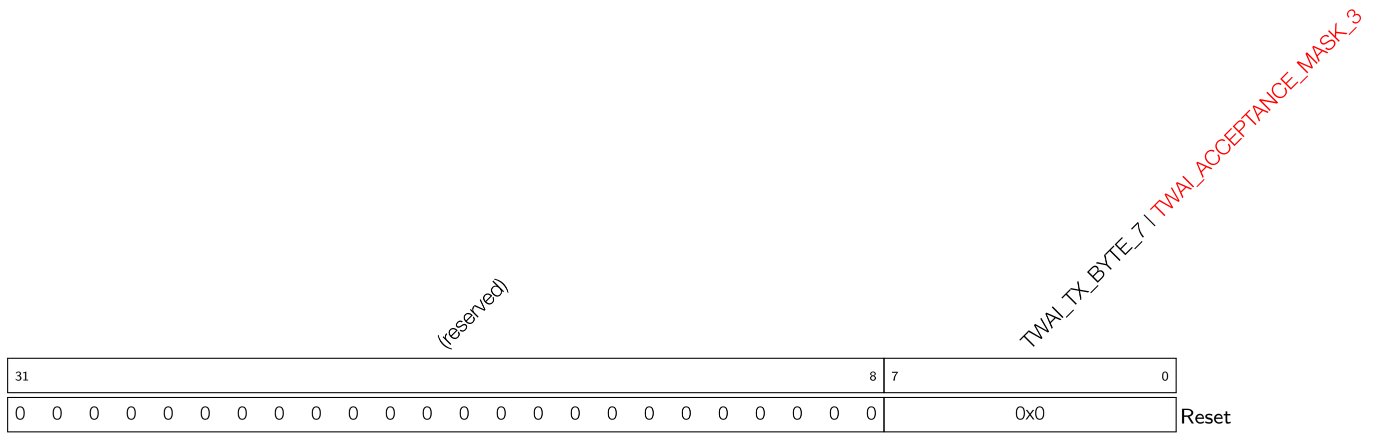
Register 29.11: TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 Stored the 6th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_2 Stored the 2nd byte of the filter code under reset mode. (R/W)

Register 29.12: TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 Stored the 7th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_3 Stored the 3rd byte of the filter code under reset mode. (R/W)

Register 29.13: TWAI_DATA_8_REG (0x0060)

(reserved)	TWAI_TX_BYTE_8	
31 ... 8	7 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_8 Stored the 8th byte information of the data to be transmitted under operating mode. (WO)

Register 29.14: TWAI_DATA_9_REG (0x0064)

(reserved)	TWAI_TX_BYTE_9	
31 ... 8	7 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

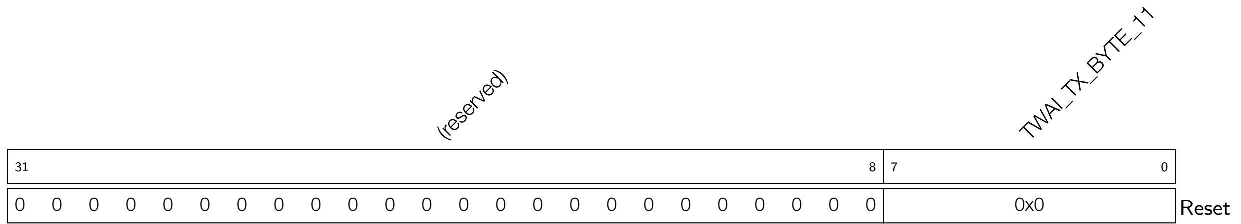
TWAI_TX_BYTE_9 Stored the 9th byte information of the data to be transmitted under operating mode. (WO)

Register 29.15: TWAI_DATA_10_REG (0x0068)

(reserved)	TWAI_TX_BYTE_10	
31 ... 8	7 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

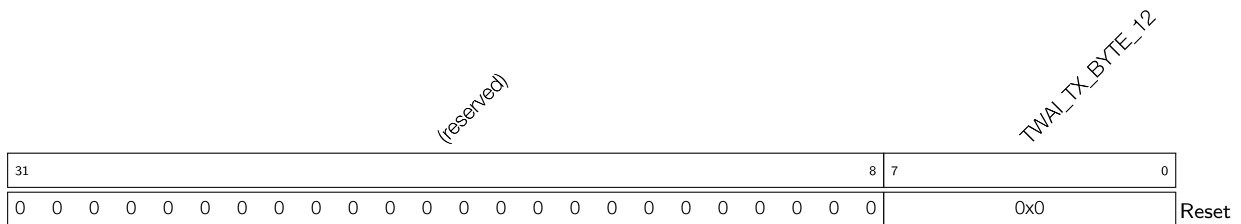
TWAI_TX_BYTE_10 Stored the 10th byte information of the data to be transmitted under operating mode. (WO)

Register 29.16: TWAI_DATA_11_REG (0x006C)



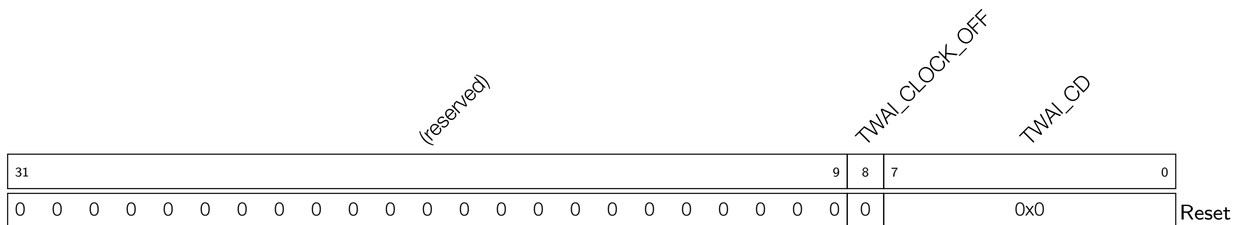
TWAI_TX_BYTE_11 Stored the 11th byte information of the data to be transmitted under operating mode. (WO)

Register 29.17: TWAI_DATA_12_REG (0x0070)



TWAI_TX_BYTE_12 Stored the 12th byte information of the data to be transmitted under operating mode. (WO)

Register 29.18: TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD These bits are used to configure frequency dividing coefficients of the external CLKOUT pin. (R/W)

TWAI_CLOCK_OFF This bit can be configured under reset mode. 1: Disable the external CLKOUT pin; 0: Enable the external CLKOUT pin (RO | R/W)

Register 29.19: TWAI_CMD_REG (0x0004)

(reserved)	TWAI_SELF_RX_REQ	TWAI_CLR_OVERRUN	TWAI_RELEASE_BUF	TWAI_ABORT_TX	TWAI_TX_REQ	
31 ... 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	Reset

TWAI_TX_REQ Set the bit to 1 to allow the driving nodes start transmission. (WO)

TWAI_ABORT_TX Set the bit to 1 to cancel a pending transmission request. (WO)

TWAI_RELEASE_BUF Set the bit to 1 to release the RX buffer. (WO)

TWAI_CLR_OVERRUN Set the bit to 1 to clear the data overrun status bit. (WO)

TWAI_SELF_RX_REQ Self reception request command. Set the bit to 1 to allow a message be transmitted and received simultaneously. (WO)

Register 29.20: TWAI_STATUS_REG (0x0008)

(reserved)	TWAI_MISS_ST	TWAI_BUS_OFF_ST	TWAI_ERR_ST	TWAI_TX_ST	TWAI_RX_ST	TWAI_TX_COMPLETE	TWAI_TX_BUF_ST	TWAI_OVERRUN_ST	TWAI_RX_BUF_ST	
31 ... 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	1	1	0	0	Reset

TWAI_RX_BUF_ST 1: The data in the RX buffer is not empty, with at least one received data packet. (RO)

TWAI_OVERRUN_ST 1: The RX FIFO is full and data overrun has occurred. (RO)

TWAI_TX_BUF_ST 1: The TX buffer is empty, the CPU may write a message into it. (RO)

TWAI_TX_COMPLETE 1: The TWAI controller has successfully received a packet from the bus. (RO)

TWAI_RX_ST 1: The TWAI Controller is receiving a message from the bus. (RO)

TWAI_TX_ST 1: The TWAI Controller is transmitting a message to the bus. (RO)

TWAI_ERR_ST 1: At least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

TWAI_BUS_OFF_ST 1: In bus-off status, the TWAI Controller is no longer involved in bus activities. (RO)

TWAI_MISS_ST This bit reflects whether the data packet in the RX FIFO is complete. 1: The current packet is missing; 0: The current packet is complete (RO)

Register 29.21: TWAI_ARB LOST CAP_REG (0x002C)

(reserved) 31..5	TWAI_ARB_LOST_CAP 4..0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_ARB_LOST_CAP This register contains information about the bit position of lost arbitration. (RO)

Register 29.22: TWAI_ERR_CODE_CAP_REG (0x0030)

(reserved) 31..8	TWAI_ECC_TYPE 7..6	TWAI_ECC_DIRECTION 5	TWAI_ECC_SEGMENT 4..0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0	0x0	Reset

TWAI_ECC_SEGMENT This register contains information about the location of errors, see Table 181 for details. (RO)

TWAI_ECC_DIRECTION This register contains information about transmission direction of the node when error occurs. 1: Error occurs when receiving a message; 0: Error occurs when transmitting a message (RO)

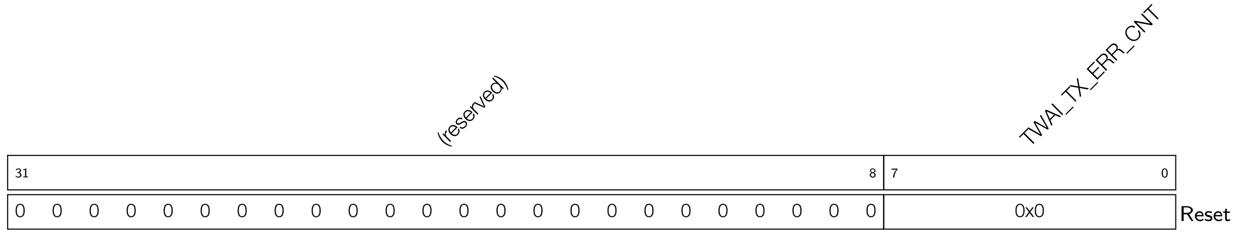
TWAI_ECC_TYPE This register contains information about error types: 00: bit error; 01: form error; 10: stuff error; 11: other type of error (RO)

Register 29.23: TWAI_RX_ERR_CNT_REG (0x0038)

(reserved) 31..8	TWAI_RX_ERR_CNT 7..0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

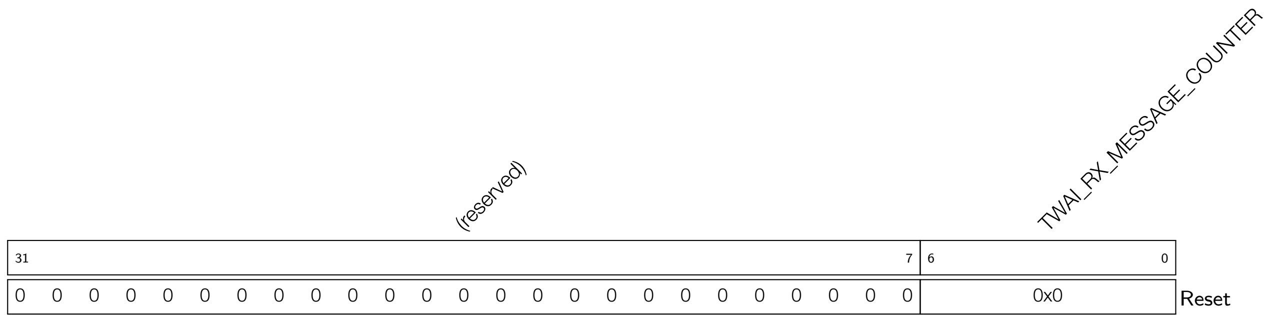
TWAI_RX_ERR_CNT The RX error counter register, reflects value changes under reception status. (RO | R/W)

Register 29.24: TWAI_TX_ERR_CNT_REG (0x003C)



TWAI_TX_ERR_CNT The TX error counter register, reflects value changes under transmission status. (RO | R/W)

Register 29.25: TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER This register reflects the number of messages available within the RX FIFO. (RO)

Register 29.26: TWAI_INT_RAW_REG (0x000C)

31 … 8	7	6	5	4	3	2	1	0	
(reserved)	TWAI_BUS_ERR_INT_ST	TWAI_ARB_LOST_INT_ST	TWAI_ERR_PASSIVE_INT_ST	(reserved)	TWAI_OVERRUN_INT_ST	TWAI_ERR_WARN_INT_ST	TWAI_TX_INT_ST	TWAI_RX_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

TWAI_RX_INT_ST Receive interrupt. If this bit is set to 1, it indicates there are messages to be handled in the RX FIFO. (RO)

TWAI_TX_INT_ST Transmit interrupt. If this bit is set to 1, it indicates the message transmitting mission is finished and a new transmission is able to execute. (RO)

TWAI_ERR_WARN_INT_ST Error warning interrupt. If this bit is set to 1, it indicates the error status signal and the bus-off status signal of Status register have changed (e.g., switched from 0 to 1 or from 1 to 0). (RO)

TWAI_OVERRUN_INT_ST Data overrun interrupt. If this bit is set to 1, it indicates a data overrun interrupt is generated in the RX FIFO. (RO)

TWAI_ERR_PASSIVE_INT_ST Error passive interrupt. If this bit is set to 1, it indicates the TWAI Controller is switched between error active status and error passive status due to the change of error counters. (RO)

TWAI_ARB_LOST_INT_ST Arbitration lost interrupt. If this bit is set to 1, it indicates an arbitration lost interrupt is generated. (RO)

TWAI_BUS_ERR_INT_ST Error interrupt. If this bit is set to 1, it indicates an error is detected on the bus. (RO)

Register 29.27: TWAI_INT ENA_REG (0x0010)

31–8	7	6	5	4	3	2	1	0	
(reserved)	TWAI_BUS_ERR_INT_ENA	TWAI_ARB_LOST_INT_ENA	TWAI_ERR_PASSIVE_INT_ENA	(reserved)	TWAI_OVERRUN_INT_ENA	TWAI_ERR_WARN_INT_ENA	TWAI_TX_INT_ENA	TWAI_RX_INT_ENA	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

TWAI_RX_INT_ENA Set this bit to 1 to enable receive interrupt. (R/W)

TWAI_TX_INT_ENA Set this bit to 1 to enable transmit interrupt. (R/W)

TWAI_ERR_WARN_INT_ENA Set this bit to 1 to enable error warning interrupt. (R/W)

TWAI_OVERRUN_INT_ENA Set this bit to 1 to enable data overrun interrupt. (R/W)

TWAI_ERR_PASSIVE_INT_ENA Set this bit to 1 to enable error passive interrupt. (R/W)

TWAI_ARB_LOST_INT_ENA Set this bit to 1 to enable arbitration lost interrupt. (R/W)

TWAI_BUS_ERR_INT_ENA Set this bit to 1 to enable error interrupt. (R/W)

30. LED PWM Controller (LEDC)

30.1 Overview

The LED PWM controller is primarily designed to control LED devices, as well as generate PWM signals. It has 14-bit timers and waveform generators.

30.2 Features

The LED PWM controller has the following features:

- Four independent timers that support division by fractions
- Eight independent waveform generators able to produce eight PWM signals
- Fading duty cycle of PWM signals without interference from any processors. An interrupt can be generated after the fade has completed
- Adjustable phase of PWM signal output
- PWM signal output in low-power mode

For the convenience of description, in the following sections the eight waveform generators are collectively referred to as PWM*n*, and the four timers are collectively referred to as Timer*x*.

30.3 Functional Description

30.3.1 Architecture

Figure 30-1 shows the architecture of the LED PWM controller. Figure 30-2 illustrates a PWM generator with its selected timer and a counter.

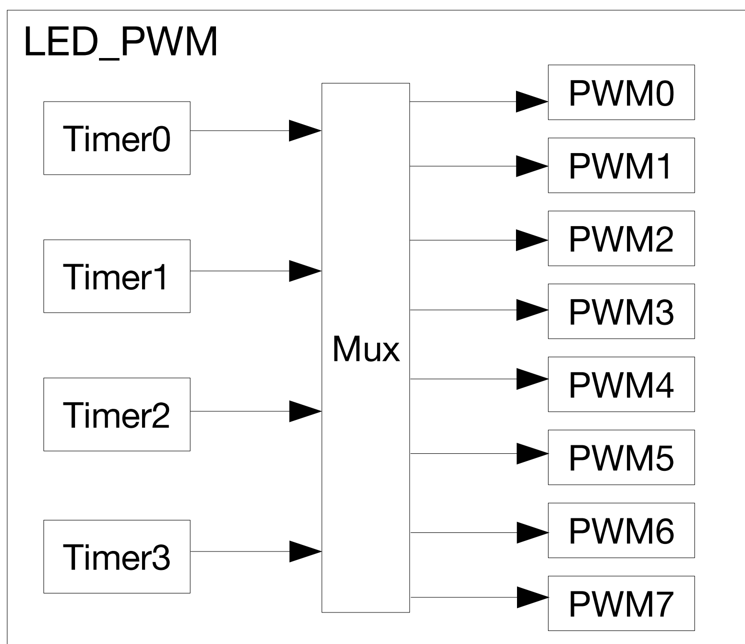


Figure 30-1. LED_PWM Architecture

30.3.2 Timers

The clock of the LED PWM controller, LEDC_PWM_CLK, has three clock sources: APB_CLK, RTC8M_CLK and XTAL_CLK, selected by configuring LEDC_APB_CLK_SEL[1:0]. The clock of each LED PWM timer, LEDC_CLK*x*,

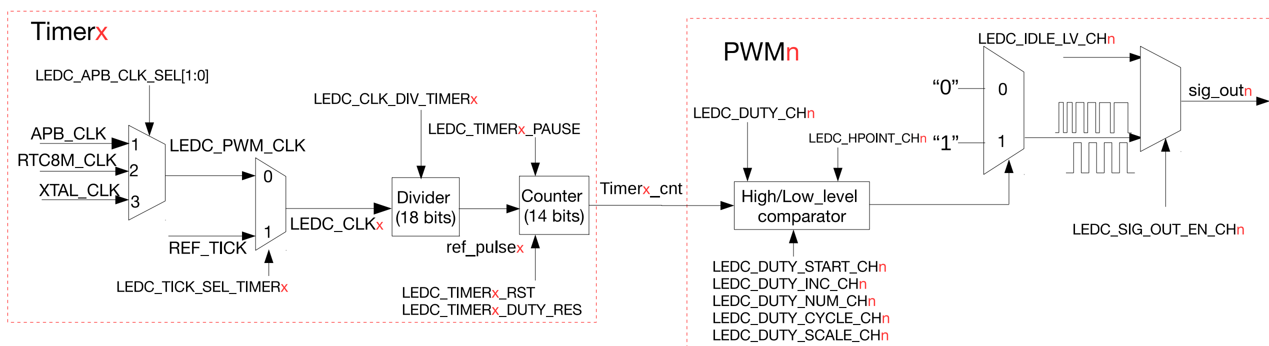


Figure 30-2. LED_PWM generator Diagram

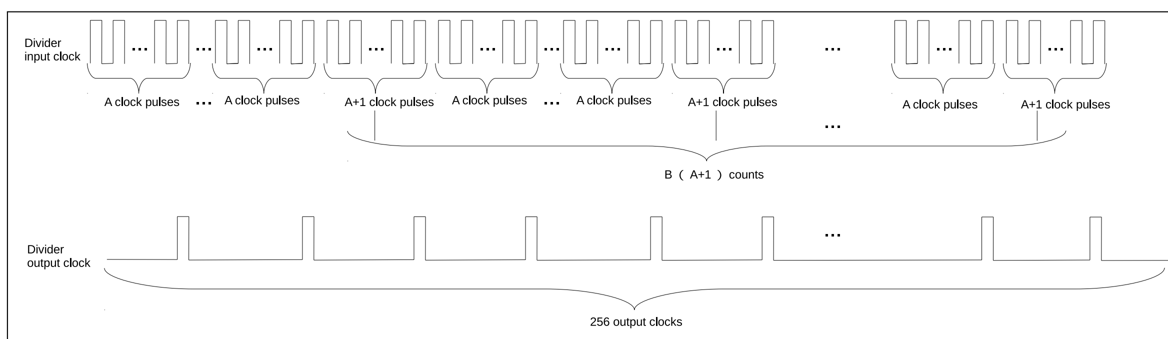


Figure 30-3. LED_PWM Divider

has two clock sources: LEDC_PWM_CLK or REF_TICK. When REF_TICK is used as the clock source of a timer, LEDC_APB_CLK_SEL[1:0] should be set 1 and the cycle of REF_TICK should be an integral multiple of APB_CLK cycles. Otherwise this clock will be not accurate. For more information on the clock sources, please see Chapter Reset and Clock.

