### 42.9.6 SSC Transmit Frame Mode Register

**Name:** SSC_TFMR

**Address:** 0xF800401C (0), 0xFC00401C (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FSLEN_EXT | | | | – | – | – | FSEDGE |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| FSDEN | FSOS | | | FSLEN | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | DATNB | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MSBF | – | DATDEF | DATLEN | | | | |

This register can only be written if the WPEN bit is cleared in the SSC Write Protection Mode Register.

- **DATLEN: Data Length**

0: Forbidden value (1-bit data length not supported).

Any other value: The bit stream contains DATLEN + 1 data bits.

- **DATDEF: Data Default Value**

This bit defines the level driven on the TD pin while out of transmission. Note that if the pin is defined as multidrive by the PIO Controller, the pin is enabled only if the SCC TD output is 1.

- **MSBF: Most Significant Bit First**

0: The lowest significant bit of the data register is shifted out first in the bit stream.

1: The most significant bit of the data register is shifted out first in the bit stream.

- **DATNB: Data Number per Frame**

This field defines the number of data words to be transferred after each transfer start, which is equal to (DATNB + 1).

- **FSLEN: Transmit Frame Sync Length**

This field defines the length of the Transmit Frame Sync signal and the number of bits shifted out from the Transmit Sync Data Register if FSDEN is 1.

This field is used with FSLEN_EXT to determine the pulse length of the Transmit Frame Sync signal.

Pulse length is equal to FSLEN + (FSLEN_EXT × 16) + 1 Transmit Clock period.

- **FSOS: Transmit Frame Sync Output Selection**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | None, TF pin is an input |
| 1 | NEGATIVE | Negative Pulse, TF pin is an output |
| 2 | POSITIVE | Positive Pulse, TF pin is an output |
| 3 | LOW | Driven Low during data transfer |
| 4 | HIGH | Driven High during data transfer |
| 5 | TOGGLING | Toggling at each start of data transfer |

- **FSDEN: Frame Sync Data Enable**

0: The TD line is driven with the default value during the Transmit Frame Sync signal.

1: SSC_TSHR value is shifted out during the transmission of the Transmit Frame Sync signal.

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on frame sync will generate the interrupt TXSYN (Status Register).

| Value | Name | Description |
|---|---|---|
| 0 | POSITIVE | Positive Edge Detection |
| 1 | NEGATIVE | Negative Edge Detection |

- **FSLEN_EXT: FSLEN Field Extension**

Extends FSLEN field. For details, refer to FSLEN bit description above.

### 42.9.7 SSC Receive Holding Register

**Name:** SSC_RHR

**Address:** 0xF8004020 (0), 0xFC004020 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RDAT | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RDAT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RDAT | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RDAT | | | | | | | |

- **RDAT: Receive Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC_RFMR.

### 42.9.8 SSC Transmit Holding Register

**Name:** SSC_THR

**Address:** 0xF8004024 (0), 0xFC004024 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TDAT | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| TDAT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TDAT | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TDAT | | | | | | | |

- **TDAT: Transmit Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC_TFMR.

### 42.9.9 SSC Receive Synchronization Holding Register

**Name:** SSC_RSHR

**Address:** 0xF8004030 (0), 0xFC004030 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RSDAT | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RSDAT | | | | | | | |

- **RSDAT: Receive Synchronization Data**

### 42.9.10 SSC Transmit Synchronization Holding Register

**Name:** SSC_TSHR

**Address:** 0xF8004034 (0), 0xFC004034 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TSDAT | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TSDAT | | | | | | | |

- **TSDAT: Transmit Synchronization Data**

### 42.9.11 SSC Receive Compare 0 Register

**Name:** SSC_RC0R

**Address:** 0xF8004038 (0), 0xFC004038 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CP0 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CP0 | | | | | | | |

This register can only be written if the WPEN bit is cleared in the SSC Write Protection Mode Register.

- **CP0: Receive Compare Data 0**

### 42.9.12 SSC Receive Compare 1 Register

**Name:** SSC_RC1R

**Address:** 0xF800403C (0), 0xFC00403C (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CP1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CP1 | | | | | | | |

This register can only be written if the WPEN bit is cleared in the SSC Write Protection Mode Register.

- **CP1: Receive Compare Data 1**

### 42.9.13 SSC Status Register

**Name:** SSC_SR

**Address:** 0xF8004040 (0), 0xFC004040 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | RXEN | TXEN |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | RXSYN | TXSYN | CP1 | CP0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRUN | RXRDY | – | – | TXEMPTY | TXRDY |

- **TXRDY: Transmit Ready**

0: Data has been loaded in SSC_THR and is waiting to be loaded in the transmit shift register (TSR).

1: SSC_THR is empty.

- **TXEMPTY: Transmit Empty**

0: Data remains in SSC_THR or is currently transmitted from TSR.

1: Last data written in SSC_THR has been loaded in TSR and last data loaded in TSR has been transmitted.

- **RXRDY: Receive Ready**

0: SSC_RHR is empty.

1: Data has been received and loaded in SSC_RHR.

- **OVRUN: Receive Overrun**

0: No data has been loaded in SSC_RHR while previous data has not been read since the last read of the Status Register.

1: Data has been loaded in SSC_RHR while previous data has not yet been read since the last read of the Status Register.

- **CP0: Compare 0**

0: A compare 0 has not occurred since the last read of the Status Register.

1: A compare 0 has occurred since the last read of the Status Register.

- **CP1: Compare 1**

0: A compare 1 has not occurred since the last read of the Status Register.

1: A compare 1 has occurred since the last read of the Status Register.

- **TXSYN: Transmit Sync**

0: A Tx Sync has not occurred since the last read of the Status Register.

1: A Tx Sync has occurred since the last read of the Status Register.

- **RXSYN: Receive Sync**

0: An Rx Sync has not occurred since the last read of the Status Register.

1: An Rx Sync has occurred since the last read of the Status Register.

- **TXEN: Transmit Enable**

0: Transmit is disabled.

1: Transmit is enabled.

- **RXEN: Receive Enable**

0: Receive is disabled.

1: Receive is enabled.

### 42.9.14 SSC Interrupt Enable Register

**Name:** SSC_IER

**Address:** 0xF8004044 (0), 0xFC004044 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | RXSYN | TXSYN | CP1 | CP0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRUN | RXRDY | – | – | TXEMPTY | TXRDY |

- **TXRDY: Transmit Ready Interrupt Enable**

0: No effect.

1: Enables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Enable**

0: No effect.

1: Enables the Transmit Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Enable**

0: No effect.

1: Enables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Enable**

0: No effect.

1: Enables the Receive Overrun Interrupt.

- **CP0: Compare 0 Interrupt Enable**

0: No effect.

1: Enables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Enable**

0: No effect.

1: Enables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Enables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Enables the Rx Sync Interrupt.

### 42.9.15 SSC Interrupt Disable Register

**Name:** SSC_IDR

**Address:** 0xF8004048 (0), 0xFC004048 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | RXSYN | TXSYN | CP1 | CP0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRUN | RXRDY | – | – | TXEMPTY | TXRDY |

- **TXRDY: Transmit Ready Interrupt Disable**

0: No effect.

1: Disables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Disable**

0: No effect.

1: Disables the Transmit Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Disable**

0: No effect.

1: Disables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Disable**

0: No effect.

1: Disables the Receive Overrun Interrupt.

- **CP0: Compare 0 Interrupt Disable**

0: No effect.

1: Disables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Disable**

0: No effect.

1: Disables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Disables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Disables the Rx Sync Interrupt.

### 42.9.16 SSC Interrupt Mask Register

**Name:** SSC_IMR

**Address:** 0xF800404C (0), 0xFC00404C (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | RXSYN | TXSYN | CP1 | CP0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRUN | RXRDY | – | – | TXEMPTY | TXRDY |

- **TXRDY: Transmit Ready Interrupt Mask**

0: The Transmit Ready Interrupt is disabled.

1: The Transmit Ready Interrupt is enabled.

- **TXEMPTY: Transmit Empty Interrupt Mask**

0: The Transmit Empty Interrupt is disabled.

1: The Transmit Empty Interrupt is enabled.

- **RXRDY: Receive Ready Interrupt Mask**

0: The Receive Ready Interrupt is disabled.

1: The Receive Ready Interrupt is enabled.

- **OVRUN: Receive Overrun Interrupt Mask**

0: The Receive Overrun Interrupt is disabled.

1: The Receive Overrun Interrupt is enabled.

- **CP0: Compare 0 Interrupt Mask**

0: The Compare 0 Interrupt is disabled.

1: The Compare 0 Interrupt is enabled.

- **CP1: Compare 1 Interrupt Mask**

0: The Compare 1 Interrupt is disabled.

1: The Compare 1 Interrupt is enabled.

- **TXSYN: Tx Sync Interrupt Mask**

0: The Tx Sync Interrupt is disabled.

1: The Tx Sync Interrupt is enabled.

- **RXSYN: Rx Sync Interrupt Mask**

0: The Rx Sync Interrupt is disabled.

1: The Rx Sync Interrupt is enabled.

### 42.9.17 SSC Write Protection Mode Register

**Name:** SSC_WPMR

**Address:** 0xF80040E4 (0), 0xFC0040E4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x535343 (“SSC” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x535343 (“SSC” in ASCII).

See Section 42.8.10 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x535343 | PASSWD | Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0. |

### 42.9.18 SSC Write Protection Status Register

**Name:** SSC_WPSR

**Address:** 0xF80040E8 (0), 0xFC0040E8 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the SSC_WPSR.

1: A write protection violation has occurred since the last read of the SSC_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protect Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

# 43. Two-wire Interface (TWIHS)

## 43.1 Description

The Atmel Two-wire Interface (TWIHS) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 kbit/s in Fast mode and up to 3.4 Mbit/s in High-speed slave mode only, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C-compatible devices, such as a Real-Time Clock (RTC), Dot Matrix/Graphic LCD Controller and temperature sensor. The TWIHS is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Table 43-1 lists the compatibility level of the Atmel Two-wire Interface in Master mode and a full $I^2C$ compatible device.

**Table 43-1. Atmel TWI Compatibility with $I^2C$ Standard**

| $I^2C$ Standard | Atmel TWI |
|---|---|
| Standard Mode Speed (100 kHz) | Supported |
| Fast Mode Speed (400 kHz) | Supported |
| High-speed Mode (Slave only, 3.4 MHz) | Supported |
| 7- or 10-bit(1) Slave Addressing | Supported |
| START Byte(2) | Not Supported |
| Repeated Start (Sr) Condition | Supported |
| ACK and NACK Management | Supported |
| Input Filtering | Supported |
| Slope Control | Not Supported |
| Clock Stretching | Supported |
| Multi Master Capability | Supported |

Notes: 1. 10-bit support in Master mode only
2. START + b000000001 + Ack + Sr

## 43.2 Embedded Characteristics

- 2 TWIHSs
- 16-byte Transmit and Receive FIFOs
- Compatible with Atmel Two-wire Interface Serial Memory and I²C Compatible Devices(1)
- One, Two or Three Bytes for Slave Address
- Sequential Read/Write Operations
- Master and Multimaster Operation (Standard and Fast Modes Only)
- Slave Mode Operation (Standard, Fast and High-Speed Modes)
- Bit Rate: Up to 400 Kbit/s in Fast Mode and 3.4 Mbit/s in High-Speed Mode (Slave Mode Only)
- General Call Supported in Slave Mode
- SleepWalking (Asynchronous and Partial Wakeup)
- SMBus Support
- Connection to DMA Controller (DMA) Channel Capabilities Optimizes Data Transfers
- Register Write Protection

Note: 1. See Table 43-1 for details on compatibility with I²C Standard.

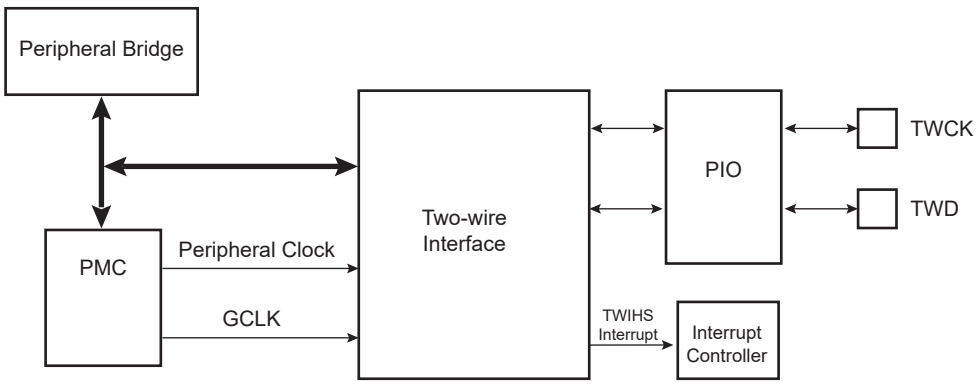
## 43.3 List of Abbreviations

**Table 43-2. Abbreviations**

| Abbreviation | Description |
|---|---|
| TWI | Two-wire Interface |
| A | Acknowledge |
| NA | Non Acknowledge |
| P | Stop |
| S | Start |
| Sr | Repeated Start |
| SADR | Slave Address |
| ADR | Any address except SADR |
| R | Read |
| W | Write |

## 43.4 Block Diagram

**Figure 43-1. Block Diagram**



### 43.4.1 I/O Lines Description

**Table 43-3. I/O Lines Description**

| Pin Name | Pin Description | Type |
|---|---|---|
| TWD | Two-wire Serial Data | Input/Output |
| TWCK | Two-wire Serial Clock | Input/Output |

## 43.5 Product Dependencies

### 43.5.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor. When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWIHS, the user must program the PIO Controller to dedicate TWD and TWCK as peripheral lines. When High-speed Slave mode is enabled, the analog pad filter must be enabled.

The user must not program TWD and TWCK as open-drain. This is already done by the hardware.

**Table 43-4. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| TWIHS0 | TWCK0 | PC0 | D |
| TWIHS0 | TWCK0 | PC28 | E |
| TWIHS0 | TWCK0 | PD22 | B |
| TWIHS0 | TWCK0 | PD30 | E |
| TWIHS0 | TWD0 | PB31 | D |
| TWIHS0 | TWD0 | PC27 | E |
| TWIHS0 | TWD0 | PD21 | B |
| TWIHS0 | TWD0 | PD29 | E |
| TWIHS1 | TWCK1 | PC7 | C |
| TWIHS1 | TWCK1 | PD5 | A |
| TWIHS1 | TWCK1 | PD20 | B |
| TWIHS1 | TWD1 | PC6 | C |
| TWIHS1 | TWD1 | PD4 | A |
| TWIHS1 | TWD1 | PD19 | B |

### 43.5.2 Power Management

Enable the peripheral clock.

The TWIHS may be clocked through the Power Management Controller (PMC), thus the user must first configure the PMC to enable the TWIHS clock.

### 43.5.3 Interrupt Sources

The TWIHS has an interrupt line connected to the Interrupt Controller. In order to handle interrupts, the Interrupt Controller must be programmed before configuring the TWIHS.

**Table 43-5. Peripheral IDs**

| Instance | ID |
|---|---|
| TWIHS0 | 29 |
| TWIHS1 | 30 |

## 43.6 Functional Description

### 43.6.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited. See Figure 43-3.

Each transfer begins with a START condition and terminates with a STOP condition. See Figure 43-2.

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines the STOP condition.

**Figure 43-2. START and STOP Conditions**

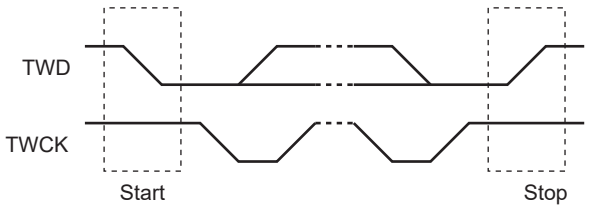
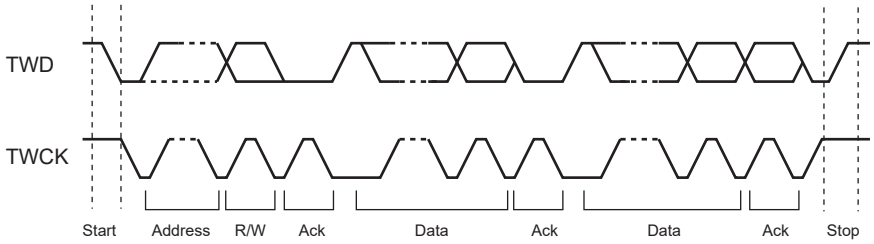



**Figure 43-3. Transfer Format**




### 43.6.2 Modes of Operation

The TWIHS has different modes of operation:

- Master Transmitter mode (Standard and Fast modes only)
- Master Receiver mode (Standard and Fast modes only)
- Multimaster Transmitter mode (Standard and Fast modes only)
- Multimaster Receiver mode (Standard and Fast modes only)
- Slave Transmitter mode (Standard, Fast and High-speed modes)
- Slave Receiver mode (Standard, Fast and High-speed modes)

These modes are described in the following sections.

### 43.6.3 Master Mode

#### 43.6.3.1 Definition

The master is the device that starts a transfer, generates a clock and stops it. This operating mode is not available if High-speed mode is selected.

#### 43.6.3.2 Programming Master Mode

The following registers must be programmed before entering Master mode:

1. TWIHS_MMR.DADR (+ IADRSZ + IADR if a 10-bit device is addressed): The device address is used to access slave devices in Read or Write mode.
2. TWIHS_CWGR.CKDIV + CHDIV + CLDIV: Clock Waveform register
3. TWIHS_CR.SVDIS: Disables the Slave mode
4. TWIHS_CR.MSEN: Enables the Master mode

Note: If the TWIHS is already in Master mode, the device address (DADR) can be configured without disabling the Master mode.

#### 43.6.3.3 Transfer Rate Clock Source

The TWIHS speed is defined in the TWIHS_CWGR. The TWIHS baud rate can be based either on the peripheral clock if the CKSRC bit value is ‘0’ or on a GCLK clock if the CKSRC bit value is ‘1’.

If CKSRC = 1, the baud rate is independent of the system/core clock (MCK) and thus the MCK frequency can be changed without affecting the TWIHS transfer rate.

The GCLK frequency must always be three times lower than the peripheral clock frequency.

#### 43.6.3.4 Master Transmitter Mode

This operating mode is not available if High-speed mode is selected.

After the master initiates a START condition when writing into the Transmit Holding register (TWIHS_THR), it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWIHS_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWIHS_MMR).

The TWIHS transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. If the slave does not acknowledge the byte, then the Not Acknowledge flag (NACK) is set in the TWIHS Status Register (TWIHS_SR) of the master and a STOP condition is sent. The NACK flag must be cleared by reading TWIHS_SR before the next write into TWIHS_THR. As with the other status bits, an interrupt can be generated if enabled in the Interrupt Enable register (TWIHS_IER). If the slave acknowledges the byte, the data written in the TWIHS_THR is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in the TWIHS_THR.

TXRDY is used as Transmit Ready for the DMA transmit channel.