The output clock derived from LEDC_CLK*x* is used as the base clock for the counter. The divider's divisor is configured by LEDC_CLK_DIV_TIMER*x*. It is a fixed-point number: the highest 10 bits is the integer part represented as A, while the lowest eight bits is the fractional part represented as B. This divisor LEDC_CLK_DIV*x* is calculated as:

$$LEDC_CLK_DIVx = A + \frac{B}{256}$$

When the fractional part B is not 0, the input and output clock of the divider is shown as in figure 30-3. Among the 256 output clocks, B of them are divided by (A+1), whereas the remaining (256-B) are divided by A. Output clocks divided by (A+1) are evenly distributed in the total 256 output clocks.

The LED PWM controller has a 14-bit counter that counts up to $2^{LEDC_TIMERx_DUTY_RES} - 1$. If the counting value reaches $2^{LEDC_TIMERx_DUTY_RES} - 1$, the counter will overflow and restart counting from 0. The counting value can be reset, suspended or read by software. A LEDC_TIMER*x*_OVF_INT interrupt can be generated every time when the counter overflows or when it overflows for (LEDC_OVF_NUM_CH*n* + 1) times. The interrupt configuration is as follows:

1. Set LEDC_OVF_CNT_EN_CH*n*
2. Configure LEDC_OVF_NUM_CH*n* with the times of overflow minus 1
3. Set LEDC_OVF_CNT_CH*n*_INT_ENA

4. Set LEDC_TIMERx_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CHn_INT interrupt

The frequency of a PWM generator output signal, sig_outn, depends on both the divisor of the divider, as well as the range of the counter:

$$f_{\text{sig_out}n} = \frac{f_{\text{LEDC_CLK}x}}{\text{LEDC_CLK_DIV}x \cdot 2^{\text{LEDC_TIMER}x\text{_DUTY_RES}}}$$

To change the divisor and times of overflow, you should configure LEDC_CLK_DIV_TIMERx and LEDC_TIMERx_DUTY_RES respectively, and then set LEDC_TIMERx_PARA_UP; otherwise this change is not valid. The newly configured value is updated upon next overflow of the counter.

30.3.3 PWM Generators

As shown in figure 30-2, each PWM generator has a high/low level comparator and two selectors. A PWM generator takes the 14-bit counting value of the selected timer, compares it to values of the comparator Hpointn and Lpointn, and therefore control the level of PWM signals.

- If Timerx_cnt == Hpointn, sig_outn is 1.
- If Timerx_cnt == Lpointn, sig_outn is 0.

Hpointn is updated by LEDC_HPOINT_CHn when the counter overflows. The initial value of Lpointn is the sum of LEDC_DUTY_CHn[18..4] and LEDC_HPOINT_CHn when the counter overflows. By configuring these two fields, the relative phase and the duty cycle of the PWM output can be set.

Figure 30-4 illustrates PWM's waveform when the duty cycle is fixed.

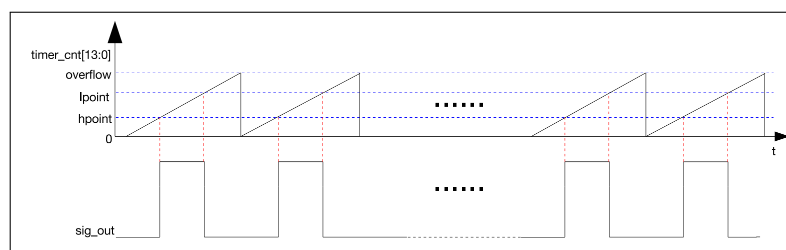


Figure 30-4. LED_PWM Output Signal Diagram

LEDC_DUTY_CHn is a fixed-point register with four fractional bits. LEDC_DUTY_CHn[18..4] is the integral part used directly for PWM calculation. LEDC_DUTY_CHn[3..0] is the fractional part used to dither the output. If LEDC_DUTY_CHn[3..0] is non-zero, then among every 16 cycles of sig_outn, LEDC_DUTY_CHn[3..0] have PWM pulses with width one timer cycle longer than that of (16-LEDC_DUTY_CHn[3..0]). This feature effectively increases the resolution of the PWM generator to 18 bits.

30.3.4 Duty Cycle Fading

The PWM generators is able to fade the duty cycle, that is to gradually change the duty cycle from one value to another. This is achieved by configuring LEDC_DUTY_CHn, LEDC_DUTY_START_CHn, LEDC_DUTY_INC_CHn, LEDC_DUTY_NUM_CHn and LEDC_DUTY_SCALE_CHn.

LEDC_DUTY_START_CHn is used to update the value of Lpointn. When this bit is set and the counter overflows, Lpointn increments or decrements automatically, depending on whether the bit LEDC_DUTY_INC_CHn is set or cleared.

The duty cycle of sig_out*n* changes every LEDC_DUTY_CYCLE_CH*n* PWM pulse cycles by adding or subtracting the value of LEDC_DUTY_SCALE_CH*n*.

Figure 30-5 is a diagram of fading duty cycle. Upon reaching LEDC_DUTY_NUM_CH*n*, the fade stops and a LEDC_DUTY_CHNG_END_CH*n*_INT interrupt is generated. When configured like this, the duty cycle of sig_out*n* increases by LEDC_DUTY_SCALE_CH*n* every LEDC_DUTY_CYCLE_CH*n* PWM pulse cycles.

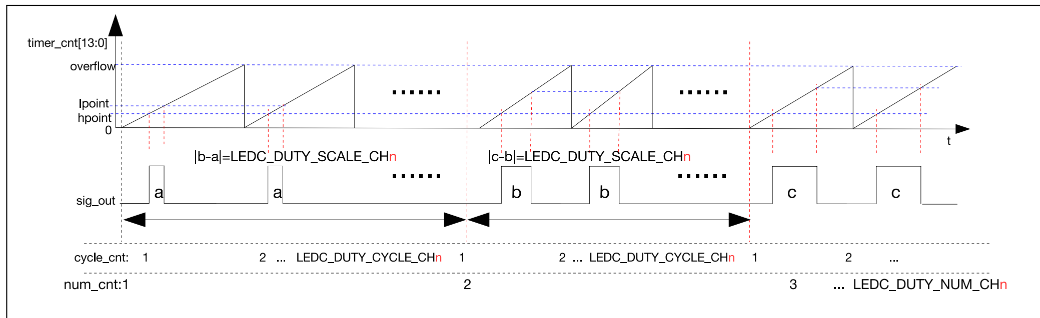


Figure 30-5. Output Signal Diagram of Fading Duty Cycle

LEDC_SIG_OUT_EN_CH*n* used to enable PWM waveform output. When LEDC_SIG_OUT_EN_CH*n* is 0, the level of sig_out*n* is constant as specified in LEDC_IDLE_LV_CH*n*.

If LEDC_HPOINT_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_SIG_OUT_EN_CH*n*, LEDC_TIMER_SEL_CH*n*, LEDC_DUTY_NUM_CH*n*, LEDC_DUTY_CYCLE_CH*n*, LEDC_DUTY_SCALE_CH*n*, LEDC_DUTY_INC_CH*n* and LEDC_OVF_CNT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration.

30.3.5 Interrupts

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for (LEDC_OVF_NUM_CH*n* + 1) times and register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on a LED PWM generator has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when a LED PWM timer has reached its maximum counter value.

30.4 Base Address

Users can access the LED PWM controller with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3 *System and Memory*.

Table 186: LED PWM Controller Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F419000
PeriBUS2	0x60019000

30.5 Register Summary

The addresses in the following table are relative to the LED PWM controller base addresses provided in Section 30.4.

Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	R/W
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	R/W
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	R/W
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	R/W
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W
LEDC_CH6_CONF0_REG	Configuration register 0 for channel 6	0x0078	varies
LEDC_CH6_CONF1_REG	Configuration register 1 for channel 6	0x0084	R/W
LEDC_CH7_CONF0_REG	Configuration register 0 for channel 7	0x008C	varies
LEDC_CH7_CONF1_REG	Configuration register 1 for channel 7	0x0098	R/W
LEDC_CONF_REG	Global ledc configuration register	0x00D0	R/W
Hpoint Register			
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
LEDC_CH6_HPOINT_REG	High point register for channel 6	0x007C	R/W
LEDC_CH7_HPOINT_REG	High point register for channel 7	0x0090	R/W
Duty Cycle Register			
LEDC_CH0_DUTY_REG	Initial duty cycle for channel 0	0x0008	R/W
LEDC_CH0_DUTY_R_REG	Current duty cycle for channel 0	0x0010	RO
LEDC_CH1_DUTY_REG	Initial duty cycle for channel 1	0x001C	R/W
LEDC_CH1_DUTY_R_REG	Current duty cycle for channel 1	0x0024	RO
LEDC_CH2_DUTY_REG	Initial duty cycle for channel 2	0x0030	R/W
LEDC_CH2_DUTY_R_REG	Current duty cycle for channel 2	0x0038	RO
LEDC_CH3_DUTY_REG	Initial duty cycle for channel 3	0x0044	R/W
LEDC_CH3_DUTY_R_REG	Current duty cycle for channel 3	0x004C	RO
LEDC_CH4_DUTY_REG	Initial duty cycle for channel 4	0x0058	R/W
LEDC_CH4_DUTY_R_REG	Current duty cycle for channel 4	0x0060	RO
LEDC_CH5_DUTY_REG	Initial duty cycle for channel 5	0x006C	R/W
LEDC_CH5_DUTY_R_REG	Current duty cycle for channel 5	0x0074	RO

Name	Description	Address	Access
LEDC_CH6_DUTY_REG	Initial duty cycle for channel 6	0x0080	R/W
LEDC_CH6_DUTY_R_REG	Current duty cycle for channel 6	0x0088	RO
LEDC_CH7_DUTY_REG	Initial duty cycle for channel 7	0x0094	R/W
LEDC_CH7_DUTY_R_REG	Current duty cycle for channel 7	0x009C	RO
Timer Register			
LEDC_TIMER0_CONF_REG	Timer 0 configuration	0x00A0	varies
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value	0x00A4	RO
LEDC_TIMER1_CONF_REG	Timer 1 configuration	0x00A8	varies
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value	0x00AC	RO
LEDC_TIMER2_CONF_REG	Timer 2 configuration	0x00B0	varies
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value	0x00B4	RO
LEDC_TIMER3_CONF_REG	Timer 3 configuration	0x00B8	varies
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value	0x00BC	RO
Interrupt Register			
LEDC_INT_RAW_REG	Raw interrupt status	0x00C0	RO
LEDC_INT_ST_REG	Masked interrupt status	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable bits	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear bits	0x00CC	WO
Version Register			
LEDC_DATE_REG	Version control register	0x00FC	R/W

30.6 Registers

Register 30.1: LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**

(reserved)	LEDC_OVF_CNT_RESET_ST_CH*n*	LEDC_OVF_CNT_RESET_CH*n*	LEDC_OVF_CNT_EN_CH*n*	LEDC_OVF_NUM_CH*n*	LEDC_PARA_UP_CH*n*	LEDC_IDLE_LV_CH*n*	LEDC_SIG_OUT_EN_CH*n*	LEDC_TIMER_SEL_CH*n*	
31 ... 18	17	16	15	14 ... 5	4	3	2	1 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0x0	0	0	0	0x0	Reset

LEDC_TIMER_SEL_CH*n* This field is used to select one of timers for channel *n*.

0: select timer0

1: select timer1

2: select timer2

3: select timer3 (R/W)

LEDC_SIG_OUT_EN_CH*n* Set this bit to enable signal output on channel *n*. (R/W)

LEDC_IDLE_LV_CH*n* This bit is used to control the output value when channel *n* is inactive. (R/W)

LEDC_PARA_UP_CH*n* This bit is used to update register LEDC_CH*n*_HPOINT and LEDC_CH*n*_DUTY for channel *n*. (WO)

LEDC_OVF_NUM_CH*n* This register is used to configure the maximum times of overflow minus 1.

The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* This bit is used to enable the ovf_cnt of channel *n*. (R/W)

LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the ovf_cnt of channel *n*. (WO)

LEDC_OVF_CNT_RESET_ST_CH*n* This is the status bit of LEDC_OVF_CNT_RESET_CH*n*. (RO)

Register 30.2: LEDC_CHn_CONF1_REG (n: 0-7) (0x000C+0x14*n)

31	30	29 … 20	19 … 10	9 … 0	
LEDC_DUTY_START_CHn	LEDC_DUTY_INC_CHn	LEDC_DUTY_NUM_CHn	LEDC_DUTY_CYCLE_CHn	LEDC_DUTY_SCALE_CHn	
0	1	0x0	0x0	0x0	Reset

LEDC_DUTY_SCALE_CHn This register is used to configure the changing step scale of duty on channel n. (R/W)

LEDC_DUTY_CYCLE_CHn The duty will change every LEDC_DUTY_CYCLE_CHn on channel n. (R/W)

LEDC_DUTY_NUM_CHn This register is used to control the number of times the duty cycle will be changed. (R/W)

LEDC_DUTY_INC_CHn This register is used to increase or decrease the duty of output signal on channel n. 1: Increase;0: Decrease. (R/W)

LEDC_DUTY_START_CHn Other configured fields in LEDC_CHn_CONF1_REG will start to take effect when this bit is set to 1. (R/W)

Register 30.3: LEDC_CONF_REG (0x00D0)

31	30 … 2	1 … 0	
LEDC_CLK_EN	(reserved)	LEDC_APB_CLK_SEL	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

LEDC_APB_CLK_SEL This bit is used to select clock source for the 4 timers .

2'd1: APB_CLK 2'd2: RTC8M_CLK 2'd3: XTAL_CLK (R/W)

LEDC_CLK_EN This bit is used to control clock.

1'b1: Force clock on for register. 1'h0: Support clock only when application writes registers. (R/W)

Register 30.4: LEDC_CHn_HPOINT_REG (n: 0-7) (0x0004+0x14*n)

(reserved)	LEDC_HPOINT_CHn	
31 ... 14	13 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

LEDC_HPOINT_CHn The output value changes to high when the selected timers has reached the value specified by this register. (R/W)

Register 30.5: LEDC_CHn_DUTY_REG (n: 0-7) (0x0008+0x14*n)

(reserved)	LEDC_DUTY_CHn	
31 ... 19	18 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0x000	Reset

LEDC_DUTY_CHn This register is used to change the output duty by controlling the Lpoint.

The output value turns to low when the selected timers has reached the Lpoint. (R/W)

Register 30.6: LEDC_CHn_DUTY_R_REG (n: 0-7) (0x0010+0x14*n)

(reserved)	LEDC_DUTY_R_CHn	
31 ... 19	18 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0x000	Reset

LEDC_DUTY_R_CHn This register stores the current duty of output signal on channel n. (RO)

Register 30.7: LEDC_TIMERx_CONF_REG (x: 0-3) (0x00A0+0x8*x)

31–26	25	24	23	22	21–4	3–0	
(reserved)	LEDC_TIMERx_PARA_UP	LEDC_TICK_SEL_TIMERx	LEDC_TIMERx_RST	LEDC_TIMERx_PAUSE	LEDC_CLK_DIV_TIMERx	LEDC_TIMERx_DUTY_RES	
0 0 0 0 0 0	0	0	1	0	0x000	0x0	Reset

LEDC_TIMERx_DUTY_RES This register is used to control the range of the counter in timer x. (R/W)

LEDC_CLK_DIV_TIMERx This register is used to configure the divisor for the divider in timer x.

The least significant eight bits represent the fractional part. (R/W)

LEDC_TIMERx_PAUSE This bit is used to suspend the counter in timer x. (R/W)

LEDC_TIMERx_RST This bit is used to reset timer x. The counter will show 0 after reset. (R/W)

LEDC_TICK_SEL_TIMERx This bit is used to select clock for timer x. When this bit is set to 1 LEDC_APB_CLK_SEL[1:0] should be 1, otherwise the timer clock may be not accurate.

1'h0: LEDC_PWM_CLK; 1'h1: REF_TICK. (R/W)

LEDC_TIMERx_PARA_UP Set this bit to update LEDC_CLK_DIV_TIMERx and LEDC_TIMERx_DUTY_RES. (WO)

Register 30.8: LEDC_TIMERx_VALUE_REG (x: 0-3) (0x00A4+0x8*x)

31–14	13–0	
(reserved)	LEDC_TIMERx_CNT	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	Reset

LEDC_TIMERx_CNT This register stores the current counter value of timer x. (RO)

Register 30.9: LEDC_INT_RAW_REG (0x00C0)

31–20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	LEDC_OVF_CNT_CH7_INT_RAW	LEDC_OVF_CNT_CH6_INT_RAW	LEDC_OVF_CNT_CH5_INT_RAW	LEDC_OVF_CNT_CH4_INT_RAW	LEDC_OVF_CNT_CH3_INT_RAW	LEDC_OVF_CNT_CH2_INT_RAW	LEDC_OVF_CNT_CH1_INT_RAW	LEDC_OVF_CNT_CH0_INT_RAW	LEDC_DUTY_CHNG_END_CH7_INT_RAW	LEDC_DUTY_CHNG_END_CH6_INT_RAW	LEDC_DUTY_CHNG_END_CH5_INT_RAW	LEDC_DUTY_CHNG_END_CH4_INT_RAW	LEDC_DUTY_CHNG_END_CH3_INT_RAW	LEDC_DUTY_CHNG_END_CH2_INT_RAW	LEDC_DUTY_CHNG_END_CH1_INT_RAW	LEDC_DUTY_CHNG_END_CH0_INT_RAW	LEDC_TIMER3_OVF_INT_RAW	LEDC_TIMER2_OVF_INT_RAW	LEDC_TIMER1_OVF_INT_RAW	LEDC_TIMER0_OVF_INT_RAW	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_TIMERx_OVF_INT_RAW Triggered when the timerx has reached its maximum counter value. (RO)

LEDC_DUTY_CHNG_END_CHn_INT_RAW Interrupt raw bit for channel n. Triggered when the gradual change of duty has finished. (RO)

LEDC_OVF_CNT_CHn_INT_RAW Interrupt raw bit for channel n. Triggered when the ovf_cnt has reached the value specified by LEDC_OVF_NUM_CHn. (RO)

Register 30.10: LEDC_INT_ST_REG (0x00C4)

31–20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(reserved)	LEDC_OVF_CNT_CH7_INT_ST	LEDC_OVF_CNT_CH6_INT_ST	LEDC_OVF_CNT_CH5_INT_ST	LEDC_OVF_CNT_CH4_INT_ST	LEDC_OVF_CNT_CH3_INT_ST	LEDC_OVF_CNT_CH2_INT_ST	LEDC_OVF_CNT_CH1_INT_ST	LEDC_OVF_CNT_CH0_INT_ST	LEDC_DUTY_CHNG_END_CH7_INT_ST	LEDC_DUTY_CHNG_END_CH6_INT_ST	LEDC_DUTY_CHNG_END_CH5_INT_ST	LEDC_DUTY_CHNG_END_CH4_INT_ST	LEDC_DUTY_CHNG_END_CH3_INT_ST	LEDC_DUTY_CHNG_END_CH2_INT_ST	LEDC_DUTY_CHNG_END_CH1_INT_ST	LEDC_DUTY_CHNG_END_CH0_INT_ST	LEDC_TIMER3_OVF_INT_ST	LEDC_TIMER2_OVF_INT_ST	LEDC_TIMER1_OVF_INT_ST	LEDC_TIMER0_OVF_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_TIMERx_OVF_INT_ST This is the masked interrupt status bit for the LEDC_TIMERx_OVF_INT interrupt when LEDC_TIMERx_OVF_INT_ENA is set to 1. (RO)

LEDC_DUTY_CHNG_END_CHn_INT_ST This is the masked interrupt status bit for the LEDC_DUTY_CHNG_END_CHn_INT interrupt when LEDC_DUTY_CHNG_END_CHn_INT_ENAIS set to 1. (RO)

LEDC_OVF_CNT_CHn_INT_ST This is the masked interrupt status bit for the LEDC_OVF_CNT_CHn_INT interrupt when LEDC_OVF_CNT_CHn_INT_ENA is set to 1. (RO)

Register 30.11: LEDC_INT_ENA_REG (0x00C8)

Bits	Field	Reset
31–20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	LEDC_OVF_CNT_CH7_INT_ENA	0
18	LEDC_OVF_CNT_CH6_INT_ENA	0
17	LEDC_OVF_CNT_CH5_INT_ENA	0
16	LEDC_OVF_CNT_CH4_INT_ENA	0
15	LEDC_OVF_CNT_CH3_INT_ENA	0
14	LEDC_OVF_CNT_CH2_INT_ENA	0
13	LEDC_OVF_CNT_CH1_INT_ENA	0
12	LEDC_OVF_CNT_CH0_INT_ENA	0
11	LEDC_DUTY_CHNG_END_CH7_INT_ENA	0
10	LEDC_DUTY_CHNG_END_CH6_INT_ENA	0
9	LEDC_DUTY_CHNG_END_CH5_INT_ENA	0
8	LEDC_DUTY_CHNG_END_CH4_INT_ENA	0
7	LEDC_DUTY_CHNG_END_CH3_INT_ENA	0
6	LEDC_DUTY_CHNG_END_CH2_INT_ENA	0
5	LEDC_DUTY_CHNG_END_CH1_INT_ENA	0
4	LEDC_DUTY_CHNG_END_CH0_INT_ENA	0
3	LEDC_TIMER3_OVF_INT_ENA	0
2	LEDC_TIMER2_OVF_INT_ENA	0
1	LEDC_TIMER1_OVF_INT_ENA	0
0	LEDC_TIMER0_OVF_INT_ENA	0

LEDC_TIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_TIMER*x*_OVF_INT interrupt. (R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. (R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA The interrupt enable bit for the LEDC_OVF_CNT_CH*n*_INT interrupt. (R/W)

Register 30.12: LEDC_INT_CLR_REG (0x00CC)

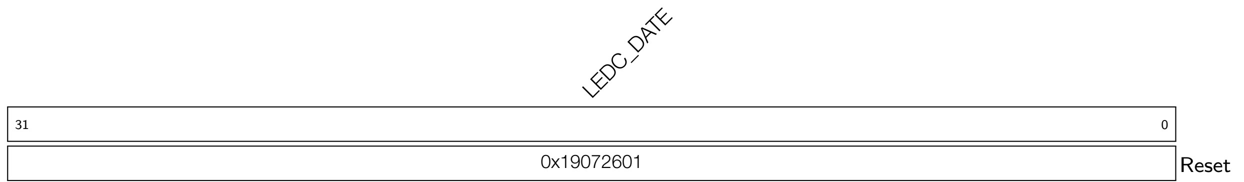
Bits	Field	Reset
31–20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	LEDC_OVF_CNT_CH7_INT_CLR	0
18	LEDC_OVF_CNT_CH6_INT_CLR	0
17	LEDC_OVF_CNT_CH5_INT_CLR	0
16	LEDC_OVF_CNT_CH4_INT_CLR	0
15	LEDC_OVF_CNT_CH3_INT_CLR	0
14	LEDC_OVF_CNT_CH2_INT_CLR	0
13	LEDC_OVF_CNT_CH1_INT_CLR	0
12	LEDC_OVF_CNT_CH0_INT_CLR	0
11	LEDC_DUTY_CHNG_END_CH7_INT_CLR	0
10	LEDC_DUTY_CHNG_END_CH6_INT_CLR	0
9	LEDC_DUTY_CHNG_END_CH5_INT_CLR	0
8	LEDC_DUTY_CHNG_END_CH4_INT_CLR	0
7	LEDC_DUTY_CHNG_END_CH3_INT_CLR	0
6	LEDC_DUTY_CHNG_END_CH2_INT_CLR	0
5	LEDC_DUTY_CHNG_END_CH1_INT_CLR	0
4	LEDC_DUTY_CHNG_END_CH0_INT_CLR	0
3	LEDC_TIMER3_OVF_INT_CLR	0
2	LEDC_TIMER2_OVF_INT_CLR	0
1	LEDC_TIMER1_OVF_INT_CLR	0
0	LEDC_TIMER0_OVF_INT_CLR	0

LEDC_TIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_TIMER*x*_OVF_INT interrupt. (WO)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. (WO)

LEDC_OVF_CNT_CH*n*_INT_CLR Set this bit to clear the LEDC_OVF_CNT_CH*n*_INT interrupt. (WO)

Register 30.13: LEDC_DATE_REG (0x00FC)



LEDC_DATE This is the version control register. (R/W)

31. Remote Control Peripheral (RMT)

31.1 Introduction

The RMT (Remote Control) module is designed to send/receive infrared remote control signals that support for a variety of remote control protocols. The RMT module converts pulse codes stored in the module's built-in RAM into output signals, or converts input signals into pulse codes and stores them back in RAM. In addition, the RMT module optionally modulates its output signals with a carrier wave, or optionally filters its input signals.

The RMT module has four channels, numbered from zero to three. Each channel has the same functionality controlled by dedicated set of registers and is able to independently transmit or receive data. Registers in each channel are indicated by *n* which is used as a placeholder for the channel number.

31.2 Functional Description

31.2.1 RMT Architecture

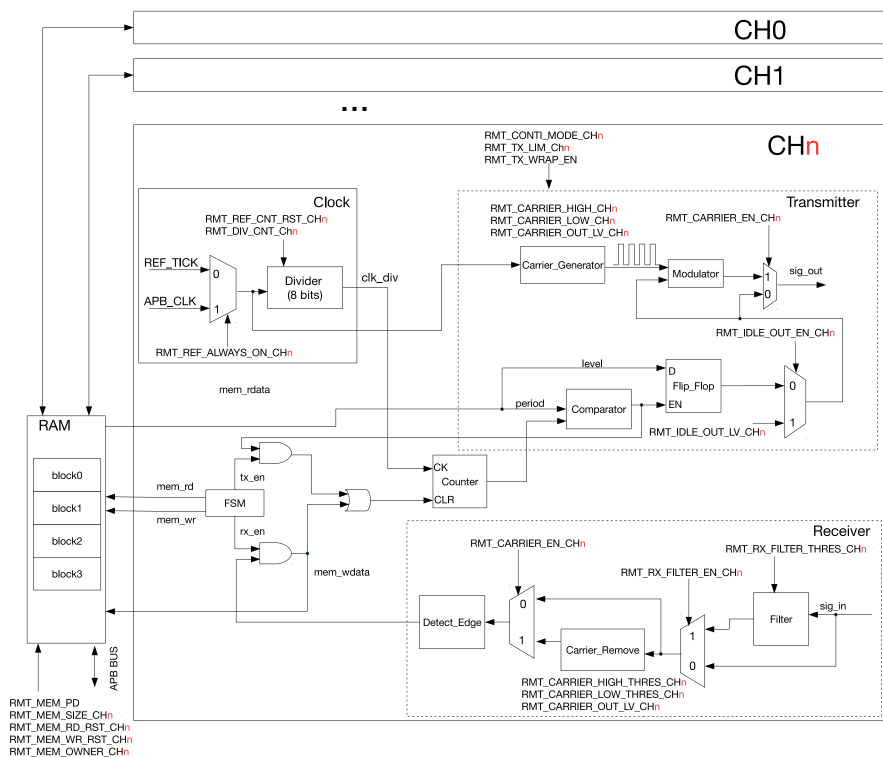


Figure 31-1. RMT Architecture

The RMT module contains four independent channels. Each channel has a clock divider, a counter, a transmitter and a receiver. As for the transmitter and the receiver of a single channel, only one of them can be active. The four channels share a 256 x 32-bit RAM.

31.2.2 RMT RAM

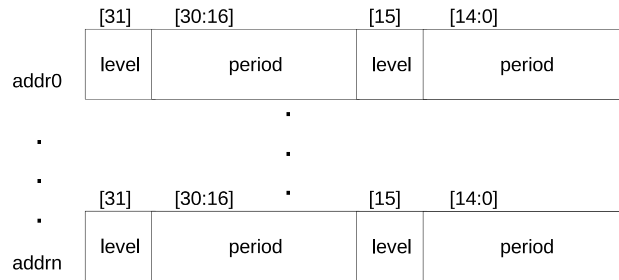


Figure 31-2. Format of Pulse Code in RAM

The format of pulse code in RAM is shown in Figure 31-2. Each pulse code contains a 16-bit entry with two fields, level and period. Level (0 or 1) indicates a high-/low-level value was received or is going to be sent, while "period" points out the clock cycles (see Figure 31-1 clk_div) for which the level lasts. A zero period is interpreted as a transmission end-marker.

The RAM is divided into four 64 x 32-bit blocks. By default, each channel uses one block (block zero for channel zero, block one for channel one, and so on). Usually, only one block of 64 x 32-bit worth of data can be sent or received in channel *n*. If the data size is larger than this block size, users can configure the channel to enable wrap mode or to use more blocks by setting RMT_MEM_SIZE_CH*n*. Setting RMT_MEM_SIZE_CH*n* > 1 will prompt channel *n* to use the memory of subsequent channels, block (*n*) ~ block (*n* + RMT_MEM_SIZE_CH*n* -1). If so, the subsequent channels *n* +1 ~ *n* + RMT_MEM_SIZE_CH*n* - 1 cannot be used once their RAM blocks are occupied. Note that the RAM used by each channel is mapped from low address to high address. In such mode, channel 0 is able to use the RAM blocks for channels 1, 2 and 3 by setting RMT_MEM_SIZE_CH*n*, but channel 3 cannot use the blocks for channels 0, 1, or 2.

The RMT RAM can be accessed via APB bus, or read by the transmitter and written by the receiver. To protect a receiver from overwriting the blocks a transmitter is about to transmit, RMT_MEM_OWNER_CH*n* can be configured to designate the block's owner, be it a transmitter or receiver. This way, if this ownership is violated, an RMT_MEM_OWNER_ERR_CH*n* flag will be generated.

When the RMT module is inactive, the RAM can be put into low-power mode by setting RMT_MEM_FORCE_PD.

31.2.3 Clock

The drive clock of a divider is generated by taking either the APB_CLK or REF_TICK according to the state of RMT_REF_ALWAYS_ON_CH*n*. (For more information on clock sources, please see Chapter Reset and Clock). Divider value is normally equal to the value of RMT_DIV_CNT_CH*n*, except value 0 that represents divider 256. The clock divider can be reset to zero by clearing RMT_REF_CNT_RST_CH*n*. The clock generated from the divider can be used by the clock counter (see Figure 31-2).

31.2.4 Transmitter

When RMT_TX_START_CH*n* is set to 1, the transmitter of channel *n* will start reading and sending pulse codes from the starting address of its RAM block. The codes are sent starting from low-address entry. The transmitter will stop the transmission, return to idle state and generate an RMT_CH*n*_TX_END_INT interrupt, when an end-marker (a zero period) is encountered. Also, setting RMT_TX_STOP_CH*n* to 1 stops the transmission and immediately sets the transmitter back to idle. The output level of a transmitter in idle state is determined by the "level" field of the end-marker or by the content of RMT_IDLE_OUT_LV_CH*n*, depending on the configuration of

RMT_IDLE_OUT_EN_CHn.

To transmit more pulse codes than can be fitted in the channel's RAM, users can enable wrap mode by configuring RMT_MEM_TX_WRAP_EN. In this mode, when the transmitter has reached the end-marker in the channel's memory, it will loop back to the first byte. For example, if RMT_MEM_SIZE_CHn is set to 1, the transmitter will start sending data from the address 64 * n, and then the data from higher RAM address. Once the transmitter finishes sending the data from (64 * (n +1) - 1), it will continue sending data from 64 * n till encounters an end-marker. Wrap mode is also applicable for RMT_MEM_SIZE_CHn > 1.

An RMT_CHn_TX_THR_EVENT_INT interrupt will be generated whenever the size of transmitted pulse codes is larger than or equal to the value set by RMT_TX_LIM_CHn. In wrap mode, RMT_TX_LIM_CHn can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CHn_TX_THR_EVENT_INT interrupt is detected, the already used RAM region should be updated by subsequent user defined events. Therefore, when the wrap mode happens the transmitter will seamlessly continue sending the new events.

The output of the transmitter can be modulated using a carrier wave by setting RMT_CARRIER_EN_CHn. The carrier waveform is configurable. In a carrier cycle, the high level lasts for (RMT_CARRIER_HIGH_CHn +1) clock cycles of APB_CLK or REF_TICK, while the low level lasts for (RMT_CARRIER_LOW_CHn +1) clock cycles of APB_CLK or REF_TICK. When RMT_CARRIER_OUT_LV_CHn is set to 1, carrier wave will be added on high-level output signals; while RMT_CARRIER_OUT_LV_CHn is set to 0, carrier wave will be added on low-level output signals. Carrier wave can be added on output signals during modulation, or just added on valid pulse codes (the data stored in RAM), which can be set by configuring RMT_CARRIER_EFF_EN_CHn.

The continuous transmission of the transmitter can be enabled by setting RMT_TX_CONTI_MODE_CHn. When this register is set, the transmitter will send the pulse codes from RAM in loops. If RMT_TX_LOOP_CNT_EN_CHn is set to 1, the transmitter will start counting loop times. Once the counting reaches the value of register RMT_TX_LOOP_NUM_CHn, an RMT_CHn_TX_LOOP_INT interrupt will be generated.

Setting RMT_TX_SIM_EN to 1 will enable multiple channels to start sending data simultaneously. RMT_TX_SIM_CHn will choose which multiple channels are enabled to send data simultaneously.

31.2.5 Receiver

When RMT_RX_EN_CHn is set to 1, the receiver in channel n becomes active, detecting signal levels and measuring clock cycles the signals lasts for. These data will be written in RAM in the form of pulse codes. Receiving ends, when the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CHn. The receiver will return to idle state and generate an RMT_CHn_RX_END_INT interrupt.

The receiver has an input signal filter which can be enabled by configuring RMT_RX_FILTER_EN_CHn. The filter samples input signals continuously, and will detect the signals which remain unchanged for a continuous RMT_RX_FILTER_THRES_CHn APB clock cycles as valid, otherwise, the signals will be detected as invalid. Only the valid signals can pass through this filter. The filter will remove pulses with a length of less than RMT_RX_FILTER_THRES_CHn APB clock cycles.

31.2.6 Interrupts

- RMT_CHn_ERR_INT: Triggered when channel n does not read or write data correctly. For example, if the transmitter still tries to read data from RAM when the RAM is empty, or the receiver still tries to write data into RAM when the RAM is full, this interrupt will be triggered.
- RMT_CHn_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the

value of RMT_TX_LIM_CHn.

- RMT_CHn_TX_END_INT: Triggered when the transmitter has finished transmitting signals.
- RMT_CHn_RX_END_INT: Triggered when the receiver has finished receiving signals.
- RMT_CHn_TX_LOOP_INT: Triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CHn.

31.3 Base Address

Users can access RMT with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 188: RMT Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F416000
PeriBUS2	0x60016000

31.4 Register Summary

The addresses in the following table are relative to RMT base addresses provided in Section 31.3.

Name	Description	Address	Access
Configuration registers			
RMT_CH0CONF0_REG	Channel 0 configuration register 0	0x0010	R/W
RMT_CH0CONF1_REG	Channel 0 configuration register 1	0x0014	varies
RMT_CH1CONF0_REG	Channel 1 configuration register 0	0x0018	R/W
RMT_CH1CONF1_REG	Channel 1 configuration register 1	0x001C	varies
RMT_CH2CONF0_REG	Channel 2 configuration register 0	0x0020	R/W
RMT_CH2CONF1_REG	Channel 2 configuration register 1	0x0024	varies
RMT_CH3CONF0_REG	Channel 3 configuration register 0	0x0028	R/W
RMT_CH3CONF1_REG	Channel 3 configuration register 1	0x002C	varies
RMT_APB_CONF_REG	RMT APB configuration register	0x0080	R/W
RMT_REF_CNT_RST_REG	RMT clock divider reset register	0x0088	R/W
RMT_CH0_RX_CARRIER_RM_REG	Channel 0 carrier remove register	0x008C	R/W
RMT_CH1_RX_CARRIER_RM_REG	Channel 1 carrier remove register	0x0090	R/W
RMT_CH2_RX_CARRIER_RM_REG	Channel 2 carrier remove register	0x0094	R/W
RMT_CH3_RX_CARRIER_RM_REG	Channel 3 carrier remove register	0x0098	R/W
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Channel 0 duty cycle configuration register	0x0060	R/W
RMT_CH1CARRIER_DUTY_REG	Channel 1 duty cycle configuration register	0x0064	R/W
RMT_CH2CARRIER_DUTY_REG	Channel 2 duty cycle configuration register	0x0068	R/W
RMT_CH3CARRIER_DUTY_REG	Channel 3 duty cycle configuration register	0x006C	R/W
Tx event configuration registers			
RMT_CH0_TX_LIM_REG	Channel 0 Tx event configuration register	0x0070	varies
RMT_CH1_TX_LIM_REG	Channel 1 Tx event configuration register	0x0074	varies
RMT_CH2_TX_LIM_REG	Channel 2 Tx event configuration register	0x0078	varies

Name	Description	Address	Access
RMT_CH3_TX_LIM_REG	Channel 3 Tx event configuration register	0x007C	varies
RMT_TX_SIM_REG	Enable RMT simultaneous transmission	0x0084	R/W
Status registers			
RMT_CH0STATUS_REG	Channel 0 status register	0x0030	RO
RMT_CH1STATUS_REG	Channel 1 status register	0x0034	RO
RMT_CH2STATUS_REG	Channel 2 status register	0x0038	RO
RMT_CH3STATUS_REG	Channel 3 status register	0x003C	RO
RMT_CH0ADDR_REG	Channel 0 address register	0x0040	RO
RMT_CH1ADDR_REG	Channel 1 address register	0x0044	RO
RMT_CH2ADDR_REG	Channel 2 address register	0x0048	RO
RMT_CH3ADDR_REG	Channel 3 address register	0x004C	RO
Version register			
RMT_DATE_REG	Version control register	0x00FC	R/W
FIFO R/W registers			
RMT_CH0DATA_REG	Read and write data for channel 0 via APB FIFO	0x0000	RO
RMT_CH1DATA_REG	Read and write data for channel 1 via APB FIFO	0x0004	RO
RMT_CH2DATA_REG	Read and write data for channel 2 via APB FIFO	0x0008	RO
RMT_CH3DATA_REG	Read and write data for channel 3 via APB FIFO	0x000C	RO
Interrupt registers			
RMT_INT_RAW_REG	Raw interrupt status register	0x0050	RO
RMT_INT_ST_REG	Masked interrupt status register	0x0054	RO
RMT_INT_ENA_REG	Interrupt enable register	0x0058	R/W
RMT_INT_CLR_REG	Interrupt clear register	0x005C	WO

31.5 Registers

Register 31.1: RMT_CHnCONF0_REG (n: 0-3) (0x0010+8*n)

(reserved)		RMT_CARRIER_OUT_LV_CHn	RMT_CARRIER_EN_CHn	RMT_CARRIER_EFF_EN_CHn	RMT_MEM_SIZE_CHn	RMT_IDLE_THRES_CHn	RMT_DIV_CNT_CHn	
31	30	29	28	27	26 24	23 8	7 0	
0	0	1	1	1	0x1	0x1000	0x2	Reset

RMT_DIV_CNT_CHn This field is used to configure clock divider for channel n. (R/W)

RMT_IDLE_THRES_CHn Receiving ends when no edge is detected on input signals for continuous clock cycles larger than this register value. (R/W)

RMT_MEM_SIZE_CHn This field is used to configure the maximum blocks allocated to channel n. The valid range is from 1 ~ 4-n. (R/W)

RMT_CARRIER_EFF_EN_CHn 1: Add carrier modulation on output signals only at data sending state for channel n. 0: Add carrier modulation on signals at all states for channel n. States here include idle state(ST_IDLE), reading data from RAM (ST_RD_MEM), and sending data stored in RAM (ST_SEND). Only valid when RMT_CARRIER_EN_CHn is set to 1. (R/W)

RMT_CARRIER_EN_CHn This bit is used to enable carrier modulation for channel n. 1: Add carrier modulation on output signals. 0: No carrier modulation is added on output signals. (R/W)

RMT_CARRIER_OUT_LV_CHn This bit is used to configure the position of carrier wave for channel n. 1'h0: Add carrier wave on low-level output signals. 1'h1: Add carrier wave on high-level output signals. (R/W)

Register 31.2: RMT_CHnCONF1_REG (n: 0-3) (0x0014+8*n)

(reserved)	RMT_TX_STOP_CHn	RMT_IDLE_OUT_EN_CHn	RMT_IDLE_OUT_LV_CHn	RMT_REF_ALWAYS_ON_CHn	RMT_CHK_RX_CARRIER_EN_CHn	RMT_RX_FILTER_THRES_CHn	RMT_RX_FILTER_EN_CHn	RMT_TX_CONTI_MODE_CHn	RMT_MEM_OWNER_CHn	RMT_APB_MEM_RST_CHn	RMT_MEM_RD_RST_CHn	RMT_MEM_WR_RST_CHn	RMT_RX_EN_CHn	RMT_TX_START_CHn	
31 ... 21	20	19	18	17	16	15 ... 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0xf	0	0	1	0	0	0	0	0	Reset