While no new data is written in the TWIHS_THR, the serial clock line is tied low. When new data is written in the TWIHS_THR, the SCL is released and the data is sent. Setting the STOP bit in TWIHS_CR generates a STOP condition.

After a master write transfer, the serial clock line is stretched (tied low) as long as no new data is written in the TWIHS_THR or until a STOP command is performed.

To clear the TXRDY flag, first set the bit TWIHS_CR.MSDIS, then set the bit TWIHS_CR.MSEN.

See Figure 43-4, Figure 43-5, and Figure 43-6.

**Figure 43-4. Master Write with One Data Byte**

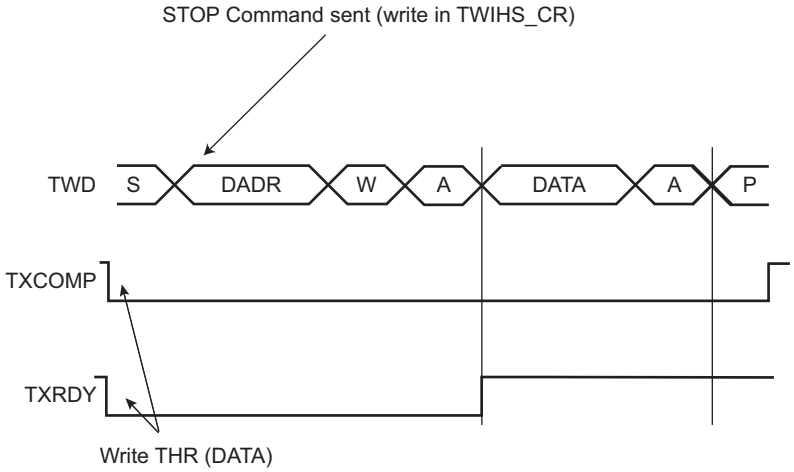



**Figure 43-5. Master Write with Multiple Data Bytes**

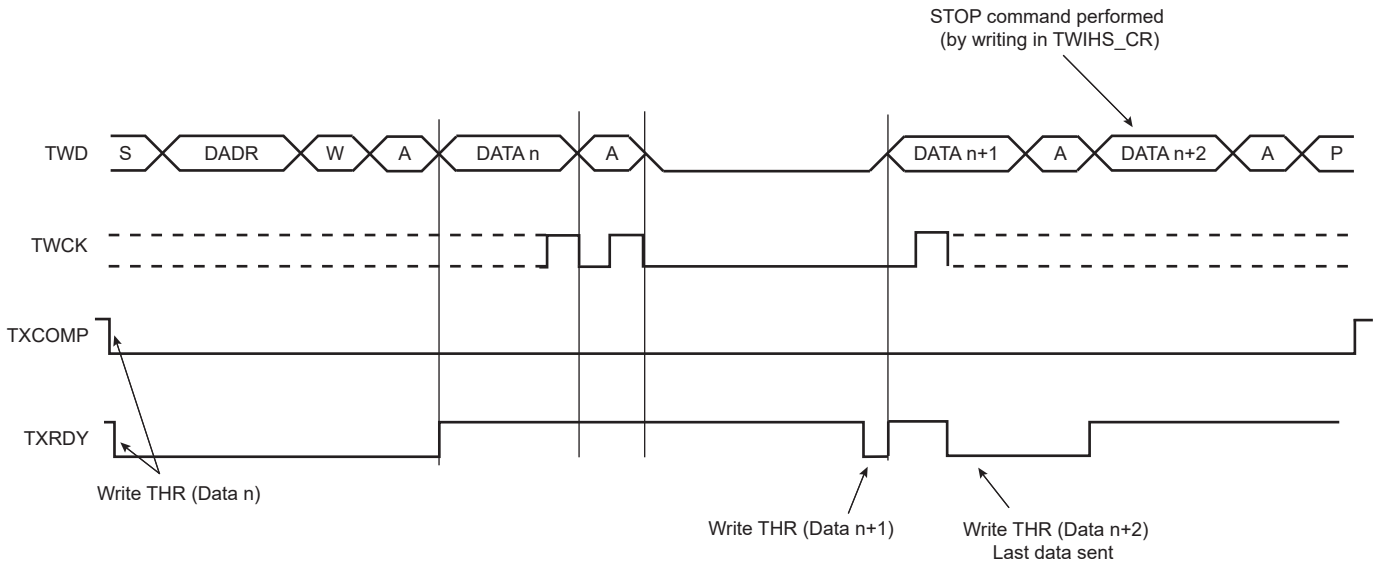
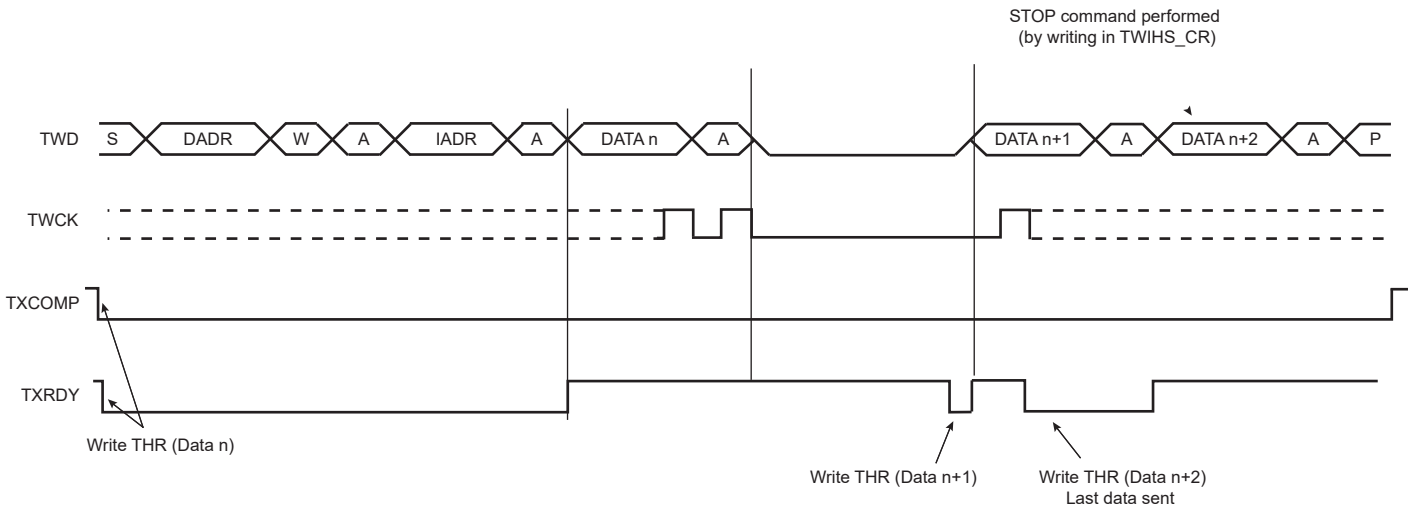



**Figure 43-6. Master Write with One-Byte Internal Address and Multiple Data Bytes**



#### 43.6.3.5 Master Receiver Mode

Master Receiver mode is not available if High-speed mode is selected.

The read sequence begins by setting the START bit. After the START condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction, 1 in this case (MREAD = 1 in TWIHS_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the NACK bit in the TWIHS_SR if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data (see Figure 43-7). When the RXRDY bit is set in the TWIHS_SR, a character has been received in the Receive Holding register (TWIHS_RHR). The RXRDY bit is reset when reading the TWIHS_RHR.

When a single data byte read is performed, with or without internal address **(**IADR), the START and STOP bits must be set at the same time. See Figure 43-7. When a multiple data byte read is performed, with or without internal address **(**IADR), the STOP bit must be set after the next-to-last data received (same condition applies for START bit to generate a REPEATED START). See Figure 43-8. For internal address usage, see Section 43.6.3.6 “Internal Address”.

If TWIHS_RHR is full (RXRDY high) and the master is receiving data, the serial clock line is tied low before receiving the last bit of the data and until the TWIHS_RHR is read. Once the TWIHS_RHR is read, the master stops stretching the serial clock line and ends the data reception. See Figure 43-9.

**Warning:** When receiving multiple bytes in Master Read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access is not completed until TWIHS_RHR is read. The last access stops on the next-to-last bit (clock stretching). When the TWIHS_RHR is read, there is only half a bit period to send the STOP (or START) command, else another read access might occur (spurious access).

A possible workaround is to set the STOP (or START) bit before reading the TWIHS_RHR on the next-to-last access (within IT handler).

**Figure 43-7. Master Read with One Data Byte**

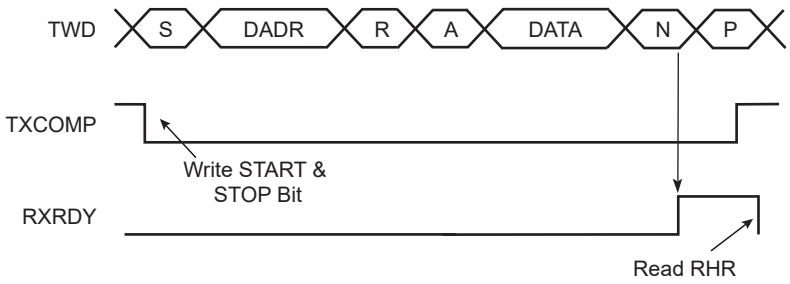


**Figure 43-8. Master Read with Multiple Data Bytes**

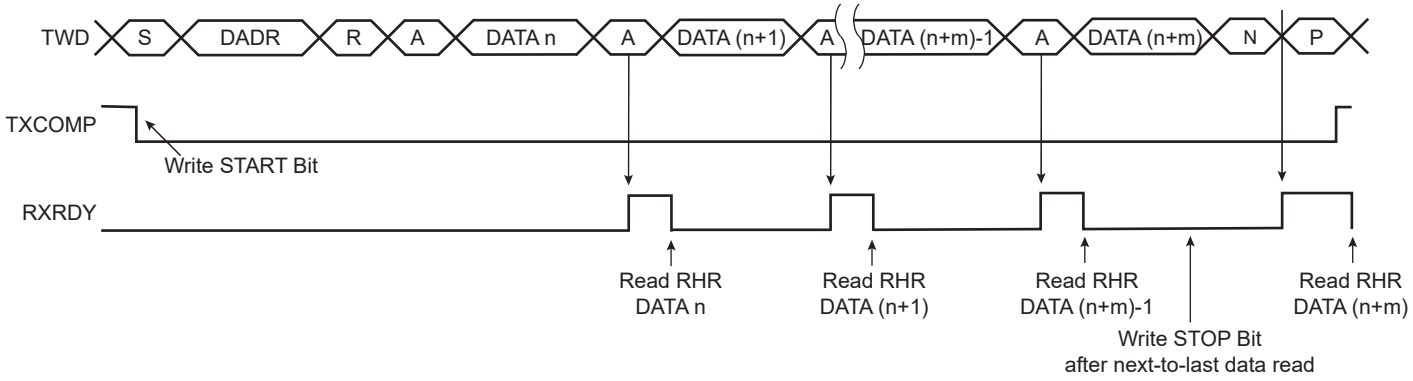
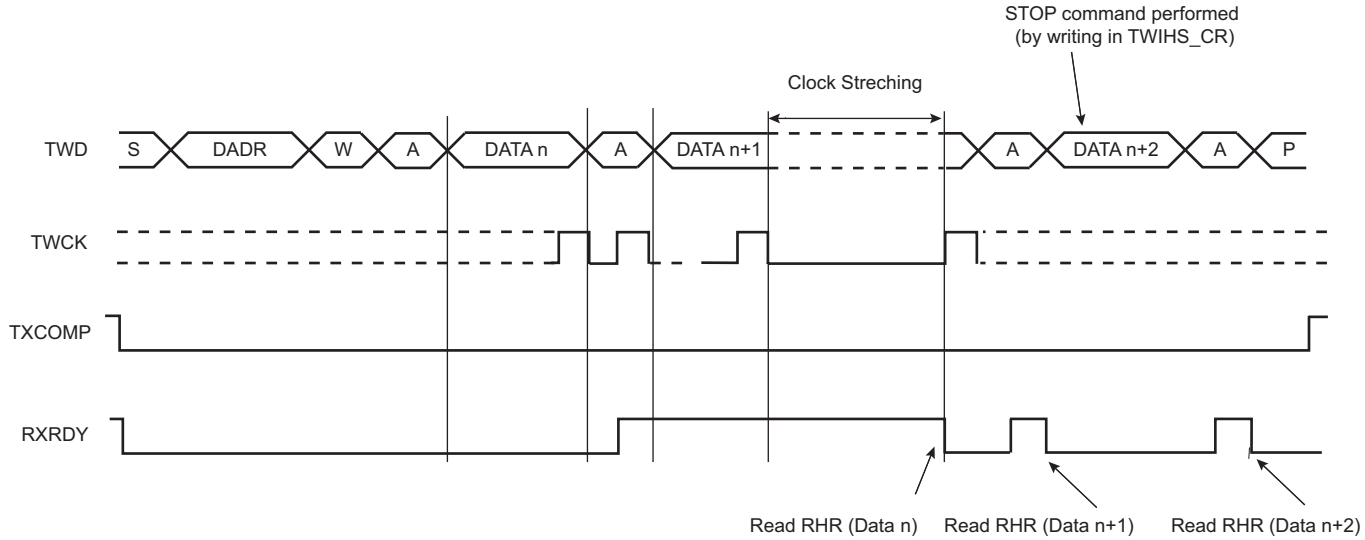



**Figure 43-9. Master Read Clock Stretching with Multiple Data Bytes**




RXRDY is used as receive ready for the DMA receive channel.

### 43.6.3.6 Internal Address

The TWIHS can perform transfers with 7-bit slave address devices and with 10-bit slave address devices.

#### 7-bit Slave Addressing

When addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, e.g. within a memory page location in a serial memory. When performing read operations with an internal address, the TWIHS performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second START condition (after sending the IADR) is sometimes called “repeated start” (Sr) in $I^2C$ fully-compatible devices. See Figure 43-11.

See Figure 43-10 and Figure 43-12 for the master write operation with internal address.

The three internal address bytes are configurable through TWIHS_MMR.

If the slave device supports only a 7-bit address, i.e., no internal address, IADRSZ must be set to 0.

Table 43-6 shows the abbreviations used in Figure 43-10 and Figure 43-11.

**Table 43-6. Abbreviations**

| Abbreviation | Definition |
|---|---|
| S | Start |
| Sr | Repeated Start |
| P | Stop |
| W | Write |
| R | Read |
| A | Acknowledge |
| NA | Not Acknowledge |
| DADR | Device Address |
| IADR | Internal Address |

**Figure 43-10. Master Write with One-, Two- or Three-Byte Internal Address and One Data Byte**

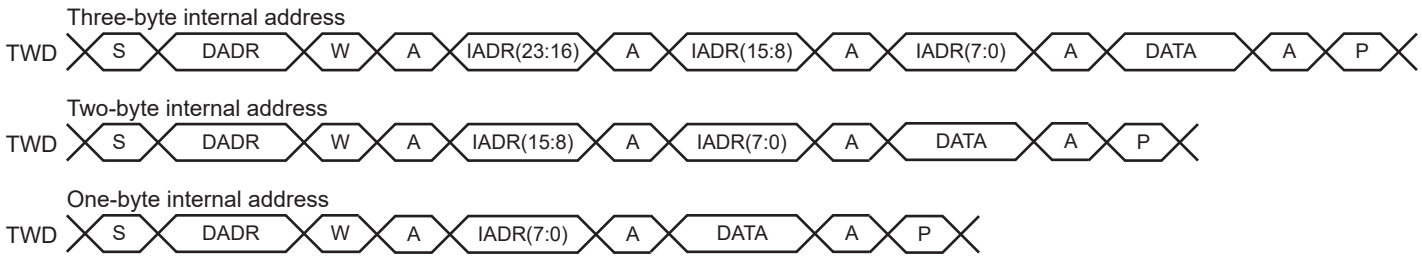
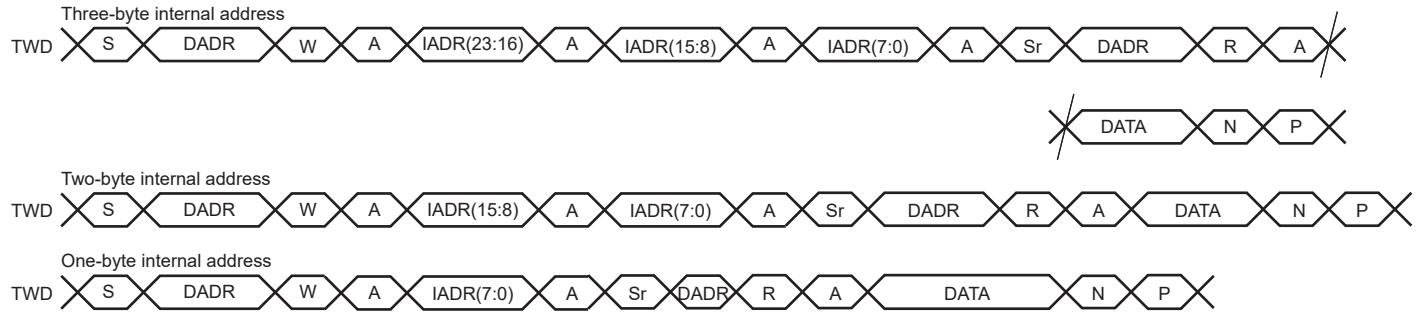



**Figure 43-11. Master Read with One-, Two- or Three-Byte Internal Address and One Data Byte**




### 10-bit Slave Addressing

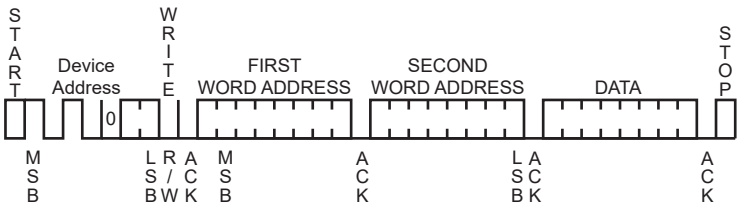
For a slave address higher than seven bits, configure the address size (IADRSZ) and set the other slave address bits in the Internal Address register (TWIHS_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16], can be used the same way as in 7-bit slave addressing.

**Example:** Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWIHS_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 43-12 shows a byte write to a memory device. This demonstrates the use of internal addresses to access the device.

**Figure 43-12. Internal Address Usage**




#### 43.6.3.7 Repeated Start

In addition to Internal Address mode, REPEATED START (Sr) can be generated manually by writing the START bit at the end of a transfer instead of the STOP bit. In such case, the parameters of the next transfer (direction, SADR, etc.) need to be set before writing the START bit at the end of the previous transfer.

See Section 43.6.3.14 “Read/Write Flowcharts” for detailed flowcharts.

Note that generating a REPEATED START after a single data read is not supported.

#### 43.6.3.8 Bus Clear Command

The TWIHS can perform a Bus Clear command:

1. Configure the Master mode (DADR, CKDIV, etc).
2. Start the transfer by setting the CLEAR bit in the TWIHS_CR.

Note: If alternative command is used (ACMEN bit set to ‘1’) DATAL field must be set to 0.

#### 43.6.3.9 Using the DMA Controller (DMAC) in Master Mode

The use of the DMA significantly reduces the CPU load.

To ensure correct implementation, follow the programming sequences below:

##### Data Transmit with the DMA in Master Mode

If Alternative Command mode is disabled (ACMEN bit set to ‘0’):

The DMA transfer size must be defined with the buffer size minus 1. The remaining character must be managed without DMA to ensure that the exact number of bytes are transmitted regardless of system bus latency conditions during the end of the buffer transfer period.

1. Initialize the DMA (channels, memory pointers, size - 1, etc.);
2. Configure the Master mode (DADR, CKDIV, MREAD = 0, etc.) or Slave mode.
3. Enable the DMA.
4. Wait for the DMA status flag indicating that the buffer transfer is complete.
5. Disable the DMA.
6. Wait for the TXRDY flag in TWIHS_SR.
7. Set the STOP bit in TWIHS_CR.
8. Write the last character in TWIHS_THR.
9. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

If Alternative Command mode is enabled (ACMEN bit set to ‘1’):

1. Initialize the transmit DMA (memory pointers, transfer size).
2. Configure the Master mode (DADR, CKDIV, etc.) and TWIHS_ACR.
3. Start the transfer by setting the DMA TXTEN bit.
4. Wait for the DMA ENDTX flag either by using the polling method or ENDTX interrupt.
5. Disable the DMA by setting the DMA TXTDIS bit.
6. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

#### Data Receive with the DMA in Master Mode

If Alternative Command mode is disabled (ACMEN bit set to ‘0’):

The DMA transfer size must be defined with the buffer size minus 2. The two remaining characters must be managed without DMA to ensure that the exact number of bytes are received regardless of system bus latency conditions encountered during the end of buffer transfer period.

1. Initialize the DMA (channels, memory pointers, size - 2, etc.);
2. Configure the Master mode (DADR, CKDIV, MREAD = 1, etc.) or Slave mode.
3. Enable the DMA.
4. (Master Only) Write the START bit in the TWIHS_CR to start the transfer.
5. Wait for the DMA status flag indicating that the buffer transfer is complete.
6. Disable the DMA.
7. Wait for the RXRDY flag in the TWIHS_SR.
8. Set the STOP bit in TWIHS_CR.
9. Read the penultimate character in TWIHS_RHR.
10. Wait for the RXRDY flag in the TWIHS_SR.
11. Read the last character in TWIHS_RHR.
12. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

If Alternative Command mode is enabled (ACMEN bit set to ‘1’):

1. Initialize the transmit DMA (memory pointers, transfer size).
2. Configure the Master mode (DADR, CKDIV, etc.) and TWIHS_ACR.
3. Set the DMA RXTEN bit.
4. (Master Only) Write the START bit in the TWIHS_CR to start the transfer.
5. Wait for the DMA ENDTX Flag either by using the polling method or ENDTX interrupt.
6. Disable the DMA by setting the DMA TXTDIS bit.
7. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

### 43.6.3.10 SMBus Mode

SMBus mode is enabled when a one is written to the SMEN bit in the TWIHS_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

- Only 7-bit addressing can be used.
- The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into TWIHS_SMBTR.
- Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
- A set of addresses has been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring the TWIHS_CR.

#### Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing a one to the PECEN bit in TWIHS_CR enables automatic PEC handling in the current transfer. Transfers with and without PEC can be intermixed in the same system, since some slaves may not support PEC. The PEC LFSR is always updated on every bit transmitted or received, so that PEC handling on combined transfers is correct.

In Master Transmitter mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave compares it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave returns an ACK to the master. If the PEC values differ, data was corrupted, and the slave returns a NACK value. Some slaves may not be able to check the

received PEC in time to return a NACK if an error occurred. In this case, the slave should always return an ACK after the PEC byte, and another method must be used to verify that the transmission was received correctly.

In Master Receiver mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master compares it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the PECERR bit in TWIHS_SR is set. In Master Receiver mode, the PEC byte is always followed by a NACK transmitted by the master, since it is the last byte in the transfer.

In combined transfers, the PECRQ bit should only be set in the last of the combined transfers.
If the Alternative Command mode is enabled, only the NPEC bit should be set.

Consider the following transfer:

S, ADR+W, COMMAND_BYTE, ACK, SR, ADR+R, DATA_BYTE, ACK, PEC_BYTE, NACK, P

See Section 43.6.3.14 “Read/Write Flowcharts” for detailed flowcharts.

#### Timeouts

The TLOWS and TLOWM fields in TWIHS_SMBTR configure the SMBus timeout values. If a timeout occurs, the master transmits a STOP condition and leaves the bus. The TOUT bit is also set in TWIHS_SR.

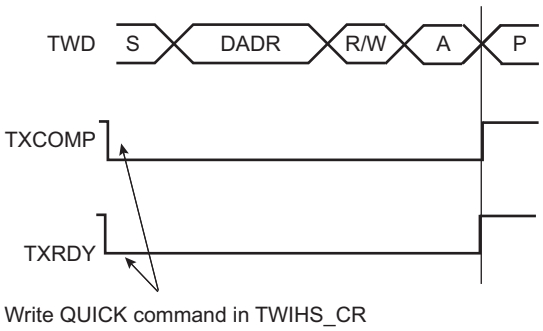
### 43.6.3.11 SMBus Quick Command (Master Mode Only)

The TWIHS can perform a quick command:

1. Configure the Master mode (DADR, CKDIV, etc).
2. Write the MREAD bit in the TWIHS_MMR at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWIHS_CR.

Note: If alternative command is used (ACMEN bit set to ‘1’) DATAL field must be set to 0.

**Figure 43-13. SMBus Quick Command**




### 43.6.3.12 Alternative Command

Another way to configure the transfer is to enable the Alternative Command mode with the ACMEN bit of the TWIHS Control Register.

In this mode, the transfer is configured through the TWIHS Alternative Command Register. It is possible to define a simple read or write transfer or a combined transfer with a repeated start.

In order to set a simple transfer, the DATAL field and the DIR field of the TWIHS Alternative Command Register must be filled accordingly and the NDATAL field must be cleared. To begin the transfer, either set the START bit in the TWIHS Control Register in case of a read transfer, or write the TWIHS Transmit Holding Register in case of a write transfer.

For a combined transfer linked by a repeated start, the NDATAL field must be filled with the length of the second transfer and NDIR with the corresponding direction.

The PEC and NPEC bits are used to set a PEC field. In the case of a single transfer with PEC, the PEC bit must be set. In the case of a combined transfer, the NPEC bit must be set.

Note: If Alternative Command mode is used, IADRSZ in TWIHS_MMR must be set to 0.

See Section 43.6.3.14 “Read/Write Flowcharts” for detailed flowcharts.

#### 43.6.3.13 Handling Errors in Alternative Command

If a NACK is generated by a slave device or SMBus timeout error, the TWIHS stops the frame immediately, although the DMA transfer may still be active. To prevent a new frame from being restarted with the remaining DMA data (transmit), the TWIHS prevents any start of frame until the LOCK flag is cleared in the TWIHS_SR.

The LOCK bit in the TWIHS_SR indicates the state of the TWIHS (locked or not locked).

When the TWIHS is locked, no transfer begins until the LOCK is cleared by the LOCKCLR bit in the TWIHS_CR and error flags are cleared by reading the TWIHS_SR.

In case of error, the TWIHS_THR may have been loaded with a new data. The THRCLR bit in the TWIHS_CR can be used to flush the TWIHS_THR. If the THRCLR bit is set, the TXRDY and TXCOMP flags are set.

#### 43.6.3.14 Read/Write Flowcharts

The flowcharts give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that TWIHS_IER be configured first.