RMT_TX_START_CHn Set this bit to start sending data on channel n. (R/W)

RMT_RX_EN_CHn Set this bit to enable receiver to receive data on channel n. (R/W)

RMT_MEM_WR_RST_CHn Set this bit to reset RAM write address accessed by the receiver for channel n. (WO)

RMT_MEM_RD_RST_CHn Set this bit to reset RAM read address accessed by the transmitter for channel n. (WO)

RMT_MEM_OWNER_CHn This bit marks the ownership of channel n's RAM block. 1'h1: Receiver is using the RAM. 1'h0: Transmitter is using the RAM. (R/W)

RMT_TX_CONTI_MODE_CHn Set this bit to restart transmission in continuous mode from the first data in channel n. (R/W)

RMT_RX_FILTER_EN_CHn Set this bit to enable the receiver's filter for channel n. (R/W)

RMT_RX_FILTER_THRES_CHn Set this field to ignore the input pulse when its width is less than RMT_RX_FILTER_THRES_CHn APB clock cycles in receive mode. (R/W)

RMT_CHK_RX_CARRIER_EN_CHn Set this bit to enable memory loop read mode when carrier modulation is enabled for channel n. (R/W)

RMT_REF_ALWAYS_ON_CHn Set this bit to select a base clock for channel n. 1'h1: APB_CLK; 1'h0: REF_TICK (R/W)

RMT_IDLE_OUT_LV_CHn This bit configures the level of output signals in channel n when the transmitter is in idle state. (R/W)

RMT_IDLE_OUT_EN_CHn This is the output enable bit for channel n in idle state. (R/W)

RMT_TX_STOP_CHn Set this bit to stop the transmitter of channel n sending data out. (R/W)

Register 31.3: RMT_APB_CONF_REG (0x0080)

31	30 … 5	4	3	2	1	0	
RMT_CLK_EN	(reserved)	RMT_MEM_FORCE_PU	RMT_MEM_FORCE_PD	RMT_MEM_CLK_FORCE_ON	RMT_MEM_TX_WRAP_EN	RMT_APB_FIFO_MASK	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	1	0	0	Reset

RMT_APB_FIFO_MASK 1'h1: Access memory directly; 1'h0: Access memory via APB FIFO. (R/W)

RMT_MEM_TX_WRAP_EN Set this bit to enable wrap mode. (R/W)

RMT_MEM_CLK_FORCE_ON Set this bit to enable the clock for RAM when RMT module starts working; disable this clock when RMT stops working, to achieve low-power scheme. (R/W)

RMT_MEM_FORCE_PD Set this bit to power down RMT memory. (R/W)

RMT_MEM_FORCE_PU 1: Disable RAM's Light-sleep power down function. 0: power down RMT RAM when RMT is in Light-sleep mode. (R/W)

RMT_CLK_EN Clock gating enable bit for RMT registers to achieve low-power scheme. 1: Power up drive clock for RMT registers. 0: Power down drive clock for RMT registers. (R/W)

Register 31.4: RMT_REF_CNT_RST_REG (0x0088)

31 … 4	3	2	1	0	
(reserved)	RMT_REF_CNT_RST_CH3	RMT_REF_CNT_RST_CH2	RMT_REF_CNT_RST_CH1	RMT_REF_CNT_RST_CH0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

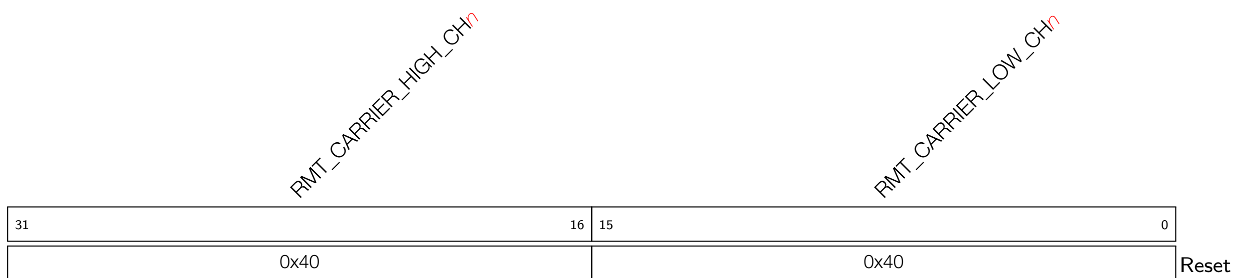
RMT_REF_CNT_RST_CH*n* This bit is used to reset the clock divider of channel *n*. (R/W)

Register 31.5: RMT_CHn_RX_CARRIER_RM_REG (n: 0-3) (0x008C+4*n)

RMT_CARRIER_LOW_THRES_CHn The low level period in carrier modulation mode is (RMT_CARRIER_LOW_THRES_CHn + 1) clock cycles for channel n. (R/W)

RMT_CARRIER_HIGH_THRES_CHn The high level period in carrier modulation mode is (RMT_CARRIER_HIGH_THRES_CHn + 1) clock cycles for channel n. (R/W)

Register 31.6: RMT_CHnCARRIER_DUTY_REG (n: 0-3) (0x0060+4*n)



RMT_CARRIER_LOW_CHn This field is used to configure the clock cycles of carrier wave at low level for channel n. (R/W)

RMT_CARRIER_HIGH_CHn This field is used to configure the clock cycles of carrier wave at high level for channel n. (R/W)

Register 31.7: RMT_CHn_TX_LIM_REG (n: 0-3) (0x0070+4*n)

(reserved)	RMT_LOOP_COUNT_RESET_CHn	RMT_TX_LOOP_CNT_EN_CHn	RMT_TX_LOOP_NUM_CHn	RMT_TX_LIM_CHn	
31 ... 21	20	19	18 ... 9	8 ... 0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0x0	0x80	Reset

RMT_TX_LIM_CHn This field is used to configure the maximum entries channel *n* can send out. When RMT_MEM_SIZE_CHn = 1, this field can be set to any value among 0 ~ 128 (64 * 32/16 = 128); when RMT_MEM_SIZE_CHn > 1, this field can be set to any value among (0 ~ 128) * RMT_MEM_SIZE_CHn. (R/W)

RMT_TX_LOOP_NUM_CHn This field is used to configure the maximum loop times when continuous transmission mode is enabled. (R/W)

RMT_TX_LOOP_CNT_EN_CHn This bit is used to enable loop counting. (R/W)

RMT_LOOP_COUNT_RESET_CHn This bit is used to reset loop counting when continuous transmission mode is valid. (WO)

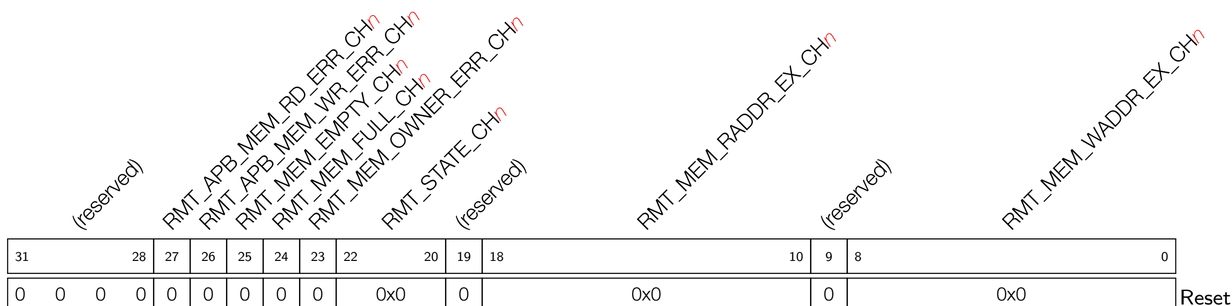
Register 31.8: RMT_TX_SIM_REG (0x0084)

(reserved)	RMT_TX_SIM_EN	RMT_TX_SIM_CH3	RMT_TX_SIM_CH2	RMT_TX_SIM_CH1	RMT_TX_SIM_CH0	
31 ... 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	Reset

RMT_TX_SIM_CHn Set this bit to enable channel *n* to start sending data simultaneously with other enabled channels. (R/W)

RMT_TX_SIM_EN This bit is used to enable multiple channels to start sending data simultaneously. (R/W)

Register 31.9: RMT_CH*n*STATUS_REG (*n*: 0-3) (0x0030+4*n)



RMT_MEM_WADDR_EX_CH*n* This field records the memory address offset when receiver of channel *n* is using the RAM. (RO)

RMT_MEM_RADDR_EX_CH*n* This field records the memory address offset when transmitter of channel *n* is using the RAM. (RO)

RMT_STATE_CH*n* This field records the FSM status of channel *n*. (RO)

RMT_MEM_OWNER_ERR_CH*n* This status bit will be set when the ownership of memory block is violated. (RO)

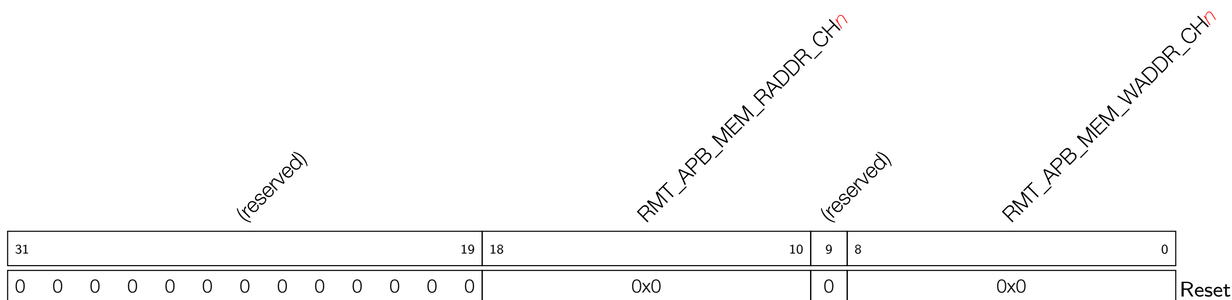
RMT_MEM_FULL_CH*n* This status bit will be set if the receiver receives more data than the memory allows. (RO)

RMT_MEM_EMPTY_CH*n* This status bit will be set when the data to be sent is more than the memory allows and the wrap mode is disabled. (RO)

RMT_APB_MEM_WR_ERR_CH*n* This status bit will be set if the offset address is out of memory size when channel *n* writes RAM via APB bus. (RO)

RMT_APB_MEM_RD_ERR_CH*n* This status bit will be set if the offset address is out of memory size when channel *n* reads RAM via APB bus. (RO)

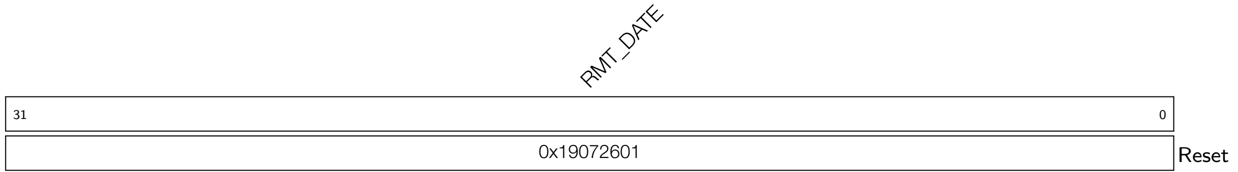
Register 31.10: RMT_CH*n*ADDR_REG (*n*: 0-3) (0x0040+4*n)



RMT_APB_MEM_WADDR_CH*n* This field records the memory address offset when channel *n* writes RAM via APB bus. (RO)

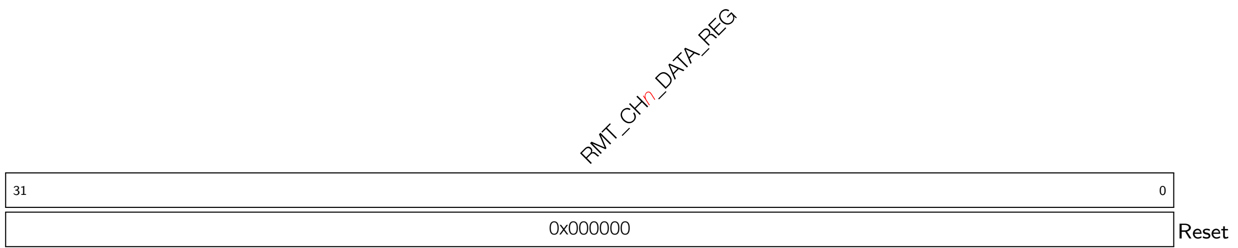
RMT_APB_MEM_RADDR_CH*n* This field records the memory address offset when channel *n* reads RAM via APB bus. (RO)

Register 31.11: RMT_DATE_REG (0x00FC)



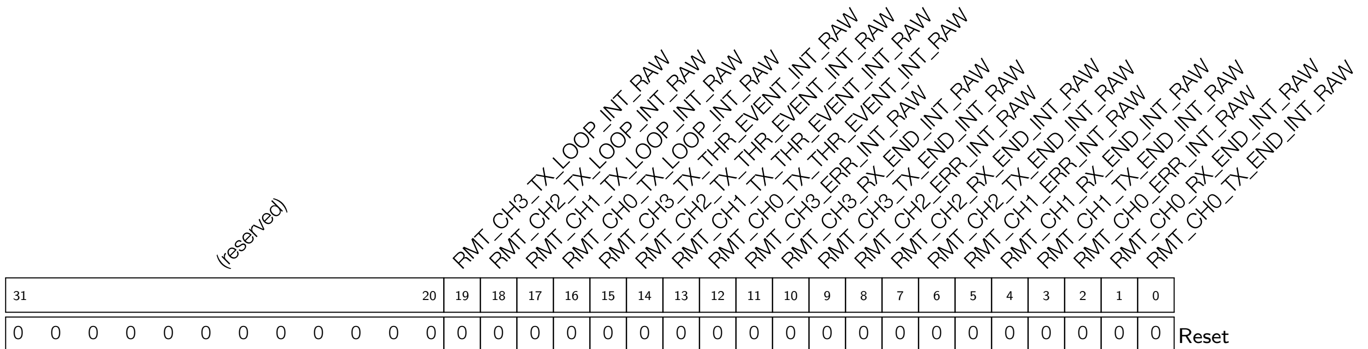
RMT_DATE Version control register. (R/W)

Register 31.12: RMT_CHnDATA_REG (n: 0-3) (0x0000+4*n)



RMT_CHn_DATA_REG This register is used to read and write data for channel n via APB FIFO. (RO)

Register 31.13: RMT_INT_RAW_REG (0x0050)



RMT_CHn_TX_END_INT_RAW The interrupt raw bit for channel n. Triggered when transmitting ends. (RO)

RMT_CHn_RX_END_INT_RAW The interrupt raw bit for channel n. Triggered when receiving ends. (RO)

RMT_CHn_ERR_INT_RAW The interrupt raw bit for channel n. Triggered when error occurs. (RO)

RMT_CHn_TX_THR_EVENT_INT_RAW The interrupt raw bit for channel n. Triggered when transmitter sends more data than configured value. (RO)

RMT_CHn_TX_LOOP_INT_RAW The interrupt raw bit for channel n. Triggered when loop counting reaches the configured threshold value. (RO)

Register 31.14: RMT_INT_ST_REG (0x0054)

(reserved)	RMT_CH3_TX_LOOP_INT_ST	RMT_CH2_TX_LOOP_INT_ST	RMT_CH1_TX_LOOP_INT_ST	RMT_CH0_TX_LOOP_INT_ST	RMT_CH3_TX_THR_EVENT_INT_ST	RMT_CH2_TX_THR_EVENT_INT_ST	RMT_CH1_TX_THR_EVENT_INT_ST	RMT_CH0_TX_THR_EVENT_INT_ST	RMT_CH3_ERR_INT_ST	RMT_CH3_RX_END_INT_ST	RMT_CH3_TX_END_INT_ST	RMT_CH2_ERR_INT_ST	RMT_CH2_RX_END_INT_ST	RMT_CH2_TX_END_INT_ST	RMT_CH1_ERR_INT_ST	RMT_CH1_RX_END_INT_ST	RMT_CH1_TX_END_INT_ST	RMT_CH0_ERR_INT_ST	RMT_CH0_RX_END_INT_ST	RMT_CH0_TX_END_INT_ST	
31 ... 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_END_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_END_INT. (RO)

RMT_CH*n*_RX_END_INT_ST The masked interrupt status bit for RMT_CH*n*_RX_END_INT. (RO)

RMT_CH*n*_ERR_INT_ST The masked interrupt status bit for RMT_CH*n*_ERR_INT. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_THR_EVENT_INT. (RO)

RMT_CH*n*_TX_LOOP_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_LOOP_INT. (RO)

Register 31.15: RMT_INT_ENA_REG (0x0058)

(reserved)	RMT_CH3_TX_LOOP_INT_ENA	RMT_CH2_TX_LOOP_INT_ENA	RMT_CH1_TX_LOOP_INT_ENA	RMT_CH0_TX_LOOP_INT_ENA	RMT_CH3_TX_THR_EVENT_INT_ENA	RMT_CH2_TX_THR_EVENT_INT_ENA	RMT_CH1_TX_THR_EVENT_INT_ENA	RMT_CH0_TX_THR_EVENT_INT_ENA	RMT_CH3_ERR_INT_ENA	RMT_CH3_RX_END_INT_ENA	RMT_CH3_TX_END_INT_ENA	RMT_CH2_ERR_INT_ENA	RMT_CH2_RX_END_INT_ENA	RMT_CH2_TX_END_INT_ENA	RMT_CH1_ERR_INT_ENA	RMT_CH1_RX_END_INT_ENA	RMT_CH1_TX_END_INT_ENA	RMT_CH0_ERR_INT_ENA	RMT_CH0_RX_END_INT_ENA	RMT_CH0_TX_END_INT_ENA	
31 ... 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_END_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_END_INT. (R/W)

RMT_CH*n*_RX_END_INT_ENA Interrupt enable bit for RMT_CH*n*_RX_END_INT. (R/W)

RMT_CH*n*_ERR_INT_ENA Interrupt enable bit for RMT_CH*n*_ERR_INT. (R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_THR_EVENT_INT. (R/W)

RMT_CH*n*_TX_LOOP_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_LOOP_INT. (R/W)

Register 31.16: RMT_INT_CLR_REG (0x005C)

(reserved)	RMT_CH3_TX_LOOP_INT_CLR	RMT_CH2_TX_LOOP_INT_CLR	RMT_CH1_TX_LOOP_INT_CLR	RMT_CH0_TX_LOOP_INT_CLR	RMT_CH3_TX_THR_EVENT_INT_CLR	RMT_CH2_TX_THR_EVENT_INT_CLR	RMT_CH1_TX_THR_EVENT_INT_CLR	RMT_CH0_TX_THR_EVENT_INT_CLR	RMT_CH3_ERR_INT_CLR	RMT_CH3_RX_END_INT_CLR	RMT_CH3_TX_END_INT_CLR	RMT_CH2_ERR_INT_CLR	RMT_CH2_RX_END_INT_CLR	RMT_CH2_TX_END_INT_CLR	RMT_CH1_ERR_INT_CLR	RMT_CH1_RX_END_INT_CLR	RMT_CH1_TX_END_INT_CLR	RMT_CH0_ERR_INT_CLR	RMT_CH0_RX_END_INT_CLR	RMT_CH0_TX_END_INT_CLR	
31 ... 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_END_INT_CLR Set this bit to clear RMT_CH*n*_TX_END_INT interrupt. (WO)

RMT_CH*n*_RX_END_INT_CLR Set this bit to clear RMT_CH*n*_RX_END_INT interrupt. (WO)

RMT_CH*n*_ERR_INT_CLR Set this bit to clear RMT_CH*n*_ERR_INT interrupt. (WO)

RMT_CH*n*_TX_THR_EVENT_INT_CLR Set this bit to clear RMT_CH*n*_TX_THR_EVENT_INT interrupt. (WO)

RMT_CH*n*_TX_LOOP_INT_CLR Set this bit to clear RMT_CH*n*_TX_LOOP_INT interrupt. (WO)

32. On-Chip Sensor and Analog Signal Processing

32.1 Overview

ESP32-S2 provides the following on-chip sensor and signal processing peripherals:

- One temperature sensor for measuring the internal temperature of the ESP32-S2 chip.
- Two 13-bit Successive Approximation ADCs (SAR ADCs) controlled by five dedicated controllers that can input analog signals from total of 20 channels. The SAR ADCs can operate in a high-performance mode or a low-power mode.
- Two 8-bit independent DACs to generate analog signals. The DACs also support a cosine wave output mode.

32.2 SAR ADCs

32.2.1 Overview

ESP32-S2 integrates two 13-bit SAR ADCs, which are able to measure analog signals from up to 20 analog pads. It is also possible to measure internal signals, such as vdd33. The SAR ADCs are managed by five dedicated controllers:

- Two digital controllers: DIG ADC1 CTRL and DIG ADC2 CTRL, designed for high-performance multi-channel scanning and DMA continuous conversion.
- Two RTC controllers: RTC ADC1 CTRL and RTC ADC2 CTRL, designed for single conversion mode and low power mode.
- One internal controller: PWDET/PKDET CTRL, for power and peak detection (only used by Wi-Fi).

A diagram of the SAR ADCs is shown in Figure 32-1.

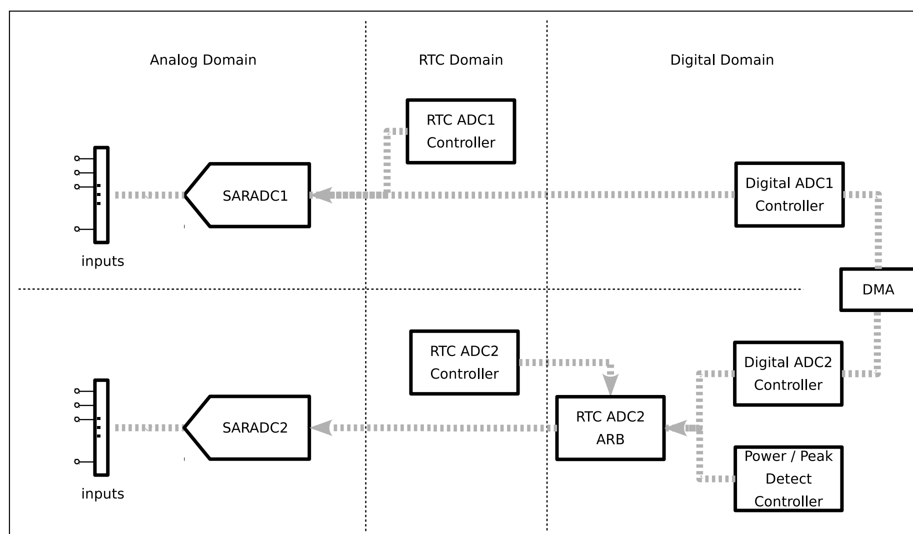


Figure 32-1. SAR ADC Overview

32.2.2 Features

- Two SAR ADCs that can operate separately (i.e., convert data simultaneously)
- 12-bit or 13-bit sampling resolution

  - RTC ADC: 13-bit sampling resolution at most
  - DIG ADC: 12-bit sampling resolution at most
- Up to 20 analog input pads
- Configurable channel-scanning duration
- Two RTC ADC controllers, with the following features:
  - Single conversion mode
  - Operation in Deep-sleep mode
  - Can be controlled by ULP coprocessor
- Two DIG ADC controllers, with the following features:
  - Sampled data is transferred using DMA
  - Single conversion mode and continuous conversion mode
  - Multiple channel-scanning mode
  - User-defined channel-scanning sequence
  - Hardware IIR filter with configurable filter coefficient
  - Threshold monitoring. An interrupt will be triggered when the value is greater or less than the threshold.
- One PWDET/PKDET controller only for Wi-Fi internal use to:
  - Monitor power
  - Monitor internal voltage

The differences between the five ADC controllers are summarized in Table 190.

Table 190: SAR ADC Controllers

Feature	RTC ADC1	RTC ADC2	DIG ADC1	DIG ADC2	PWDET
Control DAC	Y	-	-	-	-
Support Deep-sleep mode	Y	Y	-	-	-
Controlled by ULP coprocessor	Y	Y	-	-	-
Support vdd33 detection	-	Y	-	Y	-
Support PWDET/PKDET detection	-	-	-	-	Y
Support DMA continuous conversion	-	-	Y	Y	-
Support 13-bit sampling resolution	Y	Y	-	-	-

32.2.3 Functional Description

The major components of SAR ADCs and their interconnections are shown in Figure 32-2.

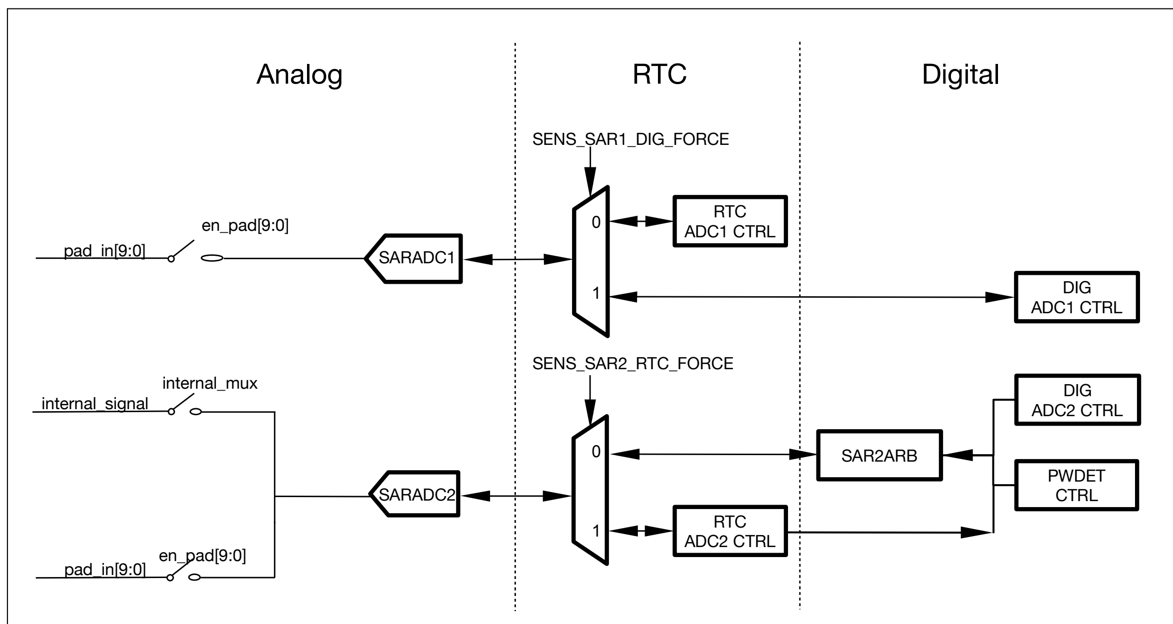


Figure 32-2. SAR ADC Function Overview

32.2.3.1 Input Signals

In order to sample an analog signal, an SAR ADC must first select the analog pad or internal signal to measure via an internal multiplexer. A summary of all the analog signals that may be sent to the SAR ADC module for processing by either ADC1 or ADC2 are presented in Table 191.

Table 191: SAR ADC Input Signals

Signal Name	Channel	Processed by
GPIO1	0	SAR ADC1
GPIO2	1	
GPIO3	2	
GPIO4	3	
GPIO5	4	
GPIO6	5	
GPIO7	6	
GPIO8	7	
GPIO9	8	
GPIO10	9	
GPIO11	0	SAR ADC2
GPIO12	1	
GPIO13	2	
GPIO14	3	
GPIO15	4	
GPIO16	5	
GPIO17	6	
GPIO18	7	
GPIO19	8	
GPIO20	9	

Signal Name	Channel	Processed by
pa_pkdet1	n/a	
pa_pkdet2	n/a	
vdd33	n/a	

32.2.3.2 ADC Conversion and Attenuation

When the SAR ADCs convert (sample) an analog voltage, the resolution (12-bit or 13-bit) of the conversion spans voltage range from 0 mV to V_{ref} which is the SAR ADCs internal reference voltage. Thus the output value of the conversion $(data)$ will map to analog voltage V_{data} using the following formulas:

- For 12-bit conversion: $V_{data} = \frac{V_{ref}}{4095} \times data$
- For 13-bit conversion: $V_{data} = \frac{V_{ref}}{8191} \times data$

In order to convert voltages larger than V_{ref}, input signals can be attenuated before being input into the SAR ADCs.The attenuation can be configured to 0 dB, 2.5 dB, 6 dB, and 11 dB.

32.2.4 RTC ADC Controllers

The RTC ADC controllers (RTC ADC1 CTRL and RTC ADC2 CTRL) are powered in the RTC power domain, thus allow the SAR ADCs to conduct measurements at a low frequency with minimal power consumption. The outline of a single controller's function is shown in Figure 32-3.

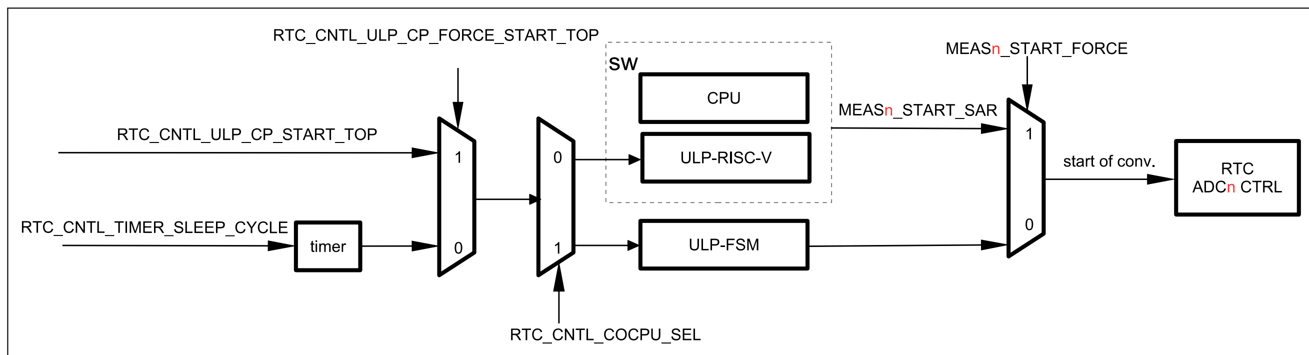


Figure 32-3. RTC SAR ADC Outline

The RTC ADC controllers can be triggered to start a conversion by software or by the ULP coprocessor (ULP-FSM). SENS_MEAS*n*_START_FORCE will select whether an RTC ADC controller is triggered by software or by the ULP-FSM.

To trigger the RTC ADC controller to start a conversion by software, set SENS_MEAS*n*_START_SAR. When the conversion is complete, SENS_MEAS*n*_DONE_SAR will be set and the conversion result will be stored in SENS_MEAS*n*_DATA_SAR.

The RTC ADC controllers are intertwined with the ULP coprocessor, as the ULP coprocessor has a built-in instruction to start an ADC conversion. In many cases, the controllers need to cooperate with the ULP coprocessor, e.g.:

- When the controllers periodically monitor a channel during Deep-sleep, ULP coprocessor is the only source to trigger ADC sampling by configuring RTC registers.

- Continuous scanning or DMA is not supported by the controllers. However, it is possible with the help of ULP coprocessor to scan channels continuously in a sequence.

32.2.5 DIG ADC Controllers

Compared to the RTC ADC controllers, the DIG ADC controllers are optimized for performance and throughput. The main features of the DIG ADC controllers are outlined below:

- High performance: The clock of the DIG ADC controller is much faster, thus the sample rate is much higher.
- Up to 12-bit sampling resolution
- Support single-channel scanning, double-channel scanning, or alternate-channel scanning mode. The measurement rules for each SAR ADC are defined in pattern tables.
- Scanning can be started by software or by DIG ADC timer, depending on the configuration of APB_SARADC_START_FORCE. Note that DIG ADC timer is enabled by APB_SARADC_TIMER_EN and its trigger period can be configured by APB_SARADC_TIMER_TARGET.
- Sampled data is transferred to memory via DMA.
- An interrupt will be generated when scanning is finished.

32.2.5.1 Workflow of DIG ADC Controller

Note:

The term "start of conversion" is not used in this section, because there is no direct access to starting a single SAR analog-to-digital conversion. Each conversion is started automatically by the DIG ADC controller according to a pattern table. Instead, the term "start of scan" will be used to indicate that the DIG ADC controller will scan a sequence of channels according to a pattern table.

Figure 32-4 shows a diagram of the DIG ADC controllers.

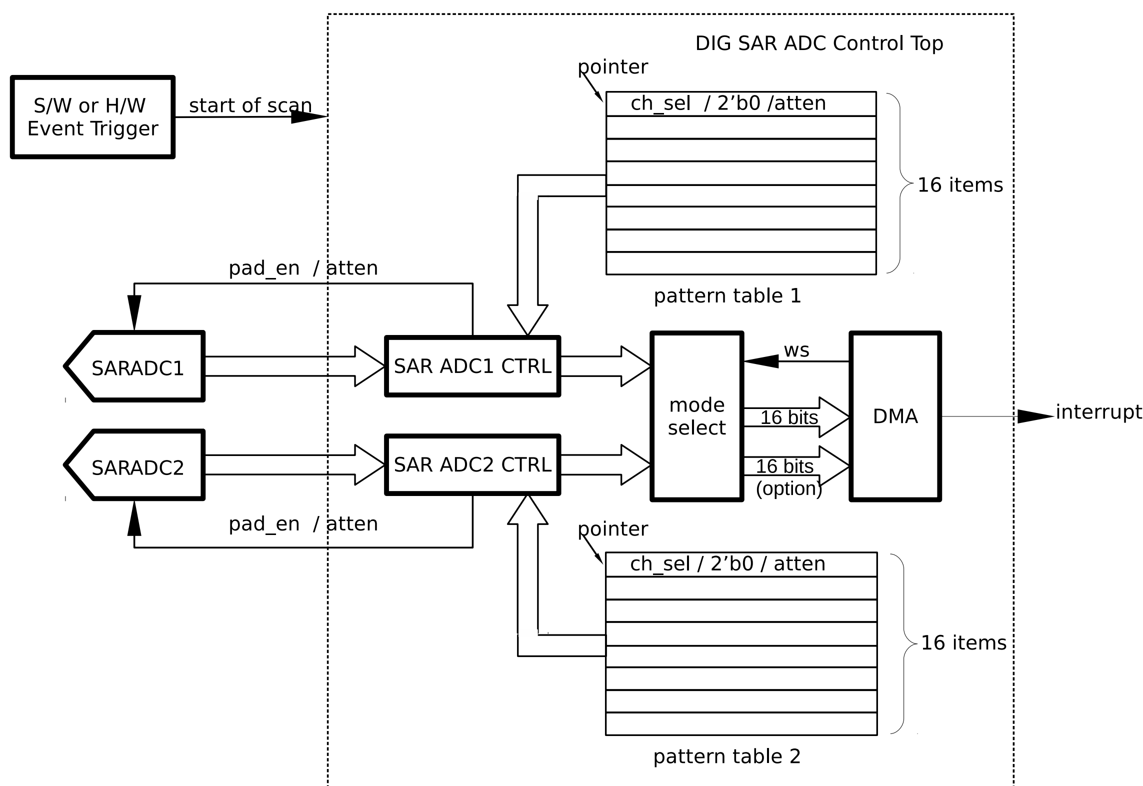


Figure 32-4. Diagram of DIG ADC Controllers

A measurement rule instructs an ADC to sample a channel (and optionally attenuate the signal of the sampled channel). Each of the DIG ADC controllers has a rules table, and each table can store 16 rules. When scanning starts, the controller reads the measurement rules one-by-one from their pattern table. For each controller, a scanning sequence can include 16 different rules at most, before repeating itself.

The 8-bit item (pattern table register) is composed of three fields that contain channel and attenuation information, as shown in Table 192.

Table 192: Fields of Pattern Table Register

Pattern Table Register[7:0]		
ch_sel[3:0]	null	atten[1:0]
Channel	Reserved	Attenuation

The scanning mode configured in APB_SARADC_WORK_MODE determines whether the DIG ADC controllers operate completely independently, or in an alternating or synchronized manner. The following scanning modes are supported:

- Single-channel scanning: SAR ADC1 and SAR ADC2 will operate independently according to their own respective pattern tables.
- Double-channel scanning: SAR ADC1 and SAR ADC2 will sample simultaneously (i.e., the two DIG ADC controllers will progress down their own pattern tables in a synchronized manner).
- Alternate-channel scanning: SAR ADC1 and SAR ADC2 will sample alternately (i.e., the two DIG ADC controllers will take turns when progressing down their own pattern tables).

The final data sent to the DMA by each conversion is 16-bit. This data is composed of the 12/11-bit ADC result and some additional information related to the scanning mode:

- For single-channel scanning, 4-bit information on channel selection is added.
- For double-channel scanning or alternate-channel scanning, 4-bit information on channel selection is added plus one extra bit indicating which SAR ADC is the data from.

Two DMA data formats, Type I and Type II, are used to store the data for the above-mentioned two situations, see Table 193 and Table 194.

Table 193: DMA Data Format (Type I)

DMA Data Format[15:0]	
ch_sel[3:0]	data[11:0]
Channel	SAR ADC Data

Table 194: DMA Data Format (Type II)

DMA Data Format[15:0]		
sar_sel	ch_sel[3:0]	data[10:0]
SAR ADC*n* (*n*= 1 or 2)	Channel	SAR ADC Data

DIG ADCs support 12-bit resolution:

- Type I data format: 12-bit resolution at most.
- Type II data format: 11-bit resolution at most.

32.2.5.2 DMA

DIG ADC controllers support direct memory access via the SPI3 DMA and is triggered by DIG ADC dedicated timer. Therefore, the DIG ADC controllers should not be used when SPI3 DMA is already being used. Users can switch the DMA data path to DIG ADC by configuring APB_SARADC_APB_ADC_TRANS via software. For specific DMA configuration, please refer to SPI3 DMA control.

32.2.5.3 ADC Filter

The DIG ADC controllers supports automatic filtering of sampled ADC data. The filter's formula is shown below:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} - 0.5$$

- $data_{cur}$ is the filtered data value.
- $data_{in}$ is the sampled data value from the ADC.
- $data_{prev}$ is the last filtered data value.
- k is the filter coefficient.

To configure the filter, set the filter coefficient using APB_SARADC_ADCn_FILTER_FACTOR then set APB_SARADC_ADCn_FILTER_EN to enable the filter. The filter is applied to each DIG ADC controller. However, due to the filter being dependent on previous filtered values ($data_{prev}$), **only one channel on each DIG ADC controller should be scanned when filters are used** so that data of different channels are not mixed by the filter.

32.2.5.4 Threshold Monitoring

Each DIG ADC controller has a threshold monitor that will trigger an interrupt if any of the sampled ADC values exceeds (or falls below) a configured threshold.

- Set the bit APB_SARADC_ADCn_THRES_EN to enable threshold monitoring.
- Configure APB_SARADC_ADCn_THRES to set the threshold value.
- Configure APB_SARADC_ADCn_THRES_MODE to set the conditions:
  - 0: When ADC_DATA < APB_SARADC_ADCn_THRES, a monitor interrupt will be generated.
  - 1: When ADC_DATA >= APB_SARADC_ADCn_THRES, a monitor interrupt will be generated.

Note: Threshold monitoring checks the ADC values on all the channels selected by SAR ADC1 or SAR ADC2, respectively.

32.2.6 SAR ADC2 Arbiter

SAR ADC2 can be controlled by three controllers, namely RTC ADC2 CTRL, DIG ADC2 CTRL, and PWDET/PKDET CTRL. To avoid any possible conflicts and to improve the efficiency of SAR ADC2, ESP32-S2 provides access arbitration for SAR ADC2. The arbiter supports fair arbitration and fixed priority arbitration.

- Fair arbitration mode (cyclic priority arbitration) can be enabled by clearing APB_SARADC_ADC_ARB_FIX_PRIORITY in register APB_SARADC_ARB_CTRL_REG.
- In fixed priority arbitration, users can set the bits in register APB_SARADC_ARB_CTRL_REG, including APB_SARADC_ADC_ARB_RTC_PRIORITY (for RTC ADC2 CTRL), APB_SARADC_ADC_ARB_APB_PRIORITY (for DIG ADC2 CTRL), and APB_SARADC_ADC_ARB_WIFI_PRIORITY (for PWDET CTRL), to configure the priorities for these controllers. A larger value indicates a higher priority.

The arbiter ensures that a higher priority controller can always start a conversion (sample) when required, regardless of whether a lower priority controller already has a conversion in progress. If a higher priority controller starts a conversion whilst the ADC already has a conversion in progress from a lower priority controller, the conversion in progress will be interrupted (stopped). The higher priority controller will then start its conversion.

If a lower priority controller attempts to start a conversion whilst the ADC has a conversion in progress from a higher priority controller, the lower priority controller will not start the conversion.