**Figure 43-14. TWIHS Write Operation with Single Data Byte without Internal Address**

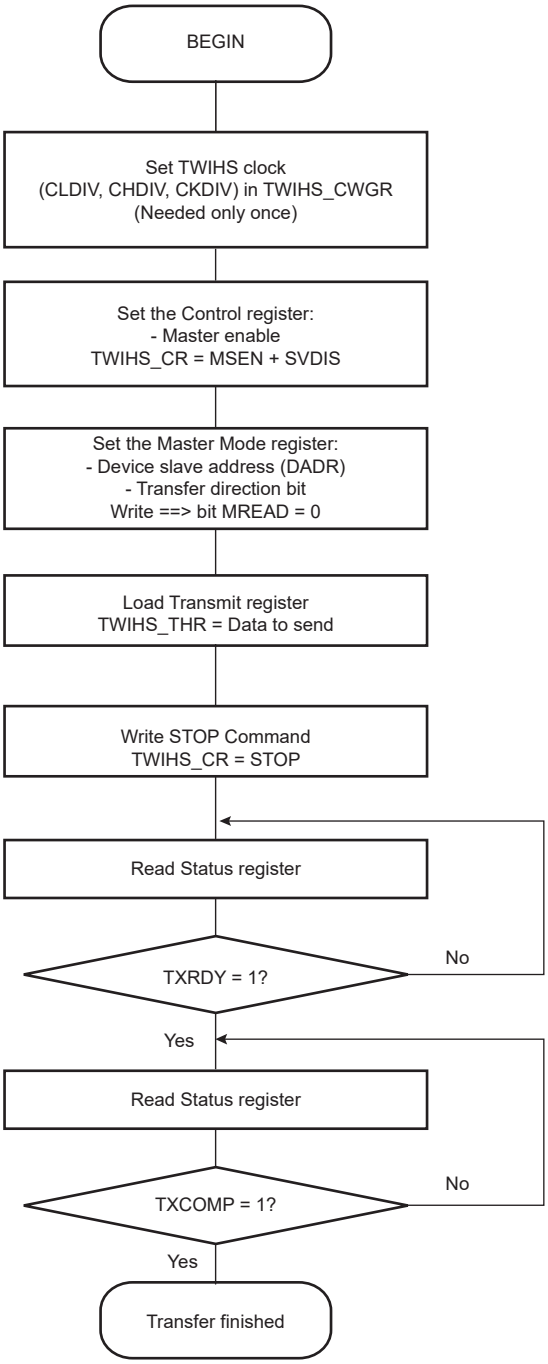


**Figure 43-15. TWIHS Write Operation with Single Data Byte and Internal Address**

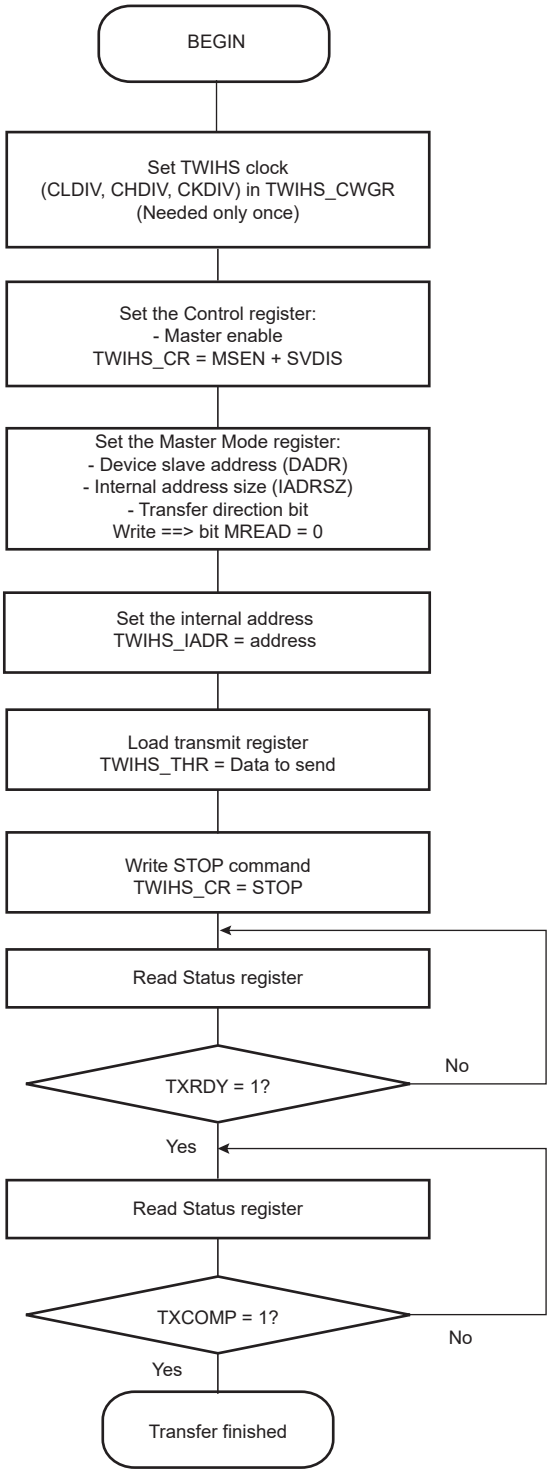


**Figure 43-16. TWIHS Write Operation with Multiple Data Bytes with or without Internal Address**

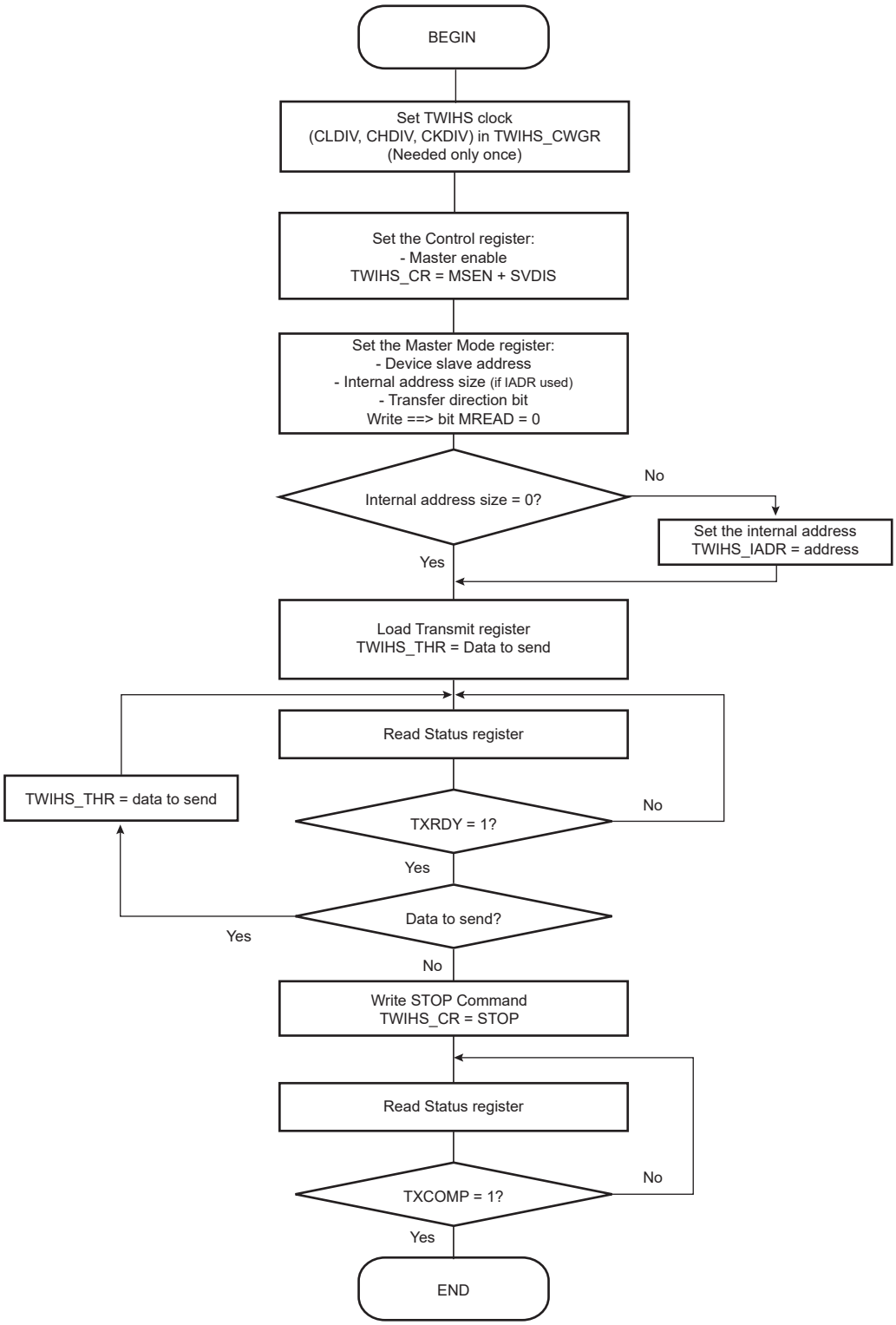


**Figure 43-17. SMBus Write Operation with Multiple Data Bytes with or without Internal Address and PEC Sending**

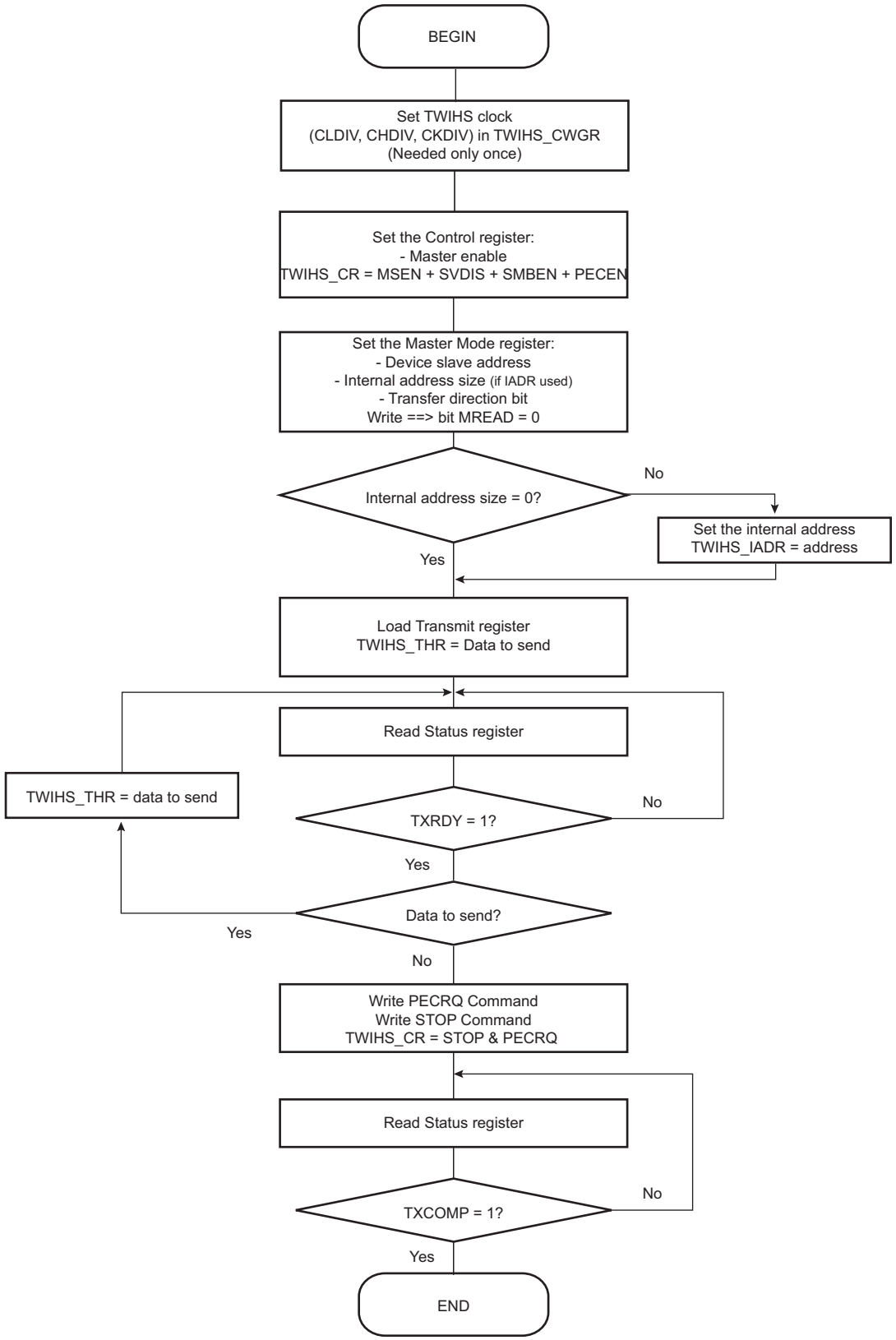


**Figure 43-18. SMBus Write Operation with Multiple Data Bytes with PEC and Alternative Command Mode**

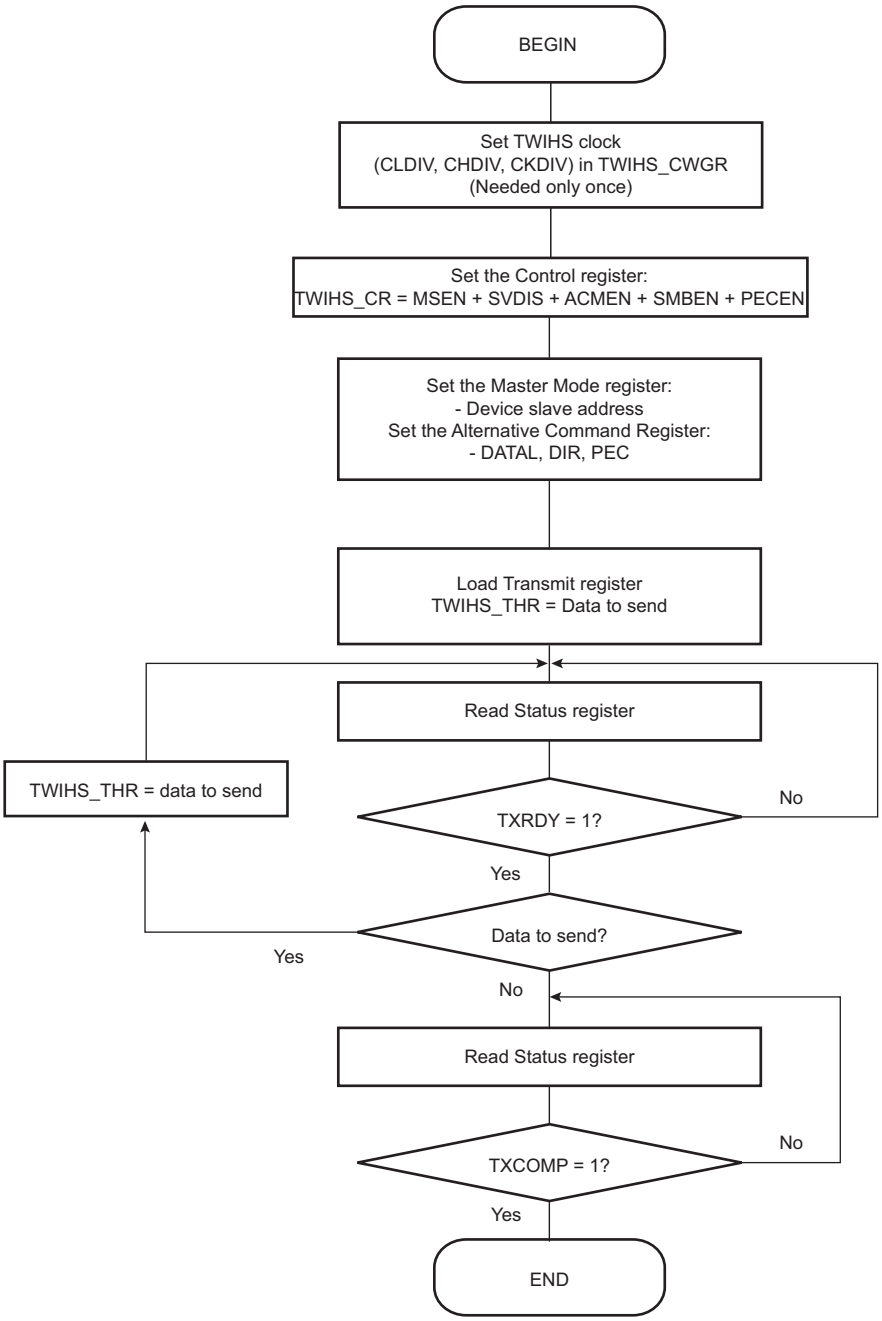


**Figure 43-19. TWIHS Write Operation with Multiple Data Bytes and Read Operation with Multiple Data Bytes (Sr)**

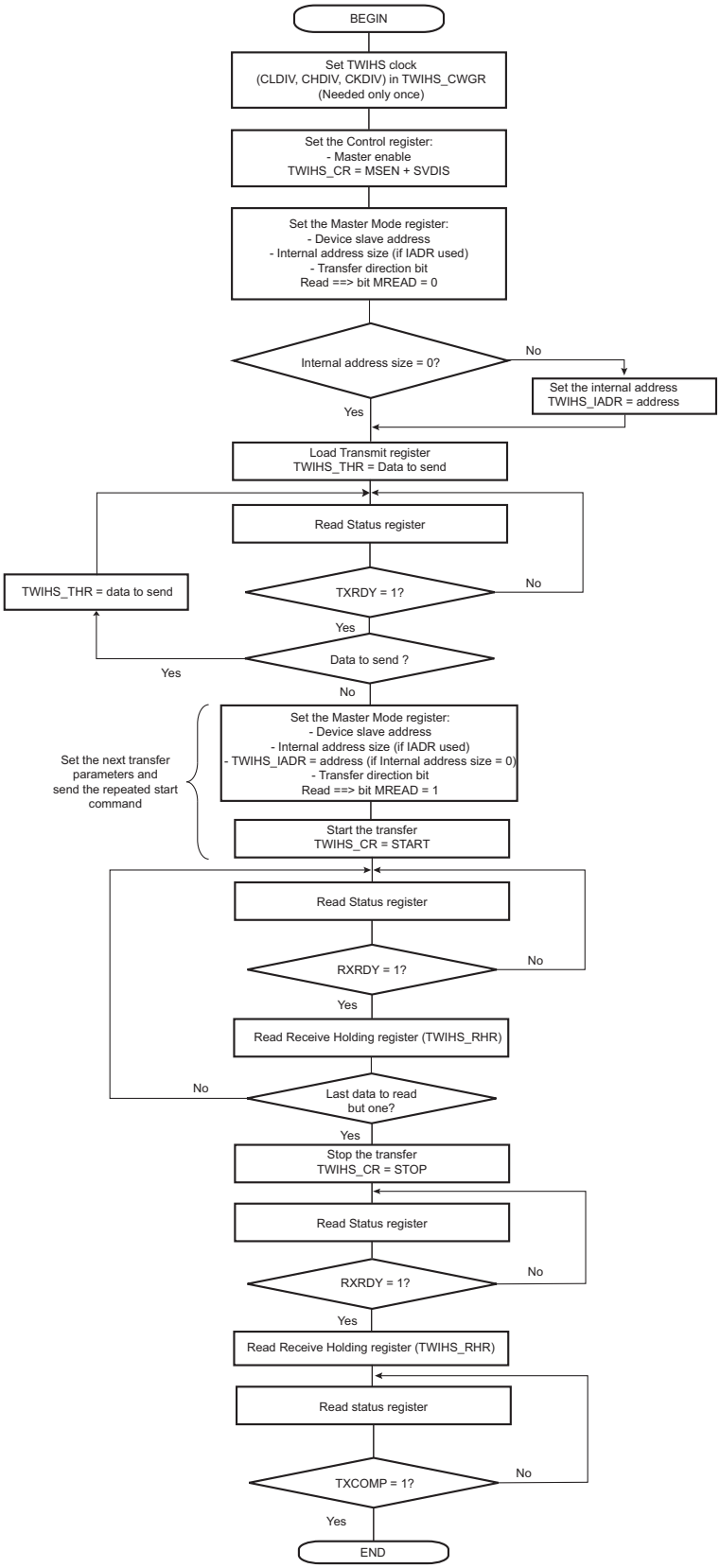


**Figure 43-20. TWIHS Write Operation with Multiple Data Bytes + Read Operation and Alternative Command Mode + PEC**

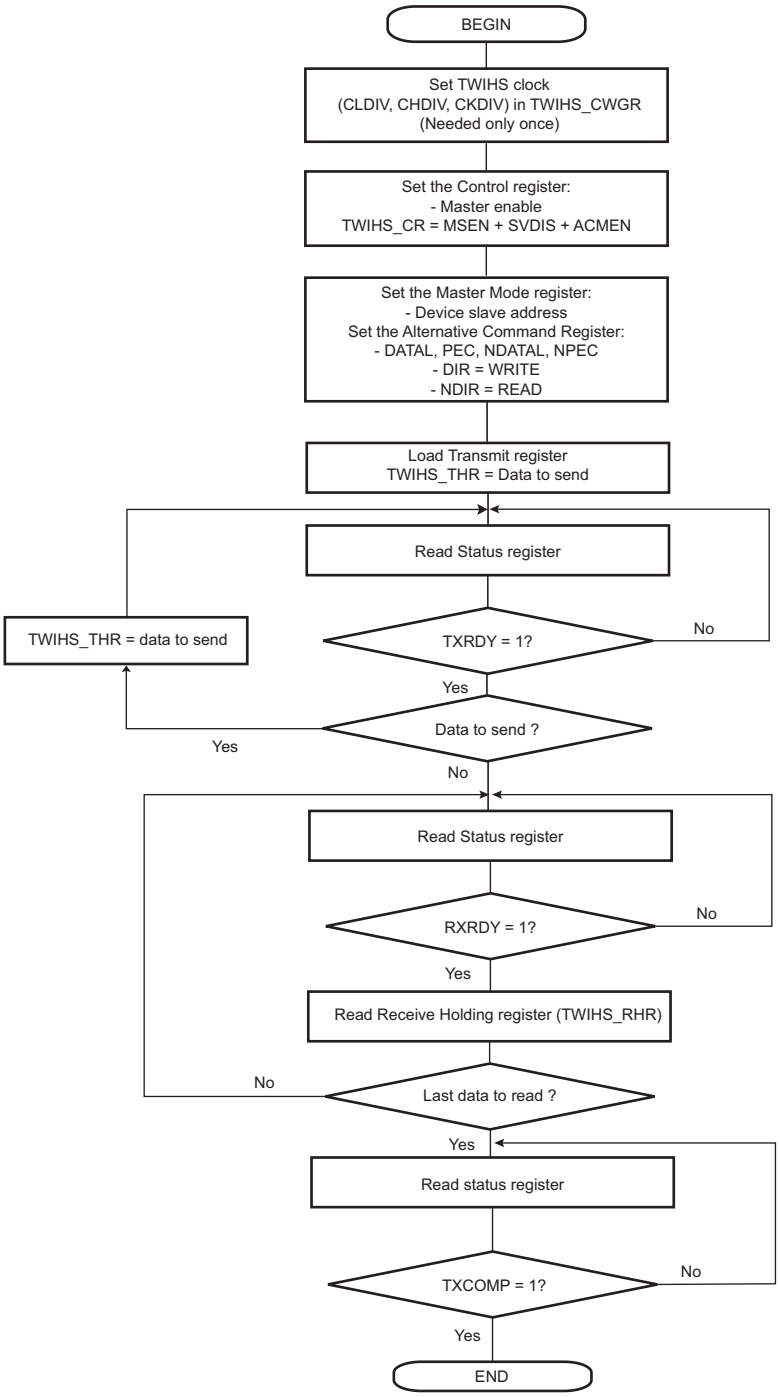


**Figure 43-21. TWIHS Read Operation with Single Data Byte without Internal Address**

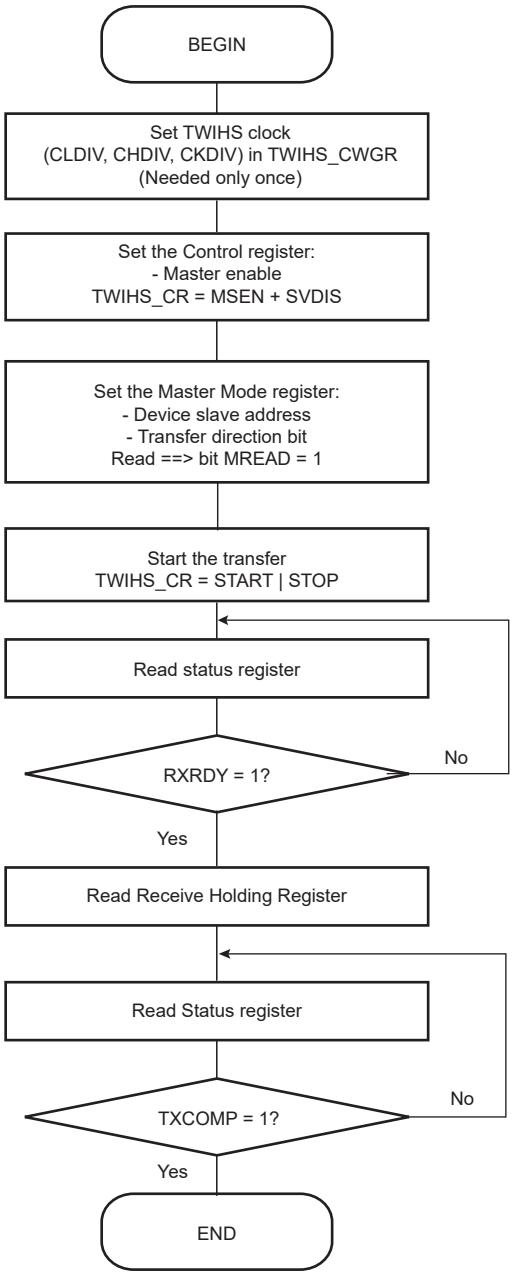