The value returned by a conversion that is interrupted or not started will be invalid. Therefore, certain data flags are embedded into the output data value to indicate whether the conversion is valid or not.

Note:

- The data flag for RTC ADC2 CTRL is the two higher bits of SENS_MEAS2_DATA_SAR.
  - 2'b10: Conversion is interrupted.

- 2'b01: Conversion is not started.
- 2'b00: The data is valid.

- The data flag for DIG ADC2 CTRL is the ch_sel[3:0] bits in DMA data.
  - 4'b1111: Conversion is interrupted.
  - 4'b1110: Conversion is not started.
  - Corresponding channel No.: The data is valid.
- The data flag for PWDET/PKDET CTRL is the two higher bits of the sampling result.
  - 2'b10: Conversion is interrupted.
  - 2'b01: Conversion is not started.
  - 2'b00: The data is valid.

Users can configure APB_SARADC_ADC_ARB_GRANT_FORCE to mask the arbiter, and set the bits APB_SARADC_ADC_ARB_RTC_FORCE, APB_SARADC_ADC_ARB_WIFI_FORCE, and APB_SARADC_ADC_ARB_APB_FORCE to authorize corresponding controllers.

Note:

- When the arbiter is masked, only one of the above APB_SARADC_ADC_ARB_XXX_FORCE bits can be set to 1.
- The arbiter uses APB_CLK as its clock source. When the clock frequency is 8 MHz or lower, the arbiter must be masked.
- In sleep mode, the SENS_SAR2_RTC_FORCE in register SENS_SAR_MEAS2_MUX_REG should be set to 1, masking the arbiter and all the signals from controllers except the RTC controllers.

32.3 DACs

32.3.1 Overview

ESP32-S2 comes with two built-in 8-bit DACs used to convert digital values into analog output signals (up to two of them). The DACs also support to output cosine waves.

32.3.2 Features

- Two 8-bit DAC converters
- Independent or synchronous conversion in both channels
- Voltage reference from VDD3P3_RTC_IO pin
- Cosine waves output
- DMA capability
- Fully controlled by ULP coprocessor via registers. See Chapter 1 *ULP Coprocessor (ULP)*.

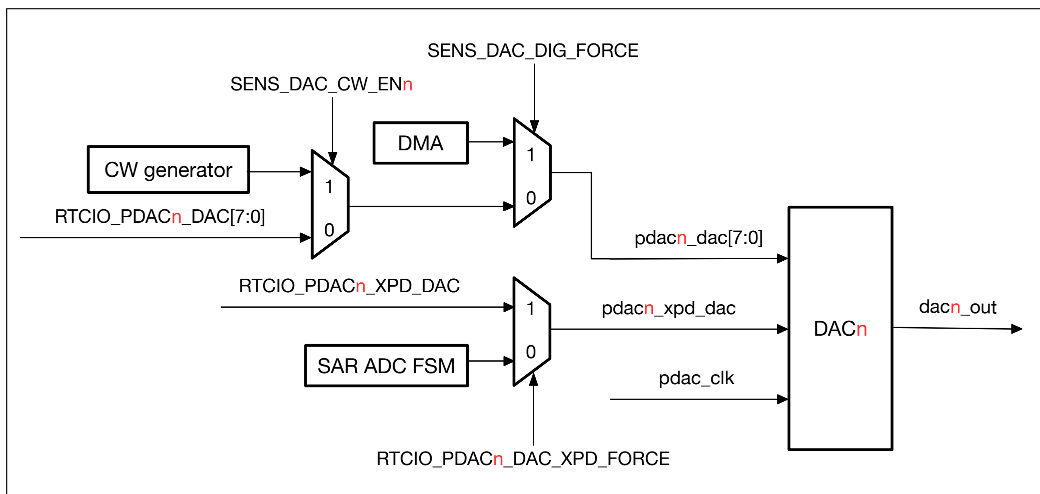


Figure 32-5. Diagram of DAC Function

32.3.3 DAC Conversion

The two 8-bit DAC channels can be configured independently. For each DAC channel, the output analog voltage can be calculated as follows:

$$\text{DAC}n\text{_OUT} = \text{VDD3P3_RTC_IO} \cdot \text{PDAC}n\text{_DAC}/255$$

- VDD3P3_RTC_IO is the voltage on pin VDD3P3_RTC_IO (usually 3.3 V).
- PDAC*n*_DAC carries the digital value to be converted into an analog voltage and comes from multiple sources: cosine waveform generator, the register RTCIO_PAD_DAC*n*_REG, or DMA.

The conversion can be started by the register RTCIO_PDAC*n*_XPD_DAC, and controlled by the software or SAR ADC FSM. For more information, please see Figure 32-5.

32.3.4 Cosine Wave Generator

The cosine wave (CW) generator can be used to generate a cosine or sine tone. Figure 32-6 shows The function of CW generator.

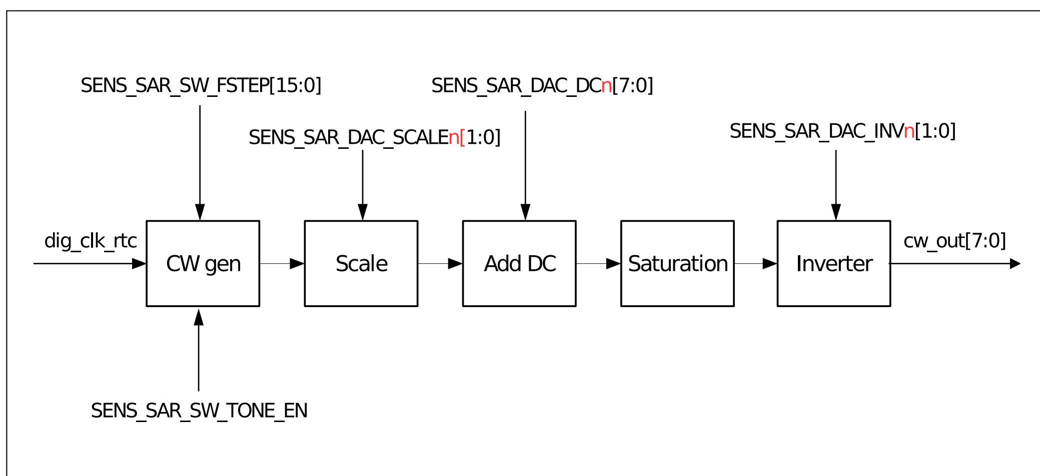


Figure 32-6. Workflow of CW Generator

CW generator features:

- Adjustable frequency
  The frequency of CW can be adjusted by register SENS_SW_FSTEP:

$$freq = dig_clk_rtc_freq \cdot SENS_SW_FSTEP/65535$$

  Typically, the frequency of dig_clk_rtc is 8 MHz.

- Scaling
  The waveform amplitude can be scaled by 1, 1/2, 1/4, or 1/8 by configuring the register SENS_DAC_SCALE*n*.

- DC offset
  The register SENS_DAC_DC*n* may introduce DC offset, leading to a saturated result.

- Phase shift
  A phase-shift of 0° or 180° can be added by setting register SENS_DAC_INV*n*.
  - 2: 0°
  - 3: 180°

32.3.5 DMA Support

A DMA controller with dual DAC channels can be used to set the output of two DAC channels. By configuring the register SENS_DAC_DIG_FORCE and the bit APB_SARADC_APB_DAC_TRANS in the register APB_SARADC_APB_DAC_CTRL_REG, users can connect DIG_SARADC_CLK to DAC clk, and SPI3_DMA_OUT to DAC_DATA for direct memory access.

For details, please refer to Chapter 2 *DMA Controller (DMA)*.

32.4 Temperature Sensor

32.4.1 Overview

ESP32-S2 provides a temperature sensor to monitor temperature changes in real time.

32.4.2 Features

- Monitored in real time by ULP coprocessor when in low-power mode
- Triggered by software or by ULP coprocessor
- Configurable temperature offset based on the environment, to improve the accuracy
- Adjustable measurement range

32.4.3 Operation Sequence

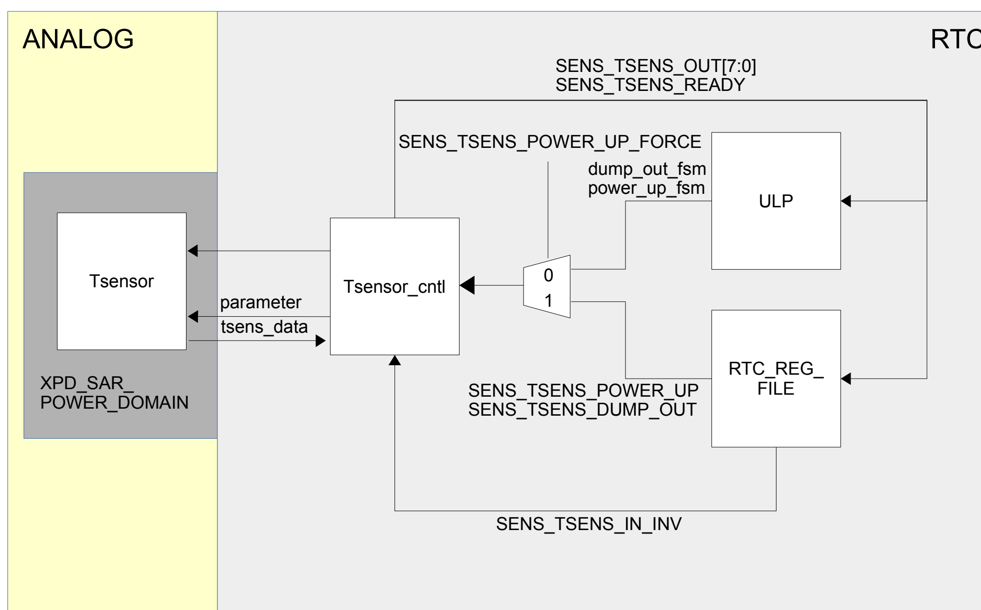


Figure 32-7. Structure of Temperature Sensor

As shown in Figure 32-7, the temperature sensor can be started by software or by ULP coprocessor:

- Started by software, i.e. by CPU or ULP-RISC-V configuring related registers:
  - Start the temperature sensor by setting the registers SENS_TSENS_POWER_UP_FORCE and SENS_TSENS_POWER_UP.
  - Wait for a while and then configure the register SENS_TSENS_DUMP_OUT. The output value gradually approaches the actual temperature linearly as the measurement time increases.
  - Wait for SENS_TSENS_READY and read the conversion result from SENS_TSENS_OUT.
- Started by ULP-FSM:
  - Clear the register SENS_TSENS_POWER_UP_FORCE.
  - ULP-FSM has a built-in instruction for temperature sampling. Executing the instruction can easily complete temperature sampling, see Section 1.5 *ULP-FSM*.

32.4.4 Temperature Conversion

The actual temperature can be obtained by converting the output of temperature sensor via the following formula:

$$T(°C) = 0.4386 * VALUE – 27.88 * offset – 20.52$$

VALUE in the formula is the output of the temperature sensor, and the offset is determined by the temperature offset TSENS_DAC. Users can set I²C register I2C_SARADC_TSENS_ADC to configure TSENS_DAC according to the actual environment (the temperature range).

Table 195: Temperature Offset

TSENS_DAC	Temperature Offset	Measurement Range (°C)
5	-2	50 ~ 125
13 or 7	-1	20 ~ 100
15	0	-10 ~ 80
11 or 14	1	-30 ~ 50
10	2	-40 ~ 20

32.5 Interrupts

- APB_SARADC_ADC1_THRES_INT: Triggered when ADC1_DATA reaches the condition set in APB_SARADC_ADC1_THRES.
- APB_SARADC_ADC2_THRES_INT: Triggered when ADC2_DATA reaches the condition set in APB_SARADC_ADC2_THRES.
- APB_SARADC_ADC1_DONE_INT: Triggered when DIG ADC1 completes data conversion.
- APB_SARADC_ADC2_DONE_INT: Triggered when DIG ADC2 completes data conversion.

For the interrupts routed to ULP-RISC-V, please refer to Section 1.6.3 *ULP-RISC-V Interrupts* in Chapter 1 *ULP Coprocessor (ULP)*.

32.6 Base Address

Users can access on-chip sensor, SAR ADCs, and DACs registers with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 3: *System and Memory*.

Table 196: On-Chip Sensor, SAR ADCs, and DACs Base Addresses

Module	Bus to Access Peripheral	Base Address
SENSOR (RTC_PERI)	PeriBUS1	0x3F408800
	PeriBUS2	0x60008800
SENSOR (DIGITAL)	PeriBUS1	0x3F440000
	PeriBUS2	0x60040000

Wherein:

- SENSOR (RTC_PERI) represents the registers, which will be reset due to the power down of RTC_PERI domain. See Chapter 9 *Low-Power Management (RTC_CNTL)*.
- SENSOR (DIG_PERI) represents the registers, which will be reset due to the power down of digital domain. See Chapter 9 *Low-Power Management (RTC_CNTL)*.

32.7 Register Summary

The address in the table is the address offset (relative address) relative to the base address. Please refer to Section 32.6 for information about the base address.

32.7.1 SENSOR (RTC_PERI) Register Summary

Name	Description	Address	Access
RTC ADC1 Controller Registers			
SENS_SAR_READER1_CTRL_REG	RTC ADC1 data and sampling control	0x0000	R/W
SENS_SAR_MEAS1_CTRL1_REG	Configure RTC ADC1 controller	0x0008	R/W
SENS_SAR_MEAS1_CTRL2_REG	Control RTC ADC1 conversion and status	0x000C	varies
SENS_SAR_MEAS1_MUX_REG	Select the controller for SAR ADC1	0x0010	R/W
SAR ADC Attention Registers			
SENS_SAR_ATTEN1_REG	Configure SAR ADC1 attenuation	0x0014	R/W
SENS_SAR_ATTEN2_REG	Configure SAR ADC2 attenuation	0x0038	R/W
RTC ADC AMP Control Register			
SENS_SAR_AMP_CTRL3_REG	AMP control register	0x0020	R/W
RTC ADC2 Controller Registers			
SENS_SAR_READER2_CTRL_REG	RTC ADC2 data and sampling control	0x0024	R/W
SENS_SAR_MEAS2_CTRL2_REG	Control RTC ADC2 conversion and status	0x0030	varies
SENS_SAR_MEAS2_MUX_REG	Select the controller for SAR ADC2	0x0034	R/W
Temperature Sensor Registers			
SENS_SAR_TSENS_CTRL_REG	Temperature sensor data control	0x0050	varies
SENS_SAR_TSENS_CTRL2_REG	Temperature sensor control	0x0054	R/W
DAC Registers			
SENS_SAR_DAC_CTRL1_REG	DAC control	0x011C	R/W
SENS_SAR_DAC_CTRL2_REG	DAC output control	0x0120	R/W
IO MUX Clock Gate			
SENS_SAR_IO_MUX_CONF_REG	Configure and reset IO MUX	0x0144	R/W

32.7.2 SENSOR (DIG_PERI) Register Summary

Name	Description	Address	Access
DIG ADC Controller Registers			
APB_SARADC_CTRL_REG	DIG ADC common configuration	0x0000	R/W
APB_SARADC_CTRL2_REG	DIG ADC common configuration	0x0004	R/W
APB_SARADC_CLKM_CONF_REG	Configure DIG ADC clock	0x005C	R/W
DIG ADC1 Pattern Table Registers			
APB_SARADC_SAR1_PATT_TAB1_REG	Item 0 ~ 3 for pattern table 1 (each item one byte)	0x0018	R/W
APB_SARADC_SAR1_PATT_TAB2_REG	Item 4 ~ 7 for pattern table 1 (each item one byte)	0x001C	R/W
APB_SARADC_SAR1_PATT_TAB3_REG	Item 8 ~ 11 for pattern table 1 (each item one byte)	0x0020	R/W
APB_SARADC_SAR1_PATT_TAB4_REG	Item 12 ~ 15 for pattern table 1 (each item one byte)	0x0024	R/W
DIG ADC2 Pattern Table Registers			
APB_SARADC_SAR2_PATT_TAB1_REG	Item 0 ~ 3 for pattern table 2 (each item one byte)	0x0028	R/W
APB_SARADC_SAR2_PATT_TAB2_REG	Item 4 ~ 7 for pattern table 2 (each item one byte)	0x002C	R/W
APB_SARADC_SAR2_PATT_TAB3_REG	Item 8 ~ 11 for pattern table 2 (each item one byte)	0x0030	R/W
APB_SARADC_SAR2_PATT_TAB4_REG	Item 12 ~ 15 for pattern table 2 (each item one byte)	0x0034	R/W

Name	Description	Address	Access
DIG ADC2 Arbiter Register			
APB_SARADC_ARB_CTRL_REG	Configure the settings of DIG ADC2 arbiter	0x0038	R/W
DIG ADC2 Filter Registers			
APB_SARADC_FILTER_CTRL_REG	Configure the settings of DIG ADC2 filter	0x003C	R/W
APB_SARADC_FILTER_STATUS_REG	Data status of DIG ADC2 filter	0x0040	RO
DIG ADC2 Threshold Register			
APB_SARADC_THRES_CTRL_REG	Configure monitor threshold for DIG ADC2	0x0044	R/W
DIG ADC Interrupt Registers			
APB_SARADC_INT_ENA_REG	Enable DIG ADC interrupts	0x0048	R/W
APB_SARADC_INT_RAW_REG	DIG ADC interrupt raw bits	0x004C	RO
APB_SARADC_INT_ST_REG	DIG ADC interrupt status	0x0050	RO
APB_SARADC_INT_CLR_REG	Clear DIG ADC interrupts	0x0054	WO
DMA Register for DIG ADC			
APB_SARADC_DMA_CONF_REG	Configure digital ADC DMA path	0x0058	R/W
DAC Control Register			
APB_SARADC_DAC_CTRL_REG	Configure DAC settings	0x0060	R/W
Version Register			
APB_SARADC_CTRL_DATE_REG	Version control register	0x03FC	R/W

32.8 Register

32.8.1 SENSOR (RTC_PERI) Registers

Register 32.1: SENS_SAR_READER1_CTRL_REG (0x0000)

(reserved)	SENS_SAR1_INT_EN	SENS_SAR1_DATA_INV	(reserved)	SENS_SAR1_CLK_DIV	
31 30	29	28	27 8	7 0	
0 0	1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2	Reset

SENS_SAR1_CLK_DIV Clock divider. (R/W)

SENS_SAR1_DATA_INV Invert SAR ADC1 data. (R/W)

SENS_SAR1_INT_EN Enable SAR ADC1 to send out interrupt. (R/W)

Register 32.2: SENS_SAR_MEAS1_CTRL1_REG (0x0008)

31–24	23	22	21–0	
(reserved)	SENS_RTC_SARADC_CLKGATE_EN	SENS_RTC_SARADC_RESET	(reserved)	
0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_RTC_SARADC_RESET SAR ADC software reset. (R/W)

SENS_RTC_SARADC_CLKGATE_EN Enable bit of SAR ADC clock gate. (R/W)

Register 32.3: SENS_SAR_MEAS1_CTRL2_REG (0x000C)

31	30–19	18	17	16	15–0	
SENS_SAR1_EN_PAD_FORCE	SENS_SAR1_EN_PAD	SENS_MEAS1_START_FORCE	SENS_MEAS1_START_SAR	SENS_MEAS1_DONE_SAR	SENS_MEAS1_DATA_SAR	
0	0	0	0	0	0	Reset

SENS_MEAS1_DATA_SAR SAR ADC1 data. (RO)

SENS_MEAS1_DONE_SAR Indicate SAR ADC1 conversion is done. (RO)

SENS_MEAS1_START_SAR SAR ADC1 controller (in RTC) starts conversion, active only when SENS_MEAS1_START_FORCE = 1. (R/W)

SENS_MEAS1_START_FORCE 1: SAR ADC1 controller (in RTC) is started by software. 0: SAR ADC1 controller is started by ULP coprocessor. (R/W)

SENS_SAR1_EN_PAD SAR ADC1 pad enable bitmap, active only when SENS_SAR1_EN_PAD_FORCE = 1. (R/W)

SENS_SAR1_EN_PAD_FORCE 1: SAR ADC1 pad enable bitmap is controlled by software. 0: SAR ADC1 pad enable bitmap is controlled by ULP coprocessor. (R/W)

Register 32.4: SENS_SAR_MEAS1_MUX_REG (0x0010)

SENS_SAR1_DIG_FORCE	(reserved)	
31	30 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_SAR1_DIG_FORCE 1: SAR ADC1 controlled by DIG ADC1 CTRL. (R/W)

Register 32.5: SENS_SAR_ATTEN1_REG (0x0014)

SENS_SAR1_ATTEN	
31 ... 0	
0xffffffff	Reset

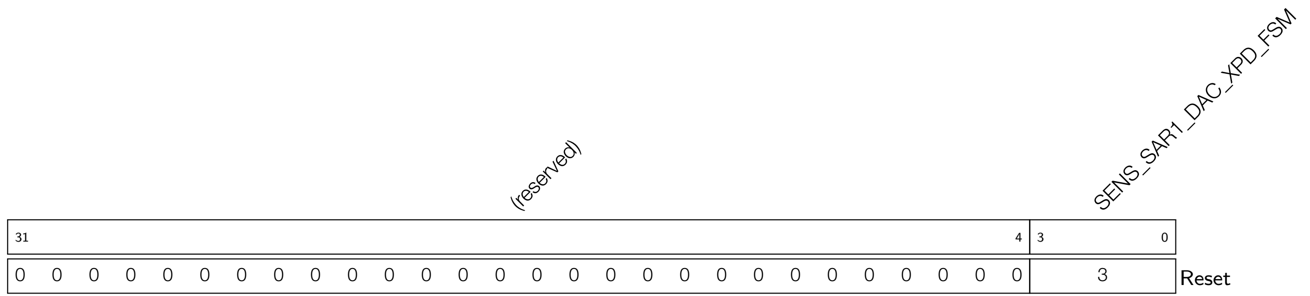
SENS_SAR1_ATTEN 2-bit attenuation for each pad. [1:0] is used for channel 0, [3:2] is used for channel 1, etc. (R/W)

Register 32.6: SENS_SAR_ATTEN2_REG (0x0038)

SENS_SAR2_ATTEN	
31 ... 0	
0xffffffff	Reset

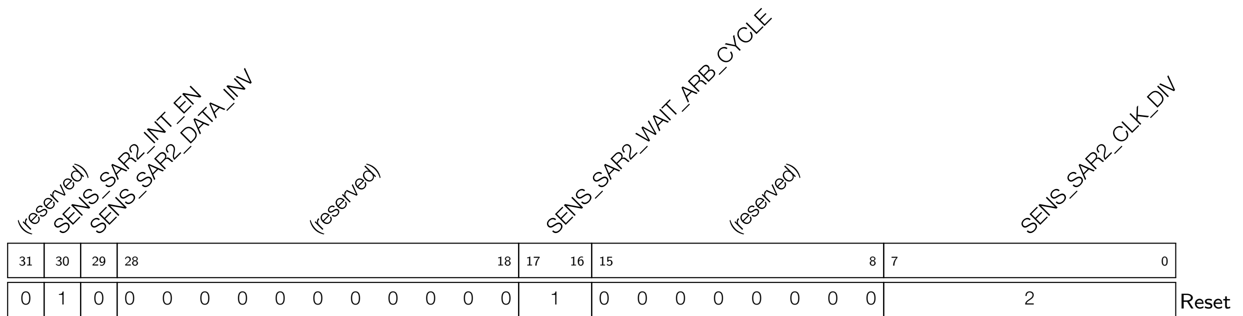
SENS_SAR2_ATTEN 2-bit attenuation for each pad. [1:0] is used for channel 0, [3:2] is used for channel 1, etc. (R/W)

Register 32.7: SENS_SAR_AMP_CTRL3_REG (0x0020)



SENS_SAR1_DAC_XPD_FSM Control of DAC. 4’b0010: disable DAC. 4’b0000: power up DAC by FSM. 4’b0011: power up DAC by software. (R/W)

Register 32.8: SENS_SAR_READER2_CTRL_REG (0x0024)



SENS_SAR2_CLK_DIV Clock divider. (R/W)

SENS_SAR2_WAIT_ARB_CYCLE Wait arbiter stable after sar_done. (R/W)

SENS_SAR2_DATA_INV Invert SAR ADC2 data. (R/W)

SENS_SAR2_INT_EN Enable SAR ADC2 to send out interrupt. (R/W)

Register 32.9: SENS_SAR_MEAS2_CTRL2_REG (0x0030)

SENS_SAR2_EN_PAD_FORCE	SENS_SAR2_EN_PAD	SENS_MEAS2_START_FORCE	SENS_MEAS2_START_SAR	SENS_MEAS2_DONE_SAR	SENS_MEAS2_DATA_SAR	
31	30 19	18	17	16	15 0	
0	0	0	0	0	0	Reset

SENS_MEAS2_DATA_SAR SAR ADC2 data. (RO)

SENS_MEAS2_DONE_SAR Indicate SAR ADC2 conversion is done. (RO)

SENS_MEAS2_START_SAR SAR ADC2 controller (in RTC) starts conversion, active only when SENS_MEAS2_START_FORCE = 1. (R/W)

SENS_MEAS2_START_FORCE 1: SAR ADC2 controller (in RTC) is started by software. 0: SAR ADC2 controller is started by ULP coprocessor. (R/W)

SENS_SAR2_EN_PAD SAR ADC2 pad enable bitmap, active only whenSENS_SAR2_EN_PAD_FORCE = 1. (R/W)

SENS_SAR2_EN_PAD_FORCE 1: SAR ADC2 pad enable bitmap is controlled by software. 0: SAR ADC2 pad enable bitmap is controlled by ULP coprocessor. (R/W)

Register 32.10: SENS_SAR_MEAS2_MUX_REG (0x0034)

SENS_SAR2_RTC_FORCE	SENS_SAR2_PWDET_CCT	(reserved)	
31	30 28	27 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_SAR2_PWDET_CCT SAR2_PWDET_CCT, PA power detector capacitance tuning. (R/W)

SENS_SAR2_RTC_FORCE In sleep, force to use RTC to control ADC. (R/W)

Register 32.11: SENS_SAR_TSENS_CTRL_REG (0x0050)

Bits	31–25	24	23	22	21–14	13	12	11–9	8	7–0
Field	(reserved)	SENS_TSENS_DUMP_OUT	SENS_TSENS_POWER_UP_FORCE	SENS_TSENS_POWER_UP	SENS_TSENS_CLK_DIV	SENS_TSENS_IN_INV	SENS_TSENS_INT_EN	(reserved)	SENS_TSENS_READY	SENS_TSENS_OUT
Reset	0 0 0 0 0 0 0	0	0	0	6	0	1	0 0 0	0	0x0

SENS_TSENS_OUT Temperature sensor data out. (RO)

SENS_TSENS_READY Indicate temperature sensor out ready. (RO)

SENS_TSENS_INT_EN Enable temperature sensor to send out interrupt. (R/W)

SENS_TSENS_IN_INV Invert temperature sensor data. (R/W)

SENS_TSENS_CLK_DIV Temperature sensor clock divider. (R/W)

SENS_TSENS_POWER_UP Temperature sensor power up. (R/W)

SENS_TSENS_POWER_UP_FORCE 1: dump out and power up controlled by software. 0: by FSM. (R/W)

SENS_TSENS_DUMP_OUT Temperature sensor dump out only active when SENS_TSENS_POWER_UP_FORCE = 1. (R/W)

Register 32.12: SENS_SAR_TSENS_CTRL2_REG (0x0054)

Bits	31–17	16	15	14–0
Field	(reserved)	SENS_TSENS_RESET	SENS_TSENS_CLKGATE_EN	(reserved)
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SENS_TSENS_CLKGATE_EN Enable temperature sensor clock. (R/W)

SENS_TSENS_RESET Reset temperature sensor. (R/W)

Register 32.13: SENS_SAR_DAC_CTRL1_REG (0x011C)

(reserved)	SENS_DAC_CLKGATE_EN	SENS_DAC_RESET	SENS_DAC_CLK_INV	SENS_DAC_CLK_FORCE_HIGH	SENS_DAC_CLK_FORCE_LOW	SENS_DAC_DIG_FORCE	(reserved)	SENS_SW_TONE_EN	SENS_SW_FSTEP	
31 28	27	26	25	24	23	22	21 17	16	15 0	
0 0 0 0	0	0	0	0	0	0	0 0 0 0 0	0	0	Reset

SENS_SW_FSTEP Frequency step for CW generator can be used to adjust the frequency. (R/W)

SENS_SW_TONE_EN 0: disable CW generator. 1: enable CW generator. (R/W)

SENS_DAC_DIG_FORCE 0: DAC1 and DAC2 do not use DMA. 1: DAC1 and DAC2 use DMA. (R/W)

SENS_DAC_CLK_FORCE_LOW 1: force PDAC_CLK to low. (R/W)

SENS_DAC_CLK_FORCE_HIGH 1: force PDAC_CLK to high. (R/W)

SENS_DAC_CLK_INV 1: invert PDAC_CLK. (R/W)

SENS_DAC_RESET Reset DAC by software. (R/W)

SENS_DAC_CLKGATE_EN DAC clock gate enable bit. (R/W)

Register 32.14: SENS_SAR_DAC_CTRL2_REG (0x0120)

(reserved)	SENS_DAC_CW_EN2	SENS_DAC_CW_EN1	SENS_DAC_INV2	SENS_DAC_INV1	SENS_DAC_SCALE2	SENS_DAC_SCALE1	SENS_DAC_DC2	SENS_DAC_DC1	
31–26	25	24	23–22	21–20	19–18	17–16	15–8	7–0	
0 0 0 0 0 0	1	1	0	0	0	0	0	0	Reset

SENS_DAC_DC1 DC offset for DAC1 CW generator. (R/W)

SENS_DAC_DC2 DC offset for DAC2 CW generator. (R/W)

SENS_DAC_SCALE1 DAC1 scaling. 00: no scale. 01: scale to 1/2. 10: scale to 1/4. 11: scale to 1/8. (R/W)

SENS_DAC_SCALE2 DAC2 scaling. 00: no scale. 01: scale to 1/2. 10: scale to 1/4. 11: scale to 1/8. (R/W)

SENS_DAC_INV1 Invert DAC1. 00: do not invert any bits. 01: invert all bits. 10: invert MSB. 11: invert all bits except MSB. (R/W)

SENS_DAC_INV2 Invert DAC2. 00: do not invert any bits. 01: invert all bits. 10: invert MSB. 11: invert all bits except MSB. (R/W)

SENS_DAC_CW_EN1 1: select CW generator as source for PDAC1_DAC. 0: select register RTCIO_PDAC1_DAC as source for PDAC1_DAC. (R/W)

SENS_DAC_CW_EN2 1: select CW generator as source for PDAC2_DAC. 0: select register RTCIO_PDAC2_DAC as source for PDAC2_DAC. (R/W)

Register 32.15: SENS_SAR_IO_MUX_CONF_REG (0x0144)

SENS_IOMUX_CLK_GATE_EN	SENS_IOMUX_RESET	(reserved)	
31	30	29–0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_IOMUX_RESET Reset IO MUX by software (R/W)

SENS_IOMUX_CLK_GATE_EN IO MUX clock gate enable bit (R/W)

32.8.2 SENSOR (DIG_PERI) Registers

Register 32.16: APB_SARADC_CTRL_REG (0x0000)

31–30	29	28–27	26	25	24	23	22–19	18–15	14–7	6	5	4–3	2	1	0	
APB_SARADC_WAIT_ARB_CYCLE	(reserved)	APB_SARADC_XPD_SAR_FORCE	(reserved)	APB_SARADC_DATA_SAR_SEL	APB_SARADC_SAR2_PATT_P_CLEAR	APB_SARADC_SAR1_PATT_P_CLEAR	APB_SARADC_SAR2_PATT_LEN	APB_SARADC_SAR1_PATT_LEN	APB_SARADC_SAR_CLK_DIV	APB_SARADC_SAR_CLK_GATED	APB_SARADC_SAR_SEL	APB_SARADC_WORK_MODE	(reserved)	APB_SARADC_START	APB_SARADC_START_FORCE	
1	0	0	0	0	0	0	15	15	4	1	0	0	0	0	0	Reset

APB_SARADC_START_FORCE 0: select FSM to start SAR ADC. 1: select software to start SAR ADC. (R/W)

APB_SARADC_START Start SAR ADC by software. (R/W)

APB_SARADC_WORK_MODE 0: single-channel scan mode. 1: double-channel scan mode. 2: alternate-channel scan mode. (R/W)

APB_SARADC_SAR_SEL 0: select SAR ADC1. 1: select SAR ADC2, only work for single-channel scan mode. (R/W)

APB_SARADC_SAR_CLK_GATED SAR clock gate enable bit. (R/W)

APB_SARADC_SAR_CLK_DIV SAR clock divider. (R/W)

APB_SARADC_SAR1_PATT_LEN 0 ~ 15 means length 1 ~ 16. (R/W)

APB_SARADC_SAR2_PATT_LEN 0 ~ 15 means length 1 ~ 16. (R/W)

APB_SARADC_SAR1_PATT_P_CLEAR Clear the pointer of pattern table for DIG ADC1 CTRL. (R/W)

APB_SARADC_SAR2_PATT_P_CLEAR Clear the pointer of pattern table for DIG ADC2 CTRL. (R/W)

APB_SARADC_DATA_SAR_SEL 1: sar_sel will be coded to the MSB of the 16-bit output data, in this case the resolution should not be larger than 11 bits. (R/W)

APB_SARADC_XPD_SAR_FORCE Force option to xpd sar blocks. (R/W)

APB_SARADC_WAIT_ARB_CYCLE Wait arbit signal stable after sar_done. (R/W)

Register 32.17: APB_SARADC_CTRL2_REG (0x0004)

(reserved)	APB_SARADC_TIMER_EN	APB_SARADC_TIMER_TARGET	(reserved)	APB_SARADC_SAR2_INV	APB_SARADC_SAR1_INV	APB_SARADC_MAX_MEAS_NUM	APB_SARADC_MEAS_NUM_LIMIT	
31 25	24	23 12	11	10	9	8 1	0	
0 0 0 0 0 0 0	0	10	0	0	0	255	0	Reset

APB_SARADC_MEAS_NUM_LIMIT Enable limit times of SAR ADC sample. (R/W)

APB_SARADC_MAX_MEAS_NUM Set maximum conversion number. (R/W)

APB_SARADC_SAR1_INV 1: data to DIG ADC1 CTRL is inverted, otherwise not. (R/W)

APB_SARADC_SAR2_INV 1: data to DIG ADC2 CTRL is inverted, otherwise not. (R/W)

APB_SARADC_TIMER_TARGET Set SAR ADC timer target. (R/W)

APB_SARADC_TIMER_EN Enable SAR ADC timer trigger. (R/W)

Register 32.18: APB_SARADC_CLKM_CONF_REG (0x005C)

(reserved)	APB_SARADC_CLK_SEL	(reserved)	APB_SARADC_CLKM_DIV_A	APB_SARADC_CLKM_DIV_B	APB_SARADC_CLKM_DIV_NUM	
31 23	22 21	20	19 14	13 8	7 0	
0 0 0 0 0 0 0 0 0	0	0	0x0	0x0	4	Reset

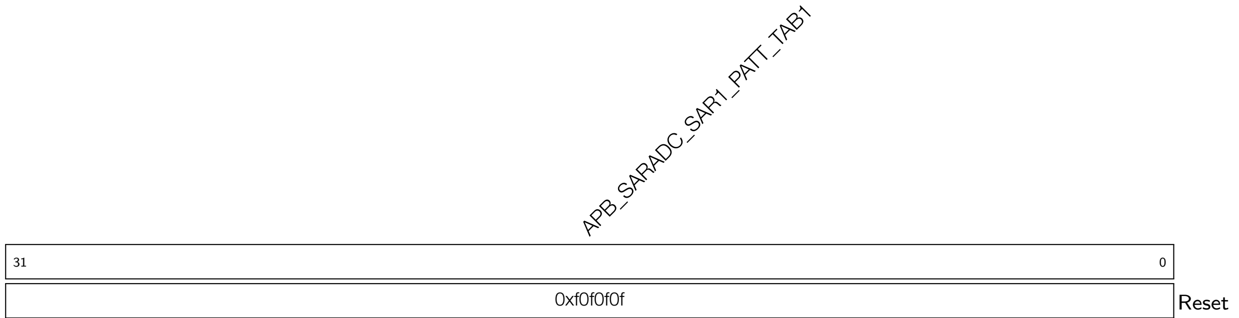
APB_SARADC_CLKM_DIV_NUM Integral DIG_ADC clock divider value (R/W)

APB_SARADC_CLKM_DIV_B Fractional clock divider numerator value (R/W)

APB_SARADC_CLKM_DIV_A Fractional clock divider denominator value (R/W)

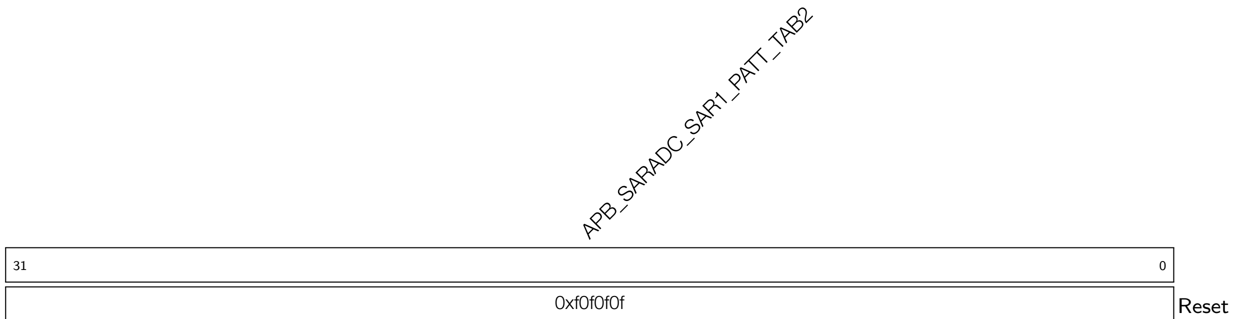
APB_SARADC_CLK_SEL 1: select APLL. 2: select APB_CLK. Other values: disable clock. (R/W)

Register 32.19: APB_SARADC_SAR1_PATT_TAB1_REG (0x0018)



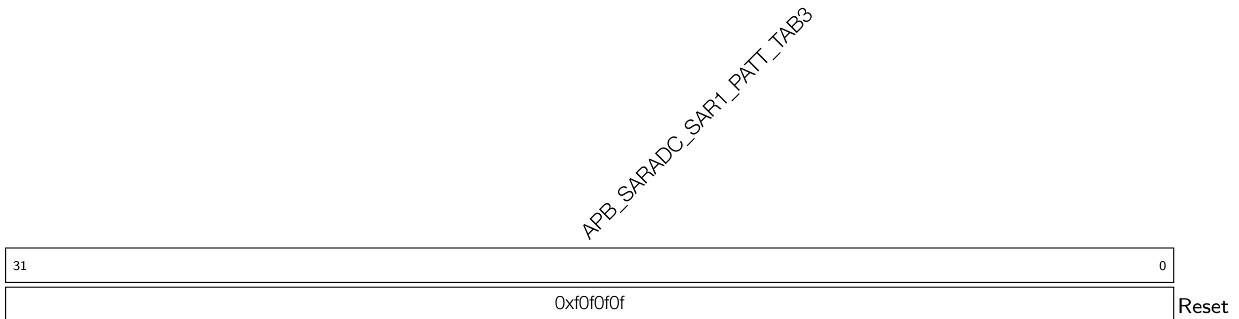
APB_SARADC_SAR1_PATT_TAB1 Item 0 ~ 3 for pattern table 1 (each item one byte). (R/W)

Register 32.20: APB_SARADC_SAR1_PATT_TAB2_REG (0x001C)



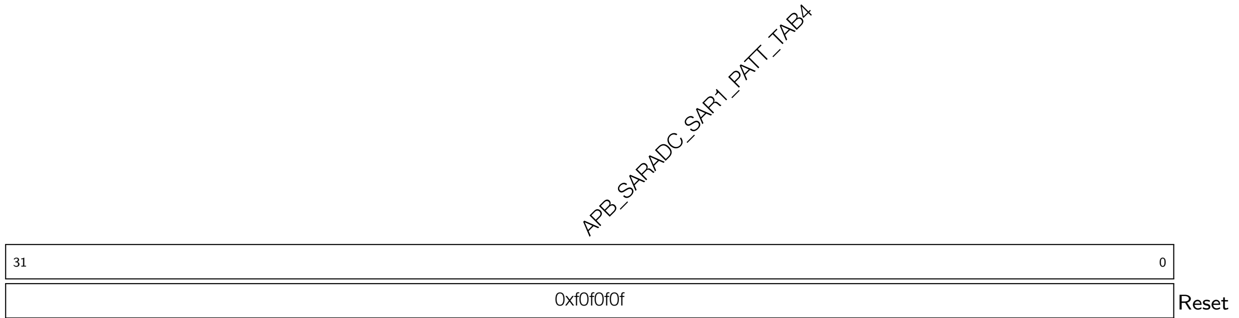
APB_SARADC_SAR1_PATT_TAB2 Item 4 ~ 7 for pattern table 1 (each item one byte). (R/W)