**Figure 43-22. TWIHS Read Operation with Single Data Byte and Internal Address**

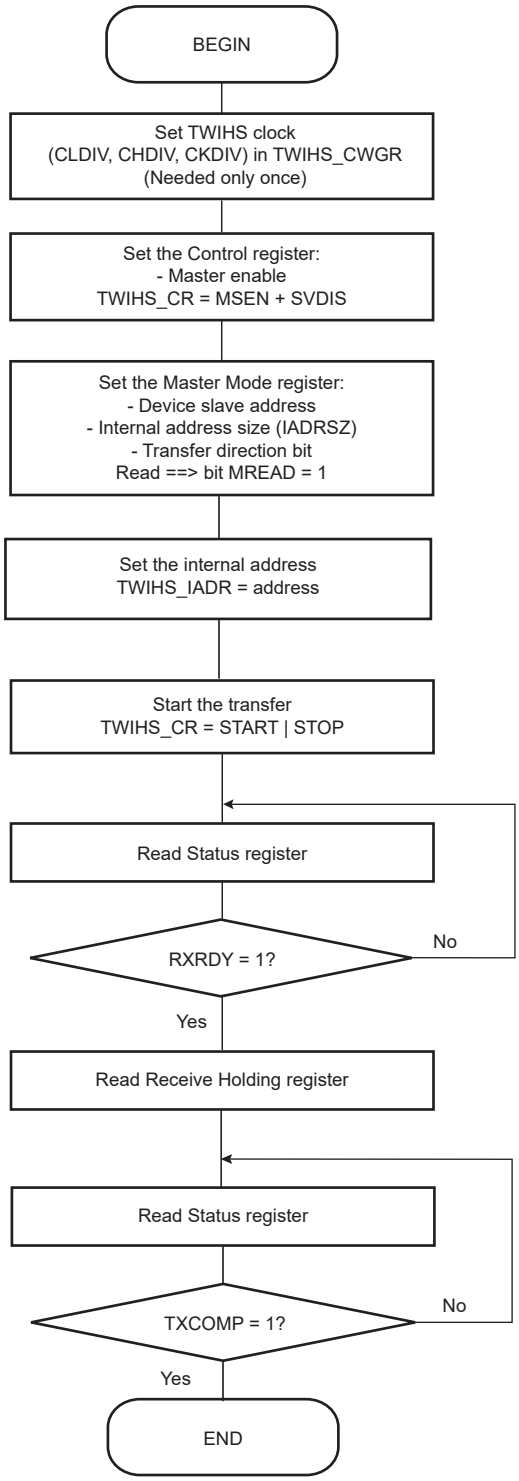


**Figure 43-23. TWIHS Read Operation with Multiple Data Bytes with or without Internal Address**

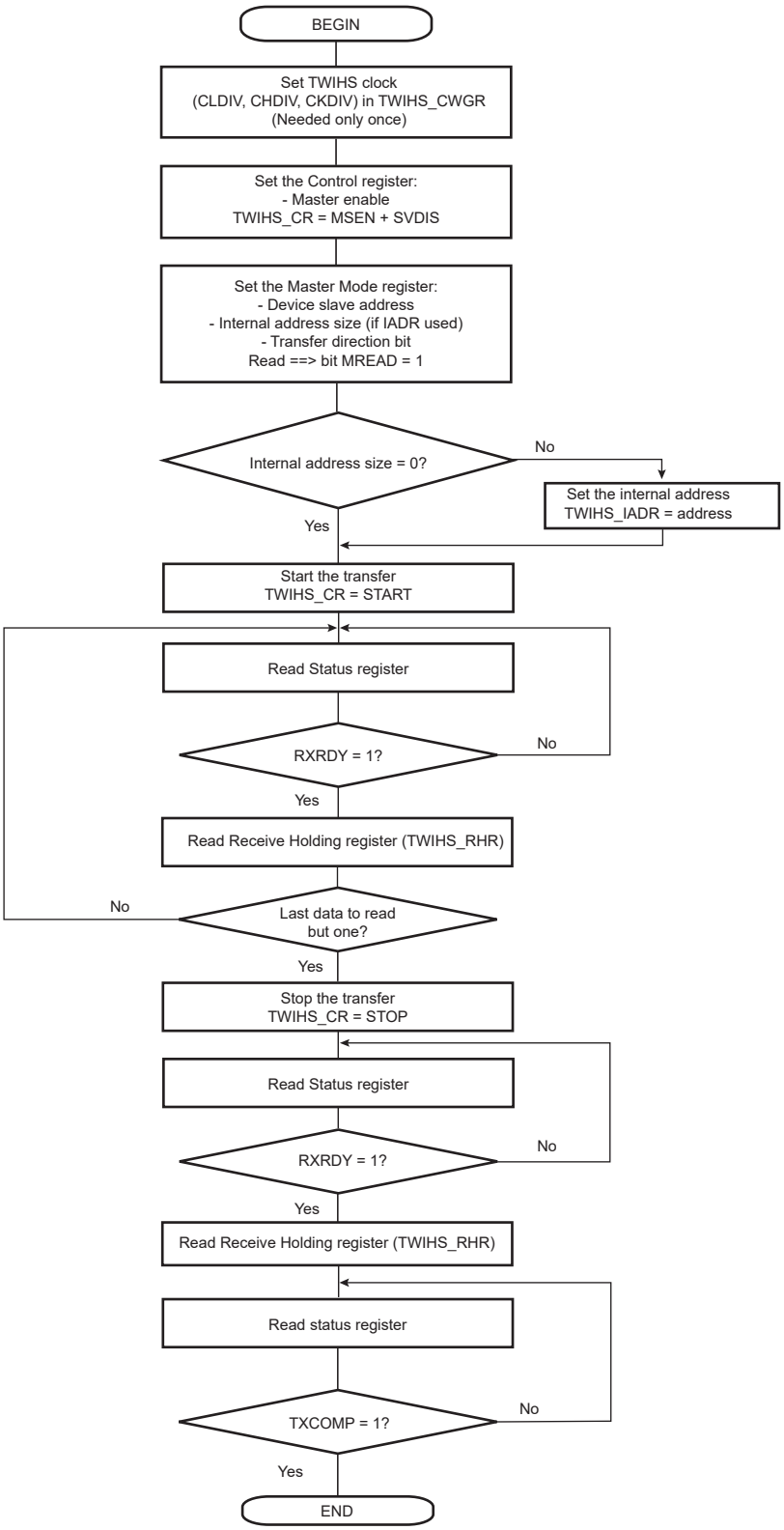


**Figure 43-24. TWIHS Read Operation with Multiple Data Bytes with or without Internal Address with PEC**

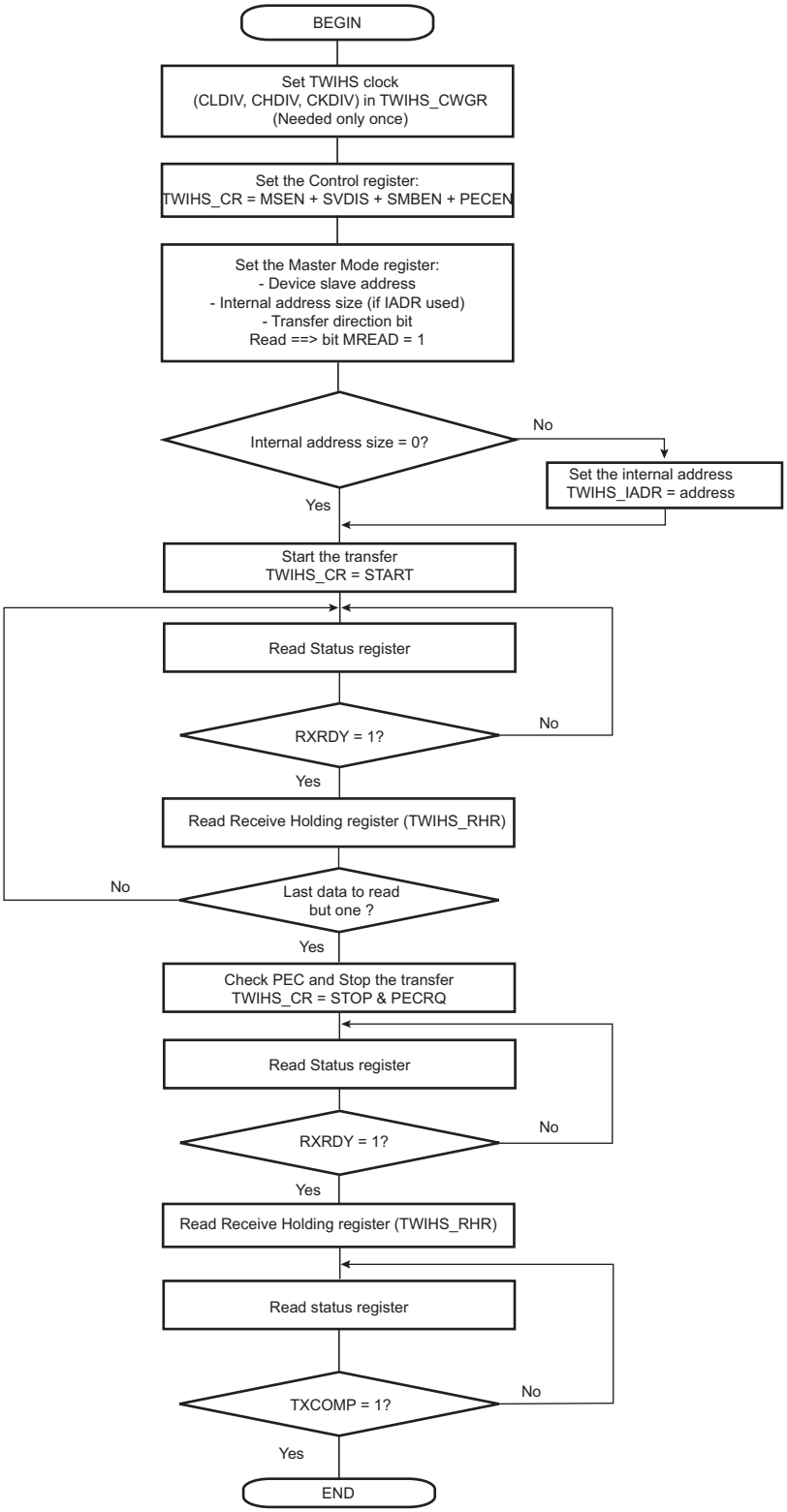


**Figure 43-25. TWIHS Read Operation with Multiple Data Bytes with Alternative Command Mode with PEC**

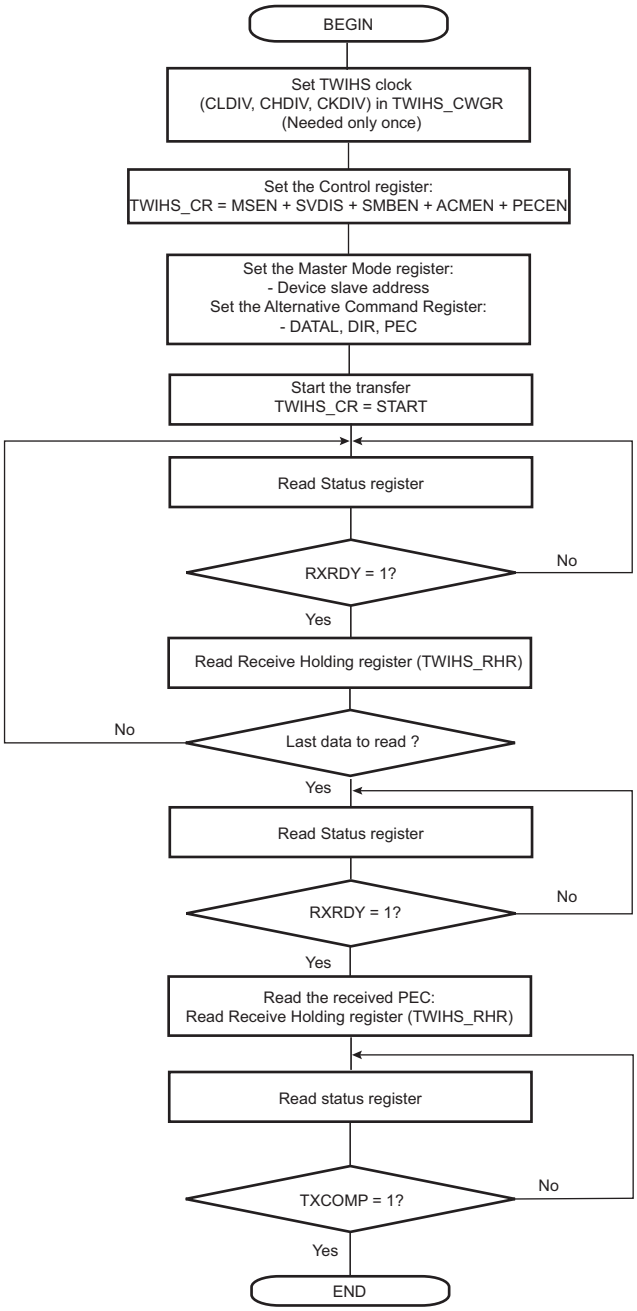


**Figure 43-26. TWIHS Read Operation with Multiple Data Bytes + Write Operation with Multiple Data Bytes (Sr)**

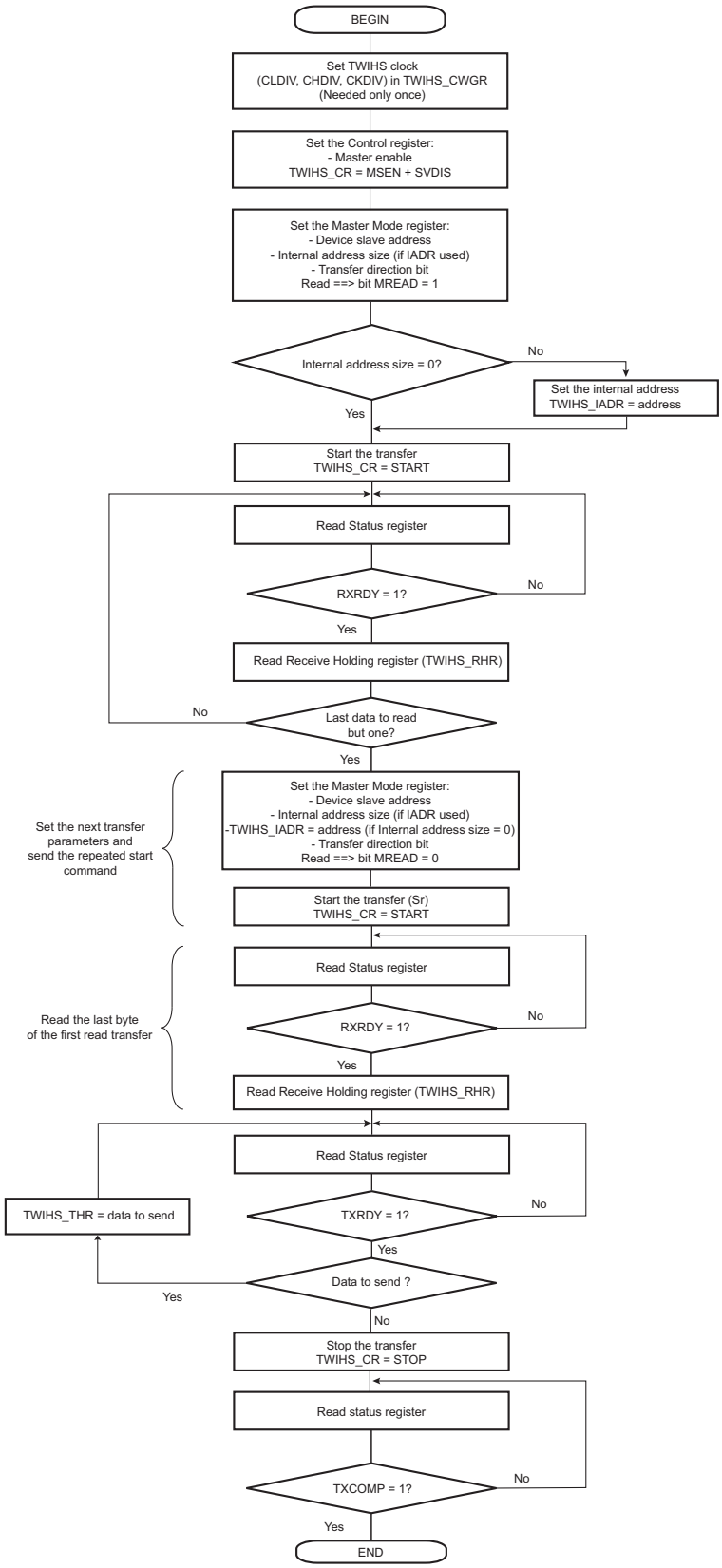
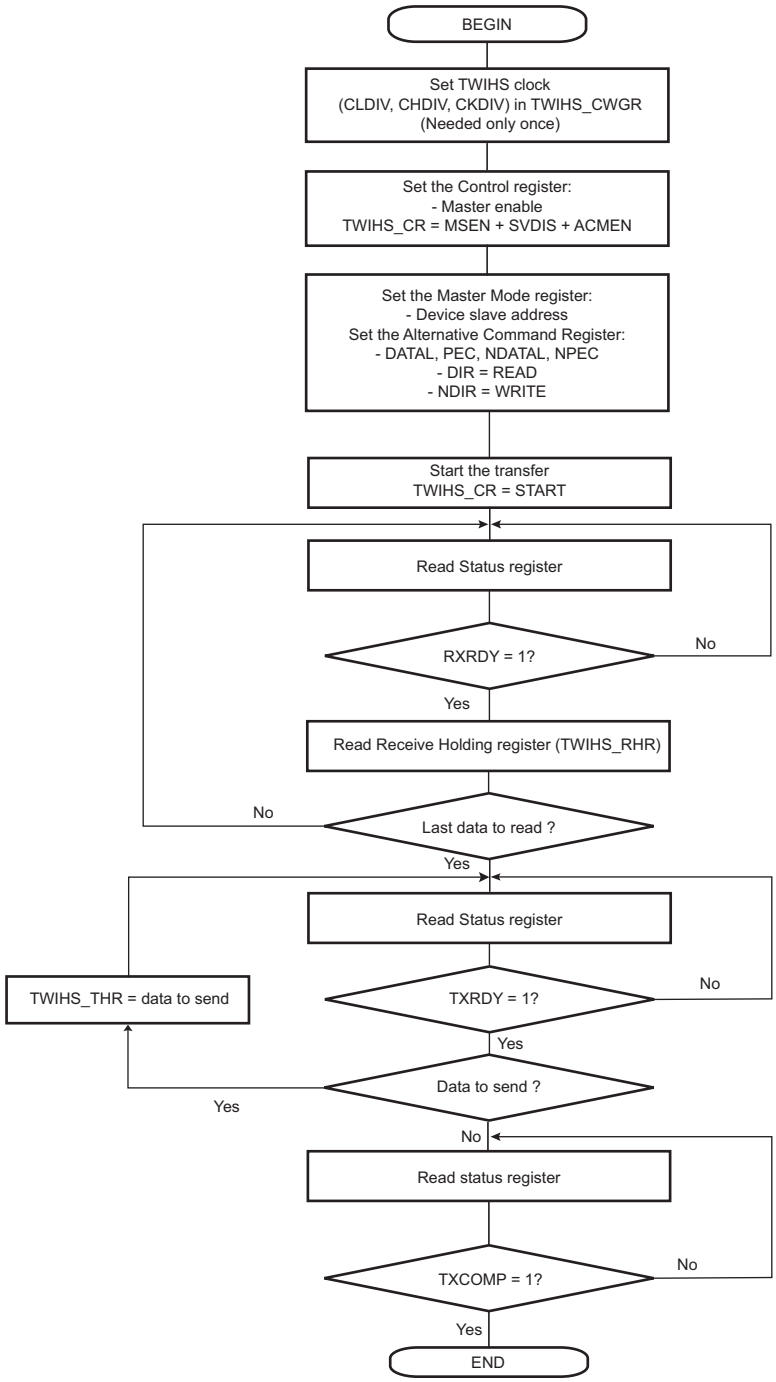


**Figure 43-27. TWIHS Read Operation with Multiple Data Bytes + Write with Alternative Command Mode with PEC**

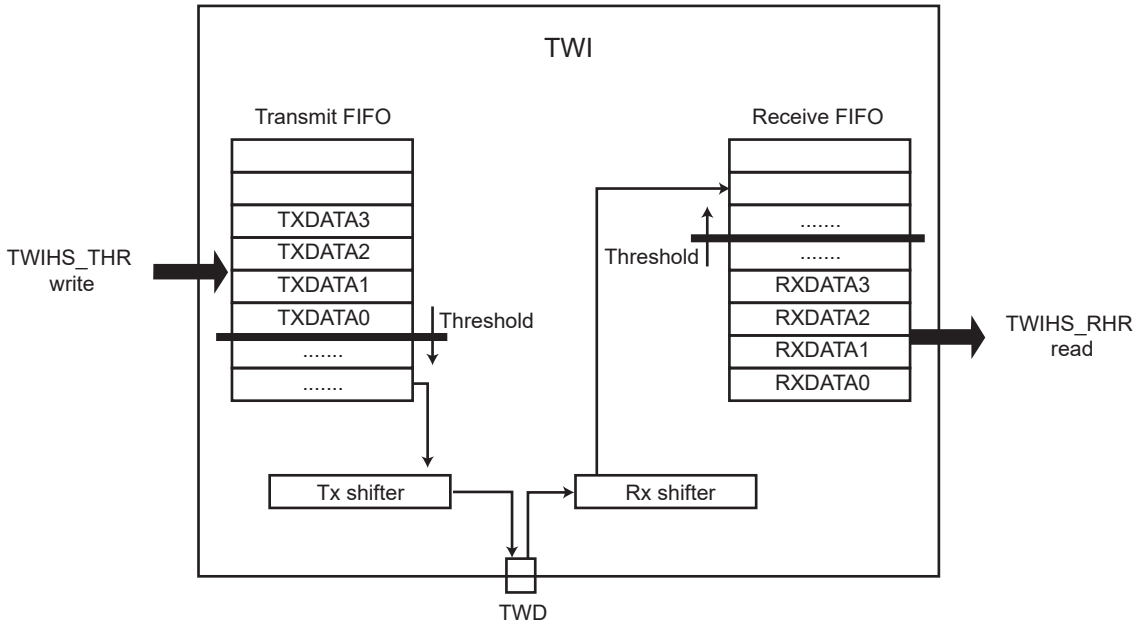



### 43.6.3.15 FIFOs

The TWIHS includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in the TWIHS_CR. It is recommended to disable both Master and Slave modes before enabling or disabling the FIFO (MSDIS and SVDIS bit in TWIHS_CR).

Writing the FIFOEN bit to ‘1’ will enable a 16-data Transmit FIFO and a 16-data Receive FIFO.