Register 32.21: APB_SARADC_SAR1_PATT_TAB3_REG (0x0020)



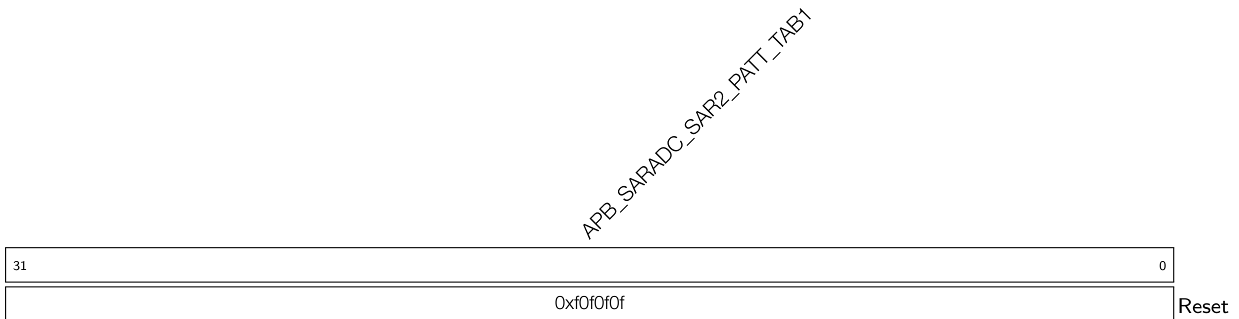
APB_SARADC_SAR1_PATT_TAB3 Item 8 ~ 11 for pattern table 1 (each item one byte). (R/W)

Register 32.22: APB_SARADC_SAR1_PATT_TAB4_REG (0x0024)



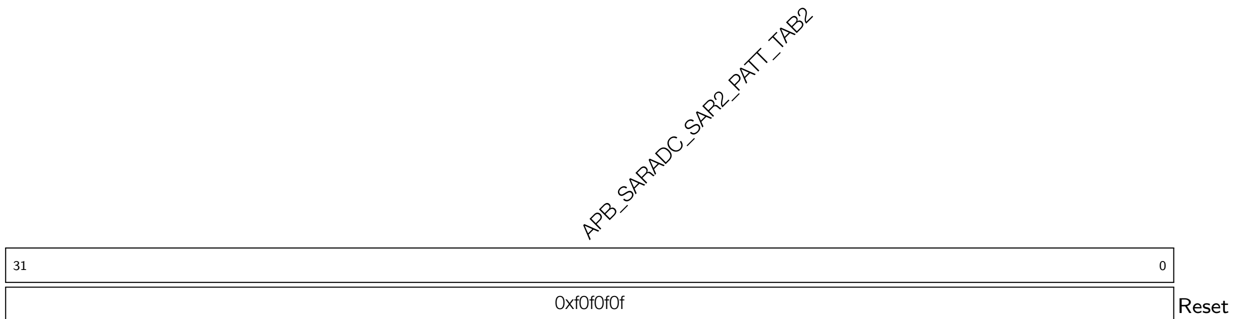
APB_SARADC_SAR1_PATT_TAB4 Item 12 ~ 15 for pattern table 1 (each item one byte). (R/W)

Register 32.23: APB_SARADC_SAR2_PATT_TAB1_REG (0x0028)



APB_SARADC_SAR2_PATT_TAB1 Item 0 ~ 3 for pattern table 2 (each item one byte). (R/W)

Register 32.24: APB_SARADC_SAR2_PATT_TAB2_REG (0x002C)



APB_SARADC_SAR2_PATT_TAB2 Item 4 ~ 7 for pattern table 2 (each item one byte). (R/W)

Register 32.25: APB_SARADC_SAR2_PATT_TAB3_REG (0x0030)

APB_SARADC_SAR2_PATT_TAB3

APB_SARADC_SAR2_PATT_TAB3 Item 8 ~ 11 for pattern table 2 (each item one byte). (R/W)

Register 32.26: APB_SARADC_SAR2_PATT_TAB4_REG (0x0034)

APB_SARADC_SAR2_PATT_TAB4

APB_SARADC_SAR2_PATT_TAB4 Item 12 ~ 15 for pattern table 2 (each item one byte). (R/W)

Register 32.27: APB_SARADC_ARB_CTRL_REG (0x0038)

Bits	31–13	12	11–10	9–8	7–6	5	4	3	2	1–0
Field	(reserved)	APB_SARADC_ADC_ARB_FIX_PRIORITY	APB_SARADC_ADC_ARB_WIFI_PRIORITY	APB_SARADC_ADC_ARB_RTC_PRIORITY	APB_SARADC_ADC_ARB_APB_PRIORITY	APB_SARADC_ADC_ARB_GRANT_FORCE	APB_SARADC_ADC_ARB_WIFI_FORCE	APB_SARADC_ADC_ARB_RTC_FORCE	APB_SARADC_ADC_ARB_APB_FORCE	(reserved)
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	2	1	0	0	0	0	0	0 0

APB_SARADC_ADC_ARB_APB_FORCE ADC2 arbiter forces to enable DIG ADC2 CTRL. (R/W)

APB_SARADC_ADC_ARB_RTC_FORCE ADC2 arbiter forces to enable RTC ADC2 CTRL. (R/W)

APB_SARADC_ADC_ARB_WIFI_FORCE ADC2 arbiter forces to enable PWDET/PKDET CTRL. (R/W)

APB_SARADC_ADC_ARB_GRANT_FORCE ADC2 arbiter force grant. (R/W)

APB_SARADC_ADC_ARB_APB_PRIORITY Set DIG ADC2 CTRL priority. (R/W)

APB_SARADC_ADC_ARB_RTC_PRIORITY Set RTC ADC2 CTRL priority. (R/W)

APB_SARADC_ADC_ARB_WIFI_PRIORITY Set PWDET/PKDET CTRL priority. (R/W)

APB_SARADC_ADC_ARB_FIX_PRIORITY ADC2 arbiter uses fixed priority. (R/W)

Register 32.28: APB_SARADC_FILTER_CTRL_REG (0x003C)

31	30	29 … 23	22 … 16	15 … 2	1	0	
APB_SARADC_ADC1_FILTER_EN	APB_SARADC_ADC2_FILTER_EN	APB_SARADC_ADC1_FILTER_FACTOR	APB_SARADC_ADC2_FILTER_FACTOR	(reserved)	APB_SARADC_ADC1_FILTER_RESET	APB_SARADC_ADC2_FILTER_RESET	
0	0	64	64	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

APB_SARADC_ADC2_FILTER_RESET Reset ADC2 filter. (R/W)

APB_SARADC_ADC1_FILTER_RESET Reset ADC1 filter. (R/W)

APB_SARADC_ADC2_FILTER_FACTOR Set filter factor for DIG ADC2 CRTL. (R/W)

APB_SARADC_ADC1_FILTER_FACTOR Set filter factor for DIG ADC1 CRTL. (R/W)

APB_SARADC_ADC2_FILTER_EN Enable DIG ADC2 CRTL filter. (R/W)

APB_SARADC_ADC1_FILTER_EN Enable DIG ADC1 CRTL filter. (R/W)

Register 32.29: APB_SARADC_FILTER_STATUS_REG (0x0040)

31 … 16	15 … 0	
APB_SARADC_ADC1_FILTER_DATA	APB_SARADC_ADC2_FILTER_DATA	
0	0	Reset

APB_SARADC_ADC2_FILTER_DATA ADC2 filter data. (RO)

APB_SARADC_ADC1_FILTER_DATA ADC1 filter data. (RO)

Register 32.30: APB_SARADC_THRES_CTRL_REG (0x0044)

31	30	29 … 17	16 … 4	3	2	1	0	
APB_SARADC_ADC1_THRES_EN	APB_SARADC_ADC2_THRES_EN	APB_SARADC_ADC1_THRES	APB_SARADC_ADC2_THRES	APB_SARADC_ADC1_THRES_MODE	APB_SARADC_ADC2_THRES_MODE	(reserved)	APB_SARADC_CLK_EN	
0	0	0	0	0	0	0	0	Reset

APB_SARADC_CLK_EN Clock gate enable. (R/W)

APB_SARADC_ADC2_THRES_MODE 1: ADC_DATA > = threshold, generate interrupt. 0: ADC_DATA < threshold, generate interrupt. (R/W)

APB_SARADC_ADC1_THRES_MODE 1: ADC_DATA > = threshold, generate interrupt. 0: ADC_DATA < threshold, generate interrupt. (R/W)

APB_SARADC_ADC2_THRES ADC2 threshold. (R/W)

APB_SARADC_ADC1_THRES ADC1 threshold. (R/W)

APB_SARADC_ADC2_THRES_EN Enable ADC2 threshold monitor. (R/W)

APB_SARADC_ADC1_THRES_EN Enable ADC1 threshold monitor. (R/W)

Register 32.31: APB_SARADC_INT_ENA_REG (0x0048)

APB_SARADC_ADC1_DONE_INT_ENA	APB_SARADC_ADC2_DONE_INT_ENA	APB_SARADC_ADC1_THRES_INT_ENA	APB_SARADC_ADC2_THRES_INT_ENA	(reserved)	
31	30	29	28	27 0	
0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

APB_SARADC_ADC2_THRES_INT_ENA Enable bit of APB_SARADC_ADC2_THRES_INT interrupt. (R/W)

APB_SARADC_ADC1_THRES_INT_ENA Enable bit of APB_SARADC_ADC1_THRES_INT interrupt. (R/W)

APB_SARADC_ADC2_DONE_INT_ENA Enable bit of APB_SARADC_ADC2_DONE_INT interrupt. (R/W)

APB_SARADC_ADC1_DONE_INT_ENA Enable bit of APB_SARADC_ADC1_DONE_INT interrupt. (R/W)

Register 32.32: APB_SARADC_INT_RAW_REG (0x004C)

APB_SARADC_ADC1_DONE_INT_RAW	APB_SARADC_ADC2_DONE_INT_RAW	APB_SARADC_ADC1_THRES_INT_RAW	APB_SARADC_ADC2_THRES_INT_RAW	(reserved)	
31	30	29	28	27 0	
0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

APB_SARADC_ADC2_THRES_INT_RAW Raw bit of APB_SARADC_ADC2_THRES_INT interrupt. (RO)

APB_SARADC_ADC1_THRES_INT_RAW Raw bit of APB_SARADC_ADC1_THRES_INT interrupt. (RO)

APB_SARADC_ADC2_DONE_INT_RAW Raw bit of APB_SARADC_ADC2_DONE_INT interrupt. (RO)

APB_SARADC_ADC1_DONE_INT_RAW Raw bit of APB_SARADC_ADC1_DONE_INT interrupt. (RO)

Register 32.33: APB_SARADC_INT_ST_REG (0x0050)

31	30	29	28	27 ... 0	
APB_SARADC_ADC1_DONE_INT_ST	APB_SARADC_ADC2_DONE_INT_ST	APB_SARADC_ADC1_THRES_INT_ST	APB_SARADC_ADC2_THRES_INT_ST	(reserved)	
0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

APB_SARADC_ADC2_THRES_INT_ST Status of APB_SARADC_ADC2_THRES_INT interrupt. (RO)

APB_SARADC_ADC1_THRES_INT_ST Status of APB_SARADC_ADC1_THRES_INT interrupt. (RO)

APB_SARADC_ADC2_DONE_INT_ST Status of APB_SARADC_ADC2_DONE_INT interrupt. (RO)

APB_SARADC_ADC1_DONE_INT_ST Status of APB_SARADC_ADC1_DONE_INT interrupt. (RO)

Register 32.34: APB_SARADC_INT_CLR_REG (0x0054)

31	30	29	28	27 ... 0	
APB_SARADC_ADC1_DONE_INT_CLR	APB_SARADC_ADC2_DONE_INT_CLR	APB_SARADC_ADC1_THRES_INT_CLR	APB_SARADC_ADC2_THRES_INT_CLR	(reserved)	
0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

APB_SARADC_ADC2_THRES_INT_CLR Clear bit of APB_SARADC_ADC2_THRES_INT interrupt. (WO)

APB_SARADC_ADC1_THRES_INT_CLR Clear bit of APB_SARADC_ADC1_THRES_INT interrupt. (WO)

APB_SARADC_ADC2_DONE_INT_CLR Clear bit of APB_SARADC_ADC2_DONE_INT interrupt. (WO)

APB_SARADC_ADC1_DONE_INT_CLR Clear bit of APB_SARADC_ADC1_DONE_INT interrupt. (WO)

Register 32.35: APB_SARADC_DMA_CONF_REG (0x0058)

APB_SARADC_APB_ADC_TRANS	APB_SARADC_APB_ADC_RESET_FSM	(reserved)	APB_SARADC_APB_ADC_EOF_NUM	
31	30	29 … 16	15 … 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	255	Reset

APB_SARADC_APB_ADC_EOF_NUM Generate dma_in_suc_eof when sample cnt = spi_eof_num. (R/W)

APB_SARADC_APB_ADC_RESET_FSM Reset DIG ADC CTRL status. (R/W)

APB_SARADC_APB_ADC_TRANS Set this bit, DIG ADC CTRL uses SPI DMA. (R/W)

Register 32.36: APB_SARADC_APB_DAC_CTRL_REG (0x0060)

(reserved)	APB_SARADC_APB_DAC_RST	APB_SARADC_DAC_RESET_FIFO	APB_SARADC_APB_DAC_TRANS	APB_SARADC_APB_DAC_ALTER_MODE	APB_SARADC_DAC_TIMER_EN	APB_SARADC_DAC_TIMER_TARGET	
31 … 17	16	15	14	13	12	11 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	1	0	100	Reset

APB_SARADC_DAC_TIMER_TARGET Set DAC timer target. (R/W)

APB_SARADC_DAC_TIMER_EN Enable read dac data. (R/W)

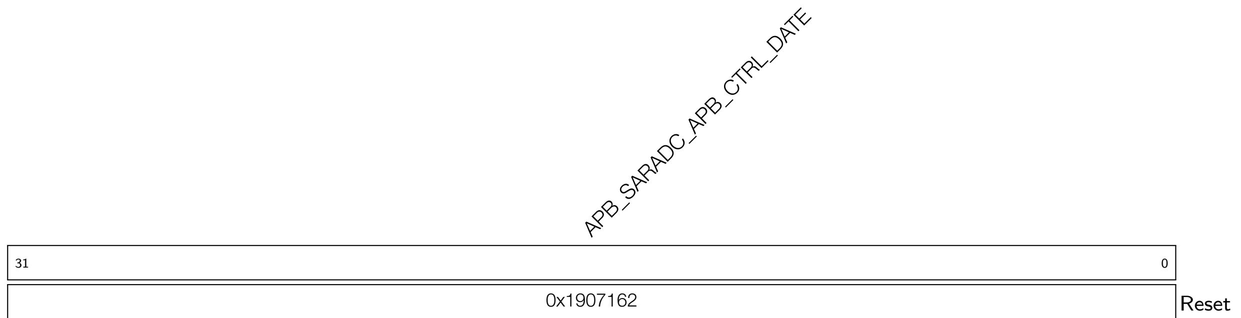
APB_SARADC_APB_DAC_ALTER_MODE Enable DAC alter mode. (R/W)

APB_SARADC_APB_DAC_TRANS Enable DMA_DAC. (R/W)

APB_SARADC_DAC_RESET_FIFO Reset DIG DAC FIFO. (R/W)

APB_SARADC_APB_DAC_RST Reset DIG DAC by software. (R/W)

Register 32.37: APB_SARADC_APB_CTRL_DATE_REG (0x03FC)



APB_SARADC_APB_CTRL_DATE Version control register (R/W)

Glossary

Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED PWM (Pulse Width Modulation) Controller
MMU	Memory Management Unit
PCNT	Pulse Count Controller
PERI	Peripheral
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
RTC	Real Time Controller. A group of circuits in SoC that keeps working in any chip mode and at any time.
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

Abbreviations for Registers

ISO	Isolation. When a module is power down, its output pins will be stuck in unknown state (some middle voltage). "ISO" registers will control to isolate its output pins to be a determined value, so it will not affect the status of other working modules which are not power down.
NMI	Non-maskable interrupt.
REG	Register.
R/W	Read/write. Software can read and write to these bits.
RO	Read-only. Software can only read these bits.
SYSREG	System Registers
WO	Write-only. Software can only write to these bits.

Revision History

Date	Version	Release notes
2021-06-11	V1.0	Added Glossary Optimized chapter order Updated the following chapters: • Updated the description of EFUSE_DIS_RTC_RAM_BOOT in Chapter 4 *eFuse Controller (eFuse)* • Fixed two typos in Chapter 9 *Low-Power Management (RTC_CNTL)* • Fixed a typo in register naming in Section 14.3.2.2 *External Memory Permission Controls* • Fixed a typo in the description *LEDC_TIMERx_CONF_REG (x: 0-3)* in Chapter 23 *UART Controller (UART)*; • Updated the base address of USB OTG in Section 28.5 *Base Address*
2020-12-28	V0.7	Added a new chapter 24 *SPI Controller (SPI)* Added register prefix to chapter names Updated the following chapters: • Updated the relationship between EDMA and Crypto DMA in Chapter 2 *DMA Controller (DMA)* • Updated Figure 9-1 in Chapter 9 *Low-Power Management (RTC_CNTL)* • Updated the description of register TIMG_RTCCALICFG_REG and register TIMG_RTCCALICFG1_REG in Chapter 11 *Timer Group (TIMG)* • Updated the description of version register in Chapter 19 *HMAC Accelerator (HMAC)* • Updated Figure 23-1 and description of UART_RXD_CNT_REG field in Chapter 23 *UART Controller (UART)* • Changed the name and trademark symbol of Chapter 29 *Two-wire Automotive Interface (TWAI)*
2020-08-12	V0.6	Added the following chapters: • Chapter 26 *I2S Controller (I2S)* • Chapter 28 *USB On-The-Go (USB)* Updated the following chapters: • Updated Figure 6-2 in Chapter 6 *Reset and Clock* • Fixed a typo in Chapter 20 *Digital Signature (DS)* • Added Section 22.4 *Programming Procedure* and updated some description in Chapter 22 *Random Number Generator (RNG)* • Updated baud rate calculation formulas in Section 23.3.4.2 *Baud Rate Detection* of Chapter 23 *UART Controller (UART)* • Updated the name of Chapter 29 *Two-wire Automotive Interface (TWAI)*

Date	Version	Release notes
2020-06-19	V0.5	Added Chapter 32 *On-Chip Sensor and Analog Signal Processing* Updated the following chapters: • Added information about Copy DMA and Crypto DMA, and updated information about Internal DMA and EMDA in Chapter 2 *DMA Controller (DMA)* • Added Section 12.4 *Super Watchdog* in Chapter 12 *Watchdog Timers (WDT)*
2020-05-21	V0.4	Added the following chapters: • Chapter 9 *Low-Power Management (RTC_CNTL)* • Chapter 29 *Two-wire Automotive Interface (TWAI)* Updated the following chapters: • Removed bit EFUSE_PGM_DATA1_REG[16] from the list of bits that are reserved and can only be used by software; added bit EFUSE_RPT4_RESERVED5 and EFUSE_RPT4_RESERVED5_ERR in Chapter 4 *eFuse Controller (eFuse)* • Added a new section 5.4 *Dedicated GPIO* in Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)* • Updated the base address in Chapter 22 *Random Number Generator (RNG)*
2020-04-01	V0.3	Added the following chapters: • Chapter 1 *ULP Coprocessor (ULP)* • Chapter 10 *System Timer (SYSTIMER)* • Chapter 13 *XTAL32K Watchdog Timer (XTWDT)* • Chapter 14 *Permission Control (PMS)* • Chapter 19 *HMAC Accelerator (HMAC)* Updated RTC_CLK frequency in Chapter 6 *Reset and Clock*
2020-01-20	V0.2	Added the following chapters: • Chapter 2 *DMA Controller (DMA)* • Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO_MUX)* • Chapter 23 *UART Controller (UART)* Updated the configurations of eFuse-programming and eFuse-read timing parameters in Chapter 4 *eFuse Controller (eFuse)*
2019-11-27	V0.1	Preliminary release

www.espressif.com

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

ALL THIRD PARTY'S INFORMATION IN THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES TO ITS AUTHENTICITY AND ACCURACY.

NO WARRANTY IS PROVIDED TO THIS DOCUMENT FOR ITS MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, NOR DOES ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2021 Espressif Systems (Shanghai) Co., Ltd. All rights reserved.

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Espressif:

ESP32-Korvo