**Figure 43-28. FIFOs Block Diagram**

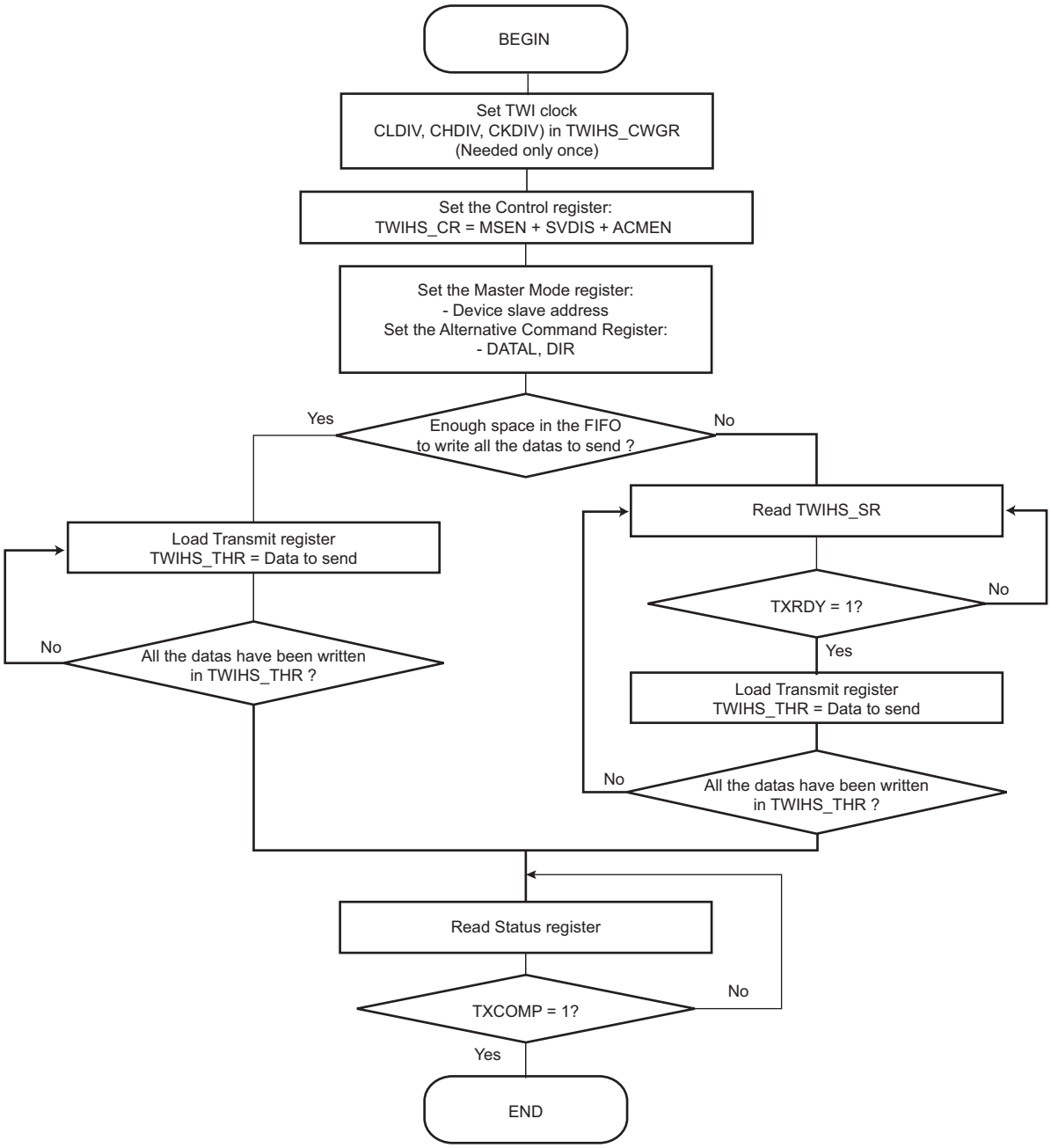



### Sending Data with FIFO Enabled

With the Transmit FIFO enabled, any write access to the TWIHS Transmit Holding Register (TWIHS_THR) brings the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TXRDY flag state to send multiple data without DMAC.

Knowing the number of data to send and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in the TWIHS_THR without checking the TXRDY flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the TWIHS FIFO Level Register (TWIHS_FLR).

**Figure 43-29. Sending Data with FIFO Flowchart**

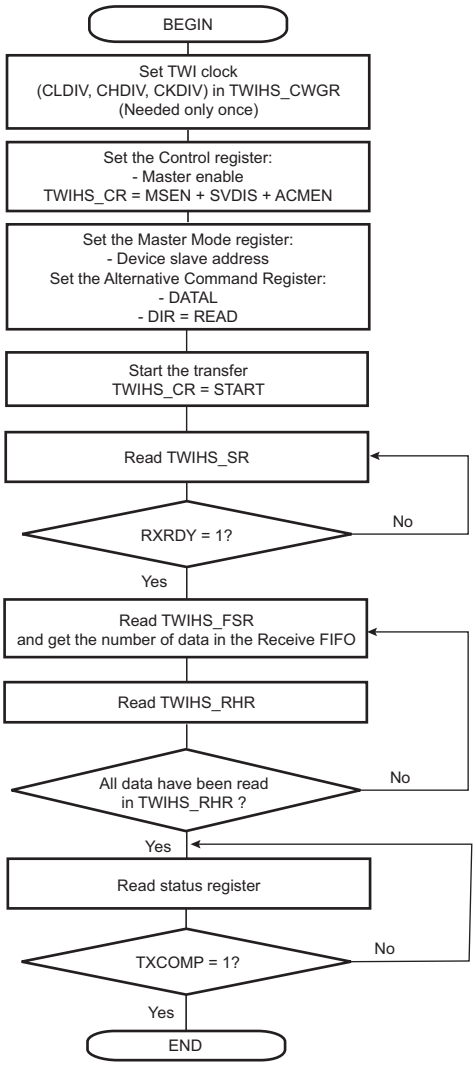



### Receiving Data with FIFO Enabled

With Receive FIFO enabled, any read access on TWIHS_RHR will pull out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RXRDY flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RXRDY flag set to ‘1’), the exact number of data can be checked with the RXFL field in the TWIHS_FLR and all the data read successively in the TWIHS_RHR without checking the RXRDY flag between each access.

**Figure 43-30. Receiving Data with FIFO Flowchart**




### Clearing/Flushing FIFOs

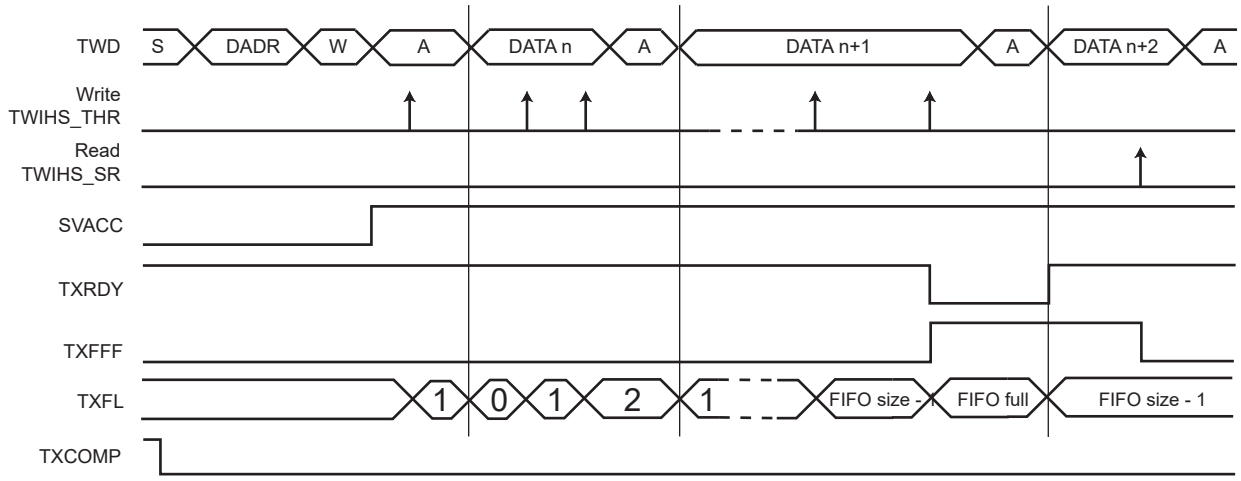
Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in the TWIHS_CR.

### TXRDY and RXRDY Behavior

If FIFOs are enabled, the behavior of the TXRDY and RXRDY flags will be slightly different.

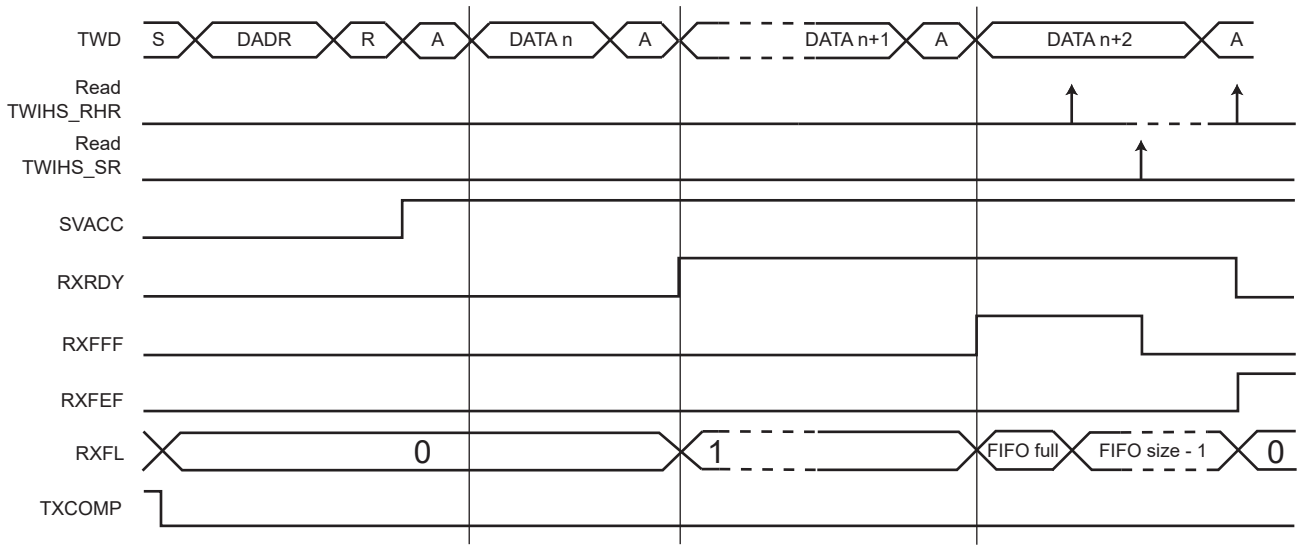
TXRDY will indicate if a data can be written in the Transmit FIFO. By default, the TXRDY flag will then stay at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 43-31. TXRDY in Single Data Mode and TXRDYM = 0x0**




RXRDY will indicate if an unread data is present in the Receive FIFO. By default, the RXRDY flag will then be at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 43-32. RXRDY in Single Data Mode and RXRDYM = 0x0**




TXRDY and RXRDY behavior can be modified using the TXRDYM and RXRDYM fields in the TWIHS FIFO Mode Register (TWIHS_FMR). In Single Data mode, there is no need to modify the TXRDY and RXRDY behavior; however, it may be useful in Multiple Data Mode.

### Master Multiple Data Mode

When the FIFOs are enabled, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write/read up to four data in one TWIHS_THR/TWIHS_RHR register access.

The number of data to write/read is defined by the size of the register access. If the access is a byte size register access, only one data will be written/read; if the access is a halfword-size register access, then two data will be written/read; finally, if the access is a word-size register access, then four data will be written/read.

Written/Read data are always right-aligned, as shown in Section 43.7.14 ”TWIHS Receive Holding Register (FIFO Enabled)” and Section 43.7.17 ”TWIHS Transmit Holding Register (FIFO Enabled)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six TWIHS_THR-byte write accesses.
- Perform three TWIHS_THR-halfword write accesses.
- Perform one TWIHS_THR-word write access and one TWIHS_THR halfword write access.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six TWIHS_RHR-byte read accesses.
- Perform three TWIHS_RHR-halfword read accesses.
- Perform one TWIHS_RHR-word read access and one TWIHS_RHR-halfword read access.

This mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

- TXRDY and RXRDY Configuration

  In Multiple Data mode, the TXRDYM and RXRDYM fields in the TWIHS_FMR become useful.

  As in Multiple Data mode, it is possible to write several data in the same access it might be useful to configure TXRDY flag behavior to indicate if 1, 2 or 4 data can be written in the FIFO depending on the access to perform on TWIHS_THR.

  If for instance four data are written each time in the TWIHS_THR it might be useful to configure TXRDYM field to 0x2 value so that TXRDY flag will be at ‘1’ only when at least four data can be written in the Transmit FIFO.

  In the same way if four data are read each time in the TWIHS_RHR it might be useful to configure RXRDYM field to 0x2 value so that RXRDY flag will be at ‘1’ only when at least four unread data are in the Receive FIFO.

- DMAC

  If DMAC transfer is used it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

#### Transmit FIFO Lock

If a frame is terminated early due to a not-acknowledge error (NACK flag), SMBus timeout error (TOUT flag) or master code acknowledge error (MACK flag), a lock is set on the Transmit FIFO preventing any new frame from being sent until it is cleared. This allows clearing the FIFO if needed, reset DMAC channels, etc., without any risk.

The LOCK bit in the TWIHS_SR is used to check the state of the Transmit FIFO lock.

The Transmit FIFO lock can be cleared setting the TXFLCLR bit to ‘1’ in the TWIHS_CR.

#### FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

If the Transmit FIFO is full and a write access is performed on the TWIHS_THR, it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in TWIHS_FSR.

In Multiple Data mode, if the number of data written in the TWIHS_THR (according to the register access size) is bigger than the Transmit FIFO free space, it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in TWIHS_FSR.

- Receive FIFO:

In Multiple Data mode, if the number of data read in the TWIHS_RHR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it will generate a Receive FIFO pointer error and set the RXFPTEF flag in TWIHS_FSR.

Pointer error should not happen if FIFO state is checked before writing/reading in TWIHS_THR/TWIHS_RHR. FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a Transmit or Receive pointer error occurs, a software reset must be performed using the SWRST bit in the TWIHS_CR. Note that issuing a software while transmitting might leave a slave in an unknown state holding the TWD line. In such case, a Bus Clear Command will allow to make the slave release the TWD line (the first frame sent afterwards might not be received properly by the slave).

#### FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

The Transmit FIFO threshold can be set using the TXFTHRES field in TWIHS_FMR. Each time the Transmit FIFO goes from the ‘above threshold’ to the ‘equal or below threshold’ state, the TXFTHF flag in TWIHS_FSR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

The Receive FIFO threshold can be set using the RXFTHRES field in TWIHS_FMR. Each time the Receive FIFO goes from the ‘below threshold’ to the ‘equal or above threshold’ state, the RXFTHF flag in TWIHS_FSR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using TWIHS_FIER and TWIHS_FIDR.

#### FIFO Flags

FIFOs come with a set of flags which can be configured each to generate an interrupt through TWIHS_FIER and TWIHS_FIDR.

FIFO flags state can be read in TWIHS_FSR. They are cleared on TWIHS_FSR read.

### 43.6.4 Multimaster Mode

#### 43.6.4.1 Definition

In Multimaster mode, more than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master that has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 43-34.

#### 43.6.4.2 Different Multimaster Modes

Two Multimaster modes may be distinguished:

1. The TWIHS is considered as a master only and is never addressed.
2. The TWIHS may be either a master or a slave and may be addressed.

Note: Arbitration in supported in both Multimaster modes.

#### TWIHS as Master Only

In this mode, the TWIHS is considered as a master only (MSEN is always at one) and must be driven like a master with the ARBLST (Arbitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the user must reinitiate the data transfer.

If starting a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWIHS automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 43-33).

Note: The state of the bus (busy or free) is not indicated in the user interface.

### TWIHS as Master or Slave

The automatic reversal from master to slave is not supported in case of a lost arbitration.

Then, in the case where TWIHS may be either a master or a slave, the user must manage the pseudo Multimaster mode described in the steps below:

1. Program the TWIHS in Slave mode (SADR + MSDIS + SVEN) and perform a slave access (if TWIHS is addressed).
2. If the TWIHS has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program the Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, the TWIHS scans the bus in order to detect if it is busy or free. When the bus is considered free, the TWIHS initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWIHS in Slave mode in case the master that won the arbitration needs to access the TWIHS.
7. If the TWIHS has to be set in Slave mode, wait until the TXCOMP flag is at 1 and then program the Slave mode.

Note: If the arbitration is lost and the TWIHS is addressed, the TWIHS does not acknowledge, even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then the master must repeat SADR.

**Figure 43-33. User Sends Data While the Bus is Busy**

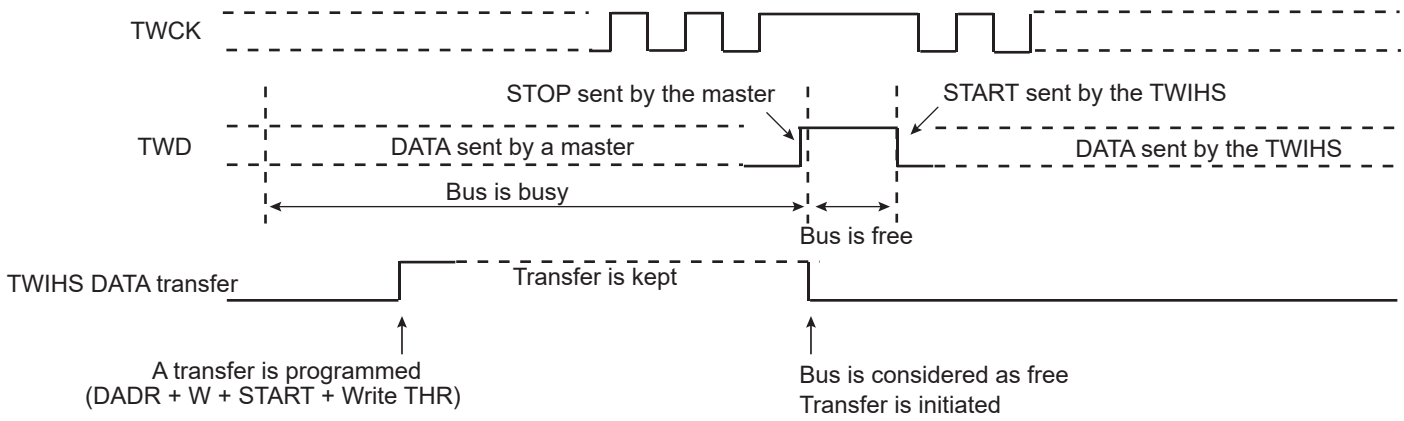
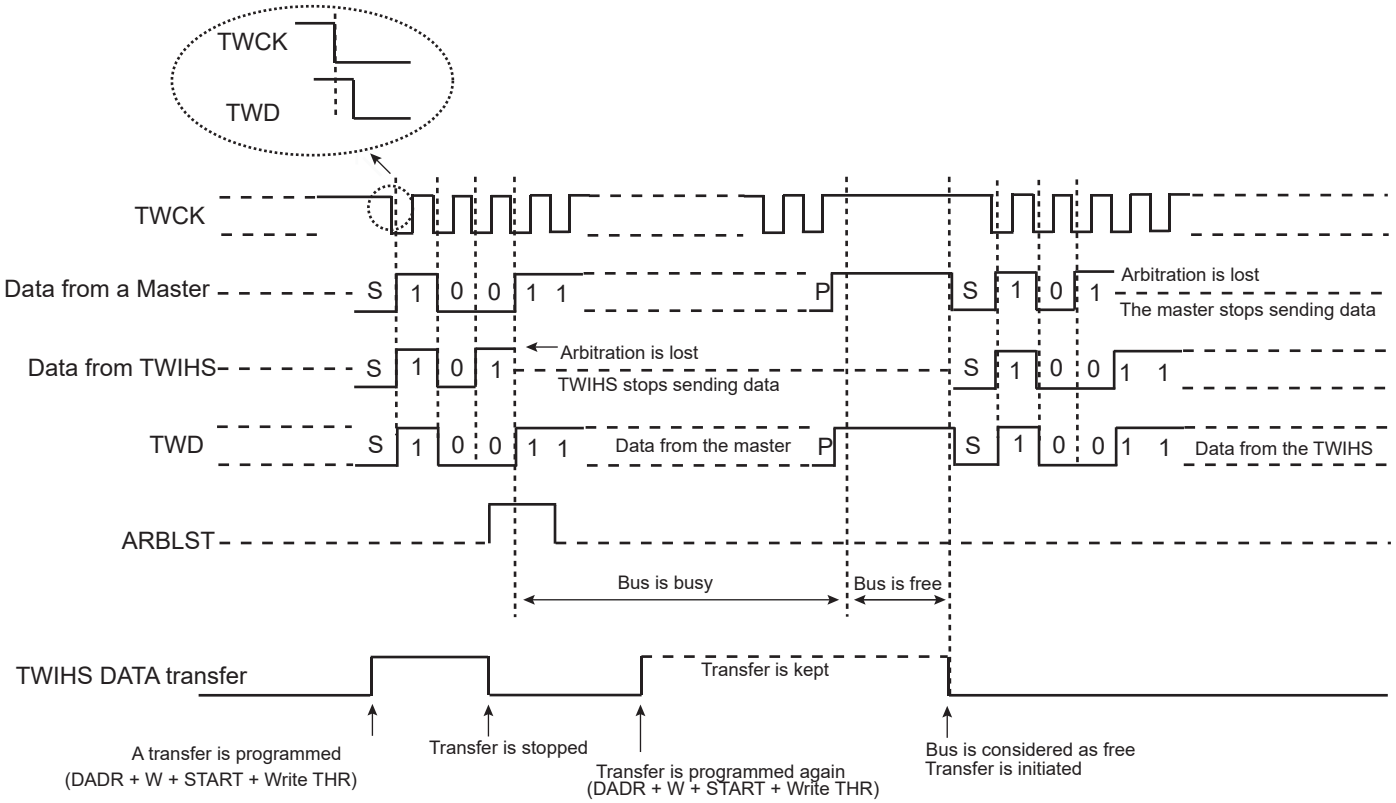
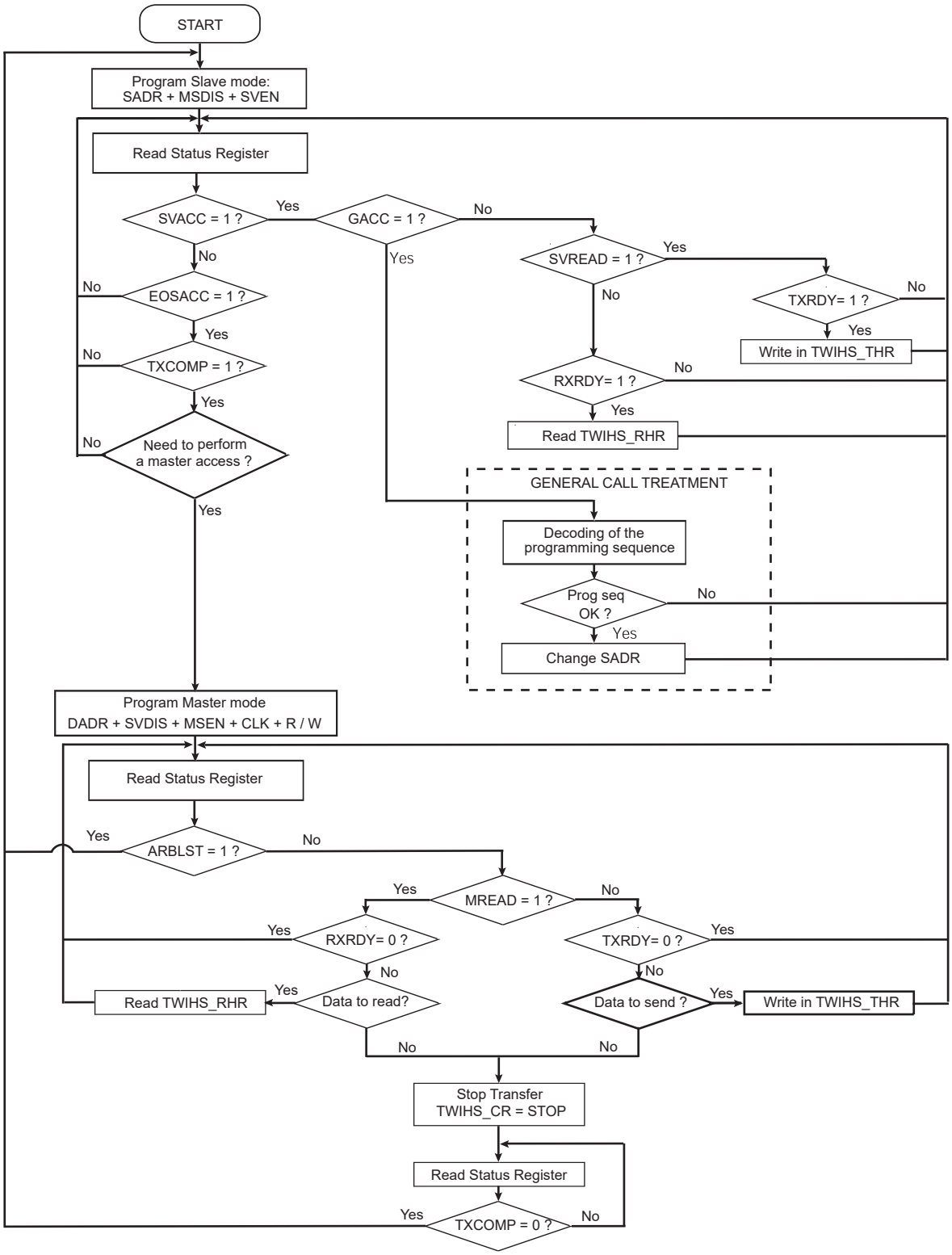


**Figure 43-34. Arbitration Cases**




The flowchart shown in Figure 43-35 gives an example of read and write operations in Multimaster mode.

**Figure 43-35. Multimaster Flowchart**



### 43.6.5 Slave Mode

#### 43.6.5.1 Definition

Slave mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED_START and STOP conditions are always provided by the master).

#### 43.6.5.2 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. TWIHS_SMR.SADR: The slave device address is used in order to be accessed by master devices in Read or Write mode.
2. (Optional) TWIHS_SMR.MASK can be set to mask some SADR address bits and thus allow multiple address matching.
3. TWIHS_CR.MSDIS: Disables the Master mode.
4. TWIHS_CR.SVEN: Enables the Slave mode.

As the device receives the clock, values written in TWIHS_CWGR are ignored.

#### 43.6.5.3 Receiving Data

After a START or REPEATED START condition is detected, and if the address sent by the master matches the slave address programmed in the SADR (Slave Address) field, the SVACC (Slave Access) flag is set and SVREAD (Slave Read) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a REPEATED START is detected. When such a condition is detected, the EOSACC (End Of Slave Access) flag is set.

##### Read Sequence

In the case of a read sequence (SVREAD is high), the TWIHS transfers data written in the TWIHS_THR until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in the TWIHS_THR, TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the internal shifter is empty and the sent data acknowledged or not. If the data is not acknowledged, the NACK flag is set.

Note that a STOP or a REPEATED START always follows a NACK.

To clear the TXRDY flag, first set the bit TWIHS_CR.SVDIS, then set the bit TWIHS_CR.SVEN.

See Figure 43-36.

##### Write Sequence

In the case of a write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in the TWIHS_RHR. RXRDY is reset when reading the TWIHS_RHR.

The TWIHS continues receiving data until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the write sequence, the TXCOMP flag is set and SVACC is reset.

See Figure 43-37.

##### Clock Stretching Sequence

If TWIHS_THR or TWIHS_RHR is not written/read in time, the TWIHS performs a clock stretching.

Clock stretching information is given by the SCLWS (Clock Wait State) bit.

See Figure 43-39 and Figure 43-40.

Note: Clock stretching can be disabled by configuring the SCLWSDIS bit in the TWIHS_SMR. In that case, the UNRE and OVRE flags indicate an underrun (when TWIHS_THR is not filled on time) or an overrun (when TWIHS_RHR is not read on time).

#### General Call

In the case where a GENERAL CALL is performed, the GACC (General Call Access) flag is set.

After GACC is set, the user must interpret the meaning of the GENERAL CALL and decode the new address programming sequence.

See Figure 43-38.

### 43.6.5.4 Data Transfer

#### Read Operation

The Read mode is defined as a data requirement from the master.

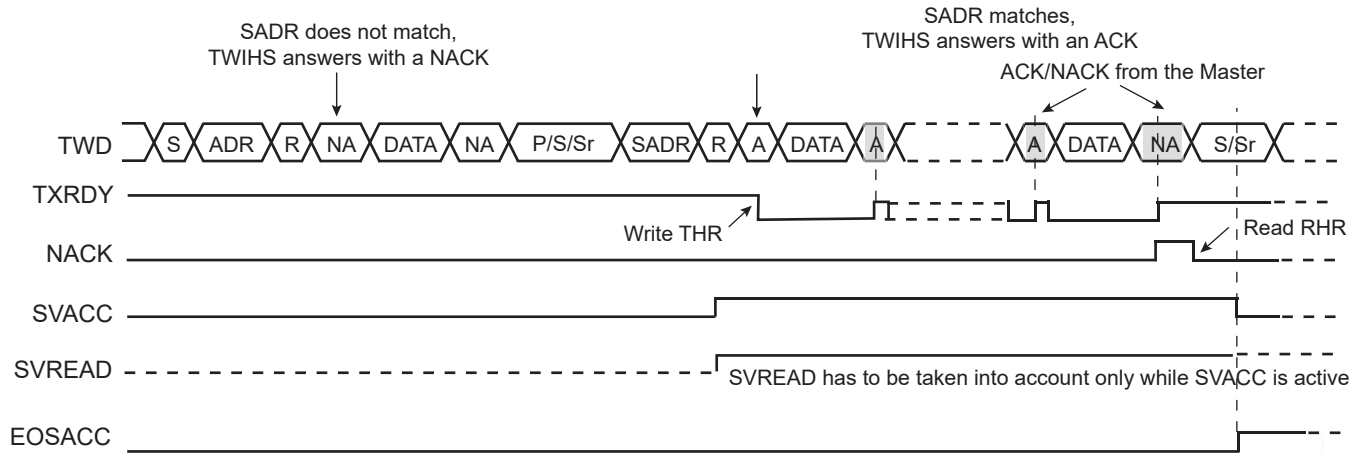
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, the TWIHS continues sending data loaded in the TWIHS_THR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 43-36 describes the read operation.

**Figure 43-36. Read Access Ordered by a Master**




Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. TXRDY is reset when data has been transmitted from TWIHS_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.

#### Write Operation

The Write mode is defined as a data transmission from the master.

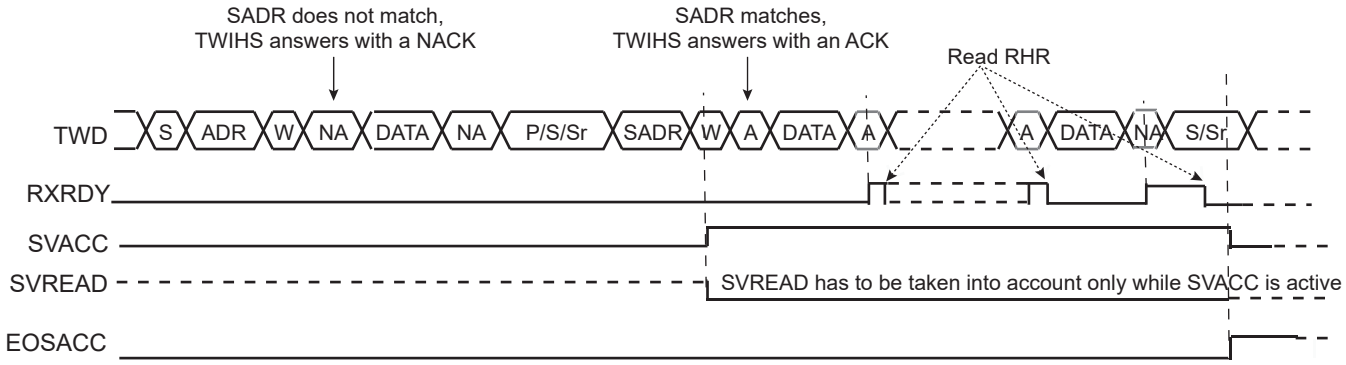
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, the TWIHS stores the received data in the TWIHS_RHR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 43-37 describes the write operation.

**Figure 43-37. Write Access Ordered by a Master**




Notes: 1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the internal shifter to the TWIHS_RHR and reset when this data is read.

### General Call

The general call is performed in order to change the address of the slave.

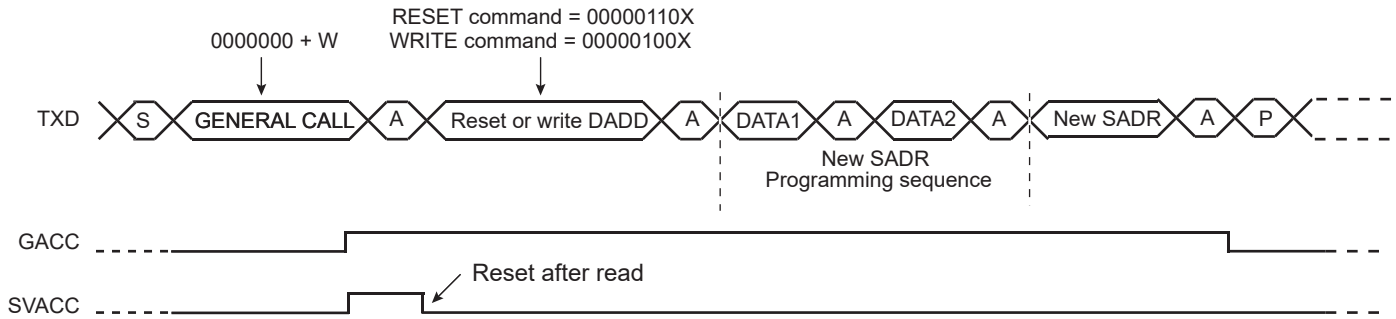
If a GENERAL CALL is detected, GACC is set.

After the detection of general call, decode the commands that follow.

In case of a WRITE command, decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 43-38 describes the general call access.

**Figure 43-38. Master Performs a General Call**




Note: This method enables the user to create a personal programming sequence by choosing the programming bytes and their number. The programming sequence has to be provided to the master.

### Clock Stretching

In both Read and Write modes, it may occur that TWIHS_THR/TWIHS_RHR buffer is not filled/emptied before the transmission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

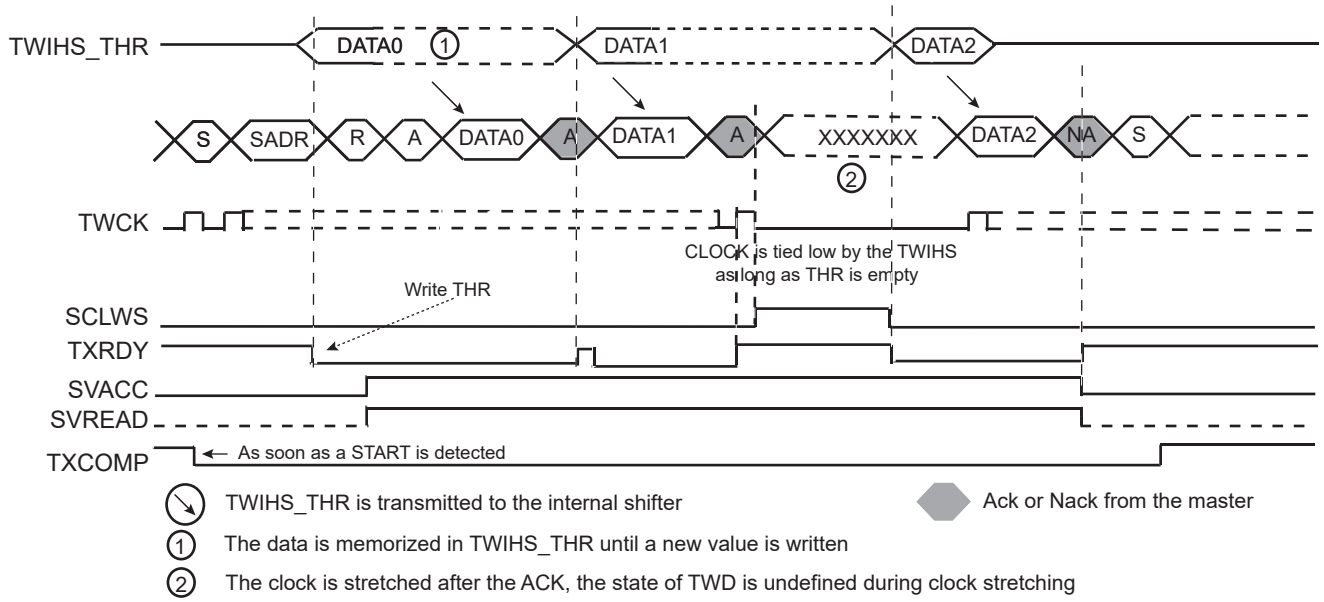
Note: Clock stretching can be disabled by setting the SCLWSDIS bit in the TWIHS_SMR. In that case the UNRE and OVRE flags indicate an underrun (when TWIHS_THR is not filled on time) or an overrun (when TWIHS_RHR is not read on time).

#### Clock Stretching in Read Mode

The clock is tied low if the internal shifter is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the internal shifter is loaded.

Figure 43-39 describes the clock stretching in Read mode.

**Figure 43-39. Clock Stretching in Read Mode**



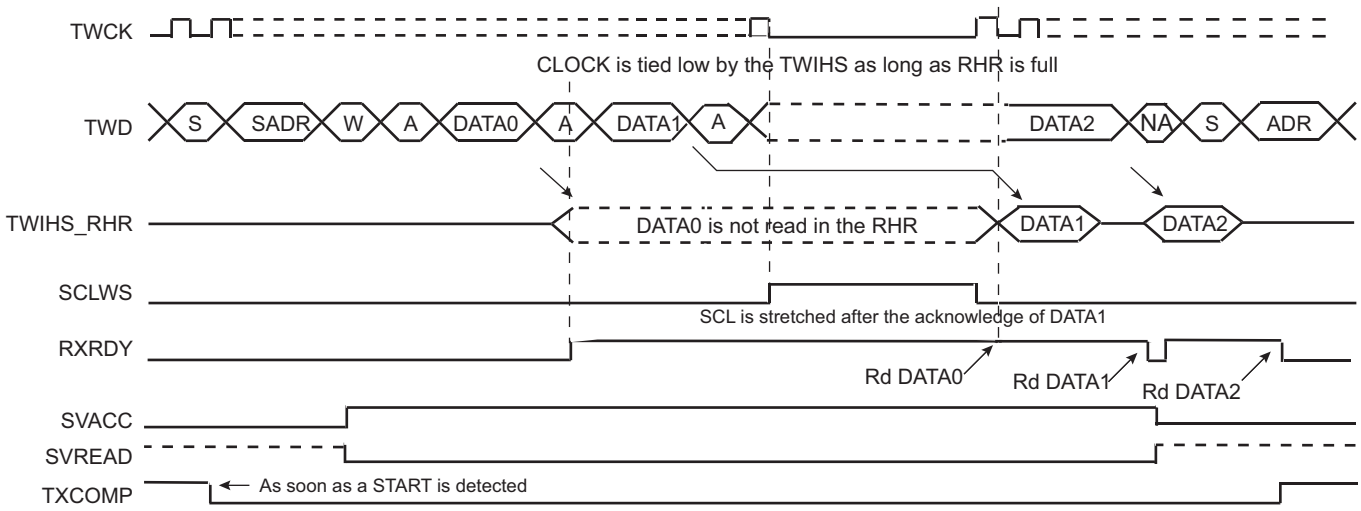

Notes:
1. TXRDY is reset when data has been written in the TWIHS_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock stretching mechanism is started.

### Clock Stretching in Write Mode

The clock is tied low if the internal shifter and the TWIHS_RHR is full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWIHS_RHR is read.

Figure 43-40 describes the clock stretching in Write mode.

**Figure 43-40. Clock Stretching in Write Mode**




Notes:
1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.

2. SCLWS is automatically set when the clock stretching mechanism is started and automatically reset when the mechanism is finished.

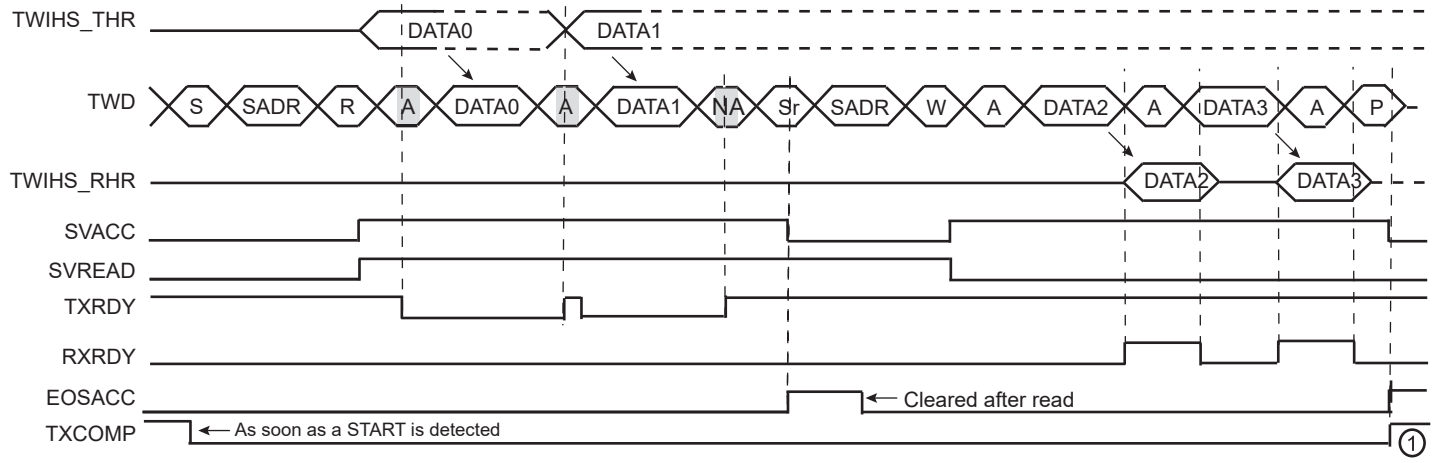
*Reversal after a Repeated Start*

### Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 43-41 describes the REPEATED START and the reversal from Read mode to Write mode.

**Figure 43-41. Repeated Start and Reversal from Read Mode to Write Mode**

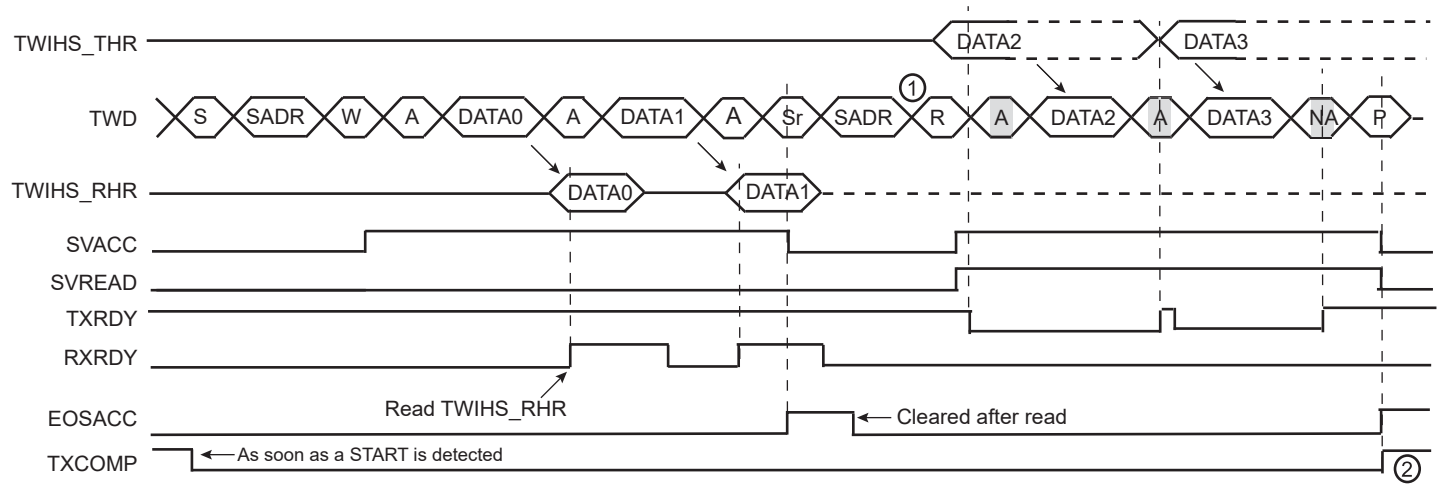



Note: TXCOMP is only set at the end of the transmission. This is because after the REPEATED START, SADR is detected again.

### Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 43-42 describes the REPEATED START and the reversal from Write mode to Read mode.

**Figure 43-42. Repeated Start and Reversal from Write Mode to Read Mode**




Notes:
1. In this case, if TWIHS_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission. This is because after the REPEATED START, SADR is detected again.

#### 43.6.5.5 Using the DMA Controller (DMAC) in Slave Mode

The use of the DMAC significantly reduces the CPU load.

##### Data Transmit with the DMA in Slave Mode

The following procedure shows an example to transmit data with DMA.

1. Initialize the transmit DMA (memory pointers, transfer size, etc).
2. Configure the Slave mode.
3. Enable the DMA.
4. Wait for the DMA status flag indicating that the buffer transfer is complete.
5. Disable the DMA.
6. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

##### Data Receive with the DMA in Slave Mode

The following procedure shows an example to transmit data with DMA where the number of characters to receive is known.

1. Initialize the DMA (channels, memory pointers, size, etc.).
2. Configure the Slave mode.
3. Enable the DMA.
4. Wait for the DMA status flag indicating that the buffer transfer is complete.
5. Disable the DMA.
6. (Only if peripheral clock must be disabled) Wait for the TXCOMP flag to be raised in TWIHS_SR.

#### 43.6.5.6 SMBus Mode

SMBus mode is enabled when a one is written to the SMEN bit in the TWIHS_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

- Only 7-bit addressing can be used.
- The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into the TWIHS_SMBTR.
- Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
- A set of addresses have been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring the TWIHS_CR.

##### Packet Error Checking

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing a one to the PECEN bit in TWIHS_CR will send/check the PEC field in the current transfer. The PEC generator is always updated on every bit transmitted or received, so that PEC handling on the following linked transfers is correct.

In Slave Receiver mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave compares it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave returns an ACK to the master. If the PEC values differ, data was corrupted, and the slave returns a NACK value. The PECERR bit in TWIHS_SR is set automatically if a PEC error occurred.

In Slave Transmitter mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master compares it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the master must take appropriate action.

See Section 43.6.5.10 “Slave Read Write Flowcharts” for detailed flowcharts.

### Timeouts

The TWIHS SMBus Timing Register (TWIHS_SMBTR) configures the SMBus timeout values. If a timeout occurs, the slave leaves the bus. The TOUT bit is also set in TWIHS_SR.

#### 43.6.5.7 High-Speed Slave Mode

High-speed mode is enabled when a one is written to the HSEN bit in the TWIHS_CR. Furthermore, the analog pad filter must be enabled, a one must be written to the PADFEN bit in the TWIHS_FILTR and the FILT bit must be cleared. TWIHS High-speed mode operation is similar to TWIHS operation with the following exceptions:

1. A master code is received first at normal speed before entering High-speed mode period.
2. When TWIHS High-speed mode is active, clock stretching is only allowed after acknowledge (ACK), not-acknowledge (NACK), START (S) or REPEATED START (Sr) (as consequence OVF may happen).

TWIHS High-speed mode allows transfers of up to 3.4 Mbit/s.

The TWIHS slave in High-speed mode requires that the peripheral clock runs at a minimum of 14 MHz if slave clock stretching is enabled (SCLWSDIS bit at ‘0’). If slave clock stretching is disabled (SCLWSDIS bit at ‘1’), the peripheral clock must run at a minimum of 11 MHz (assuming the system has no latency).

Note: When slave clock stretching is disabled, the TWIHS_RHR must always be read before receiving the next data (MASTER write frame). It is strongly recommended to use either the polling method on the RXRDY flag in TWIHS_SR, or the DMA. If the receive is managed by an interrupt, the TWIHS interrupt priority must be set to the right level and its latency minimized to avoid receive overrun.

Note: When slave clock stretching is disabled, the TWIHS_THR must be filled with the first data to send before the beginning of the frame (MASTER read frame). It is strongly recommended to use either the polling method on the TXRDY flag in TWIHS_SR, or the DMA. If the transmit is managed by an interrupt, the TWIHS interrupt priority must be set to the right level and its latency minimized to avoid transmit underrun.

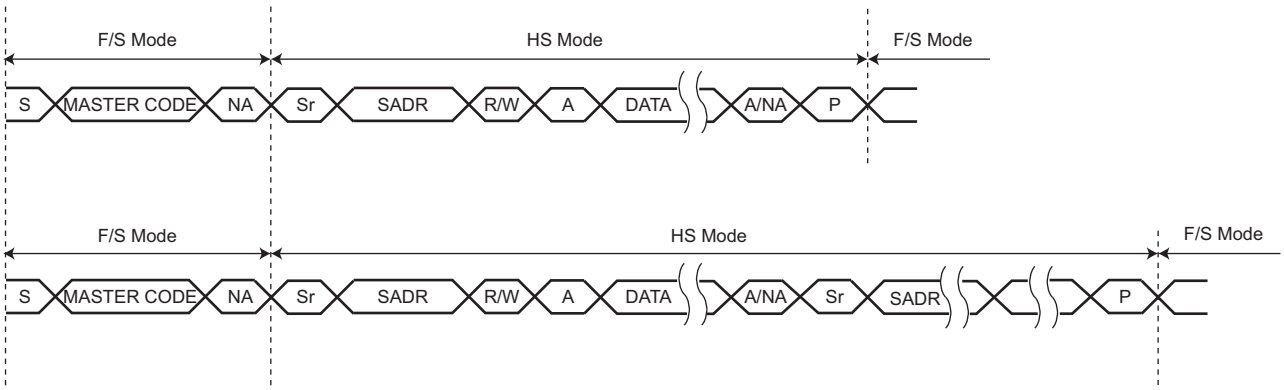
### Read/Write Operation

A TWIHS high-speed frame always begins with the following sequence:

1. START condition (S)
2. Master Code (0000 1XXX)
3. Not-acknowledge (NACK)

When the TWIHS is programmed in Slave mode and TWIHS High-speed mode is activated, master code matching is activated and internal timings are set to match the TWIHS High-speed mode requirements.

**Figure 43-43. High-Speed Mode Read/Write**




### Usage

TWIHS High-speed mode usage is the same as the standard TWIHS (See Section 43.6.3.14 “Read/Write Flowcharts”).

#### 43.6.5.8 Alternative Command

In Slave mode, Alternative Command mode is useful when SMBus mode is enabled to send or check the PEC byte.

Alternative Command mode is enabled by setting the ACMEN bit of the TWIHS Control Register and the transfer is configured in TWIHS_ACR.

For a combined transfer with PEC, only the NPEC bit in TWIHS_ACR must be set as the PEC byte is sent once at the end of the frame.

See Section 43.6.5.10 “Slave Read Write Flowcharts” for detailed flowcharts.

#### 43.6.5.9 Asynchronous Partial Wakeup (SleepWalking)

The TWIHS includes an asynchronous start condition detector. It is capable of waking the device up from a Sleep mode upon an address match (and optionally an additional data match), including Sleep modes where the TWIHS peripheral clock is stopped.

After detecting the START condition on the bus, the TWIHS stretches TWCK until the TWIHS peripheral clock has started. The time required for starting the TWIHS depends on which Sleep mode the device is in. After the TWIHS peripheral clock has started, the TWIHS releases its TWCK stretching and receives one byte of data (slave address) on the bus. At this time, only a limited part of the device, including the TWIHS module, receives a clock, thus saving power. If the address phase causes a TWIHS address match (and, optionally, if the first data byte causes data match as well), the entire device is woken up and normal TWIHS address matching actions are performed. Normal TWIHS transfer then follows. If the TWIHS is not addressed (or if the optional data match fails), the TWIHS peripheral clock is automatically stopped and the device returns to its original Sleep mode.

The TWIHS has the capability to match on more than one address. The SADR1EN, SADR2EN and SADR3EN bits in TWIHS_SMR enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in the TWIHS_SWMR. The SleepWalking matching process can be extended to the first received data byte if DATAMEN bit in TWIHS_SMR is set and, in this case, a complete matching includes address matching and first received data matching. The field DATAM in TWIHS_SWMR configures the data to match on the first received byte.

When the system is in Active mode and the TWIHS enters Asynchronous Partial Wakeup mode, the flag SVACC must be programmed as the unique source of the TWIHS interrupt and the data match comparison must be disabled.

When the system exits Wait mode as the result of a matching condition, the SVACC flag is used to determine if the TWIHS is the source of exit.

**Figure 43-44. Address Match Only (Data Matching Disabled)**

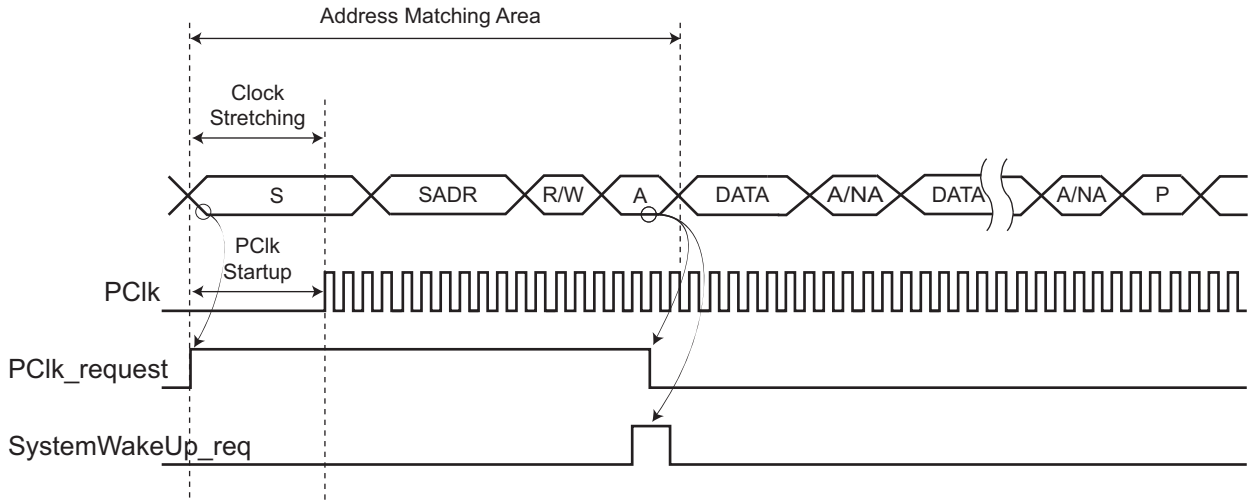



**Figure 43-45. No Address Match (Data Matching Disabled)**

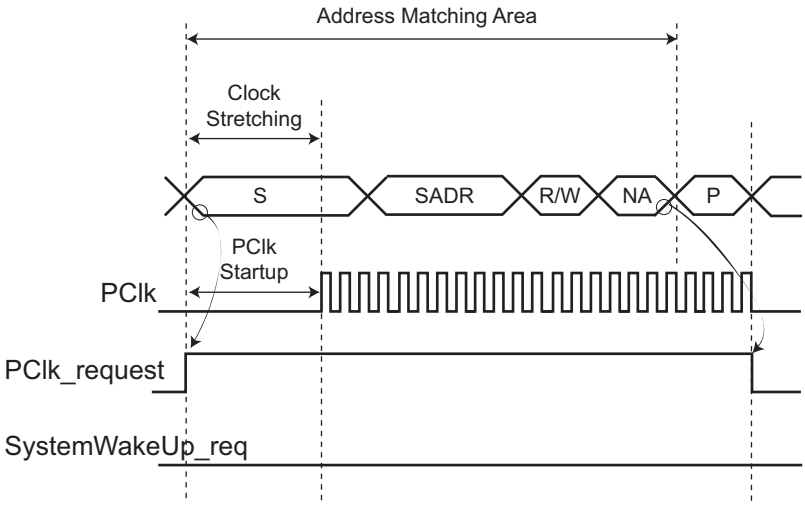


**Figure 43-46. Address Match and Data Match (Data Matching Enabled)**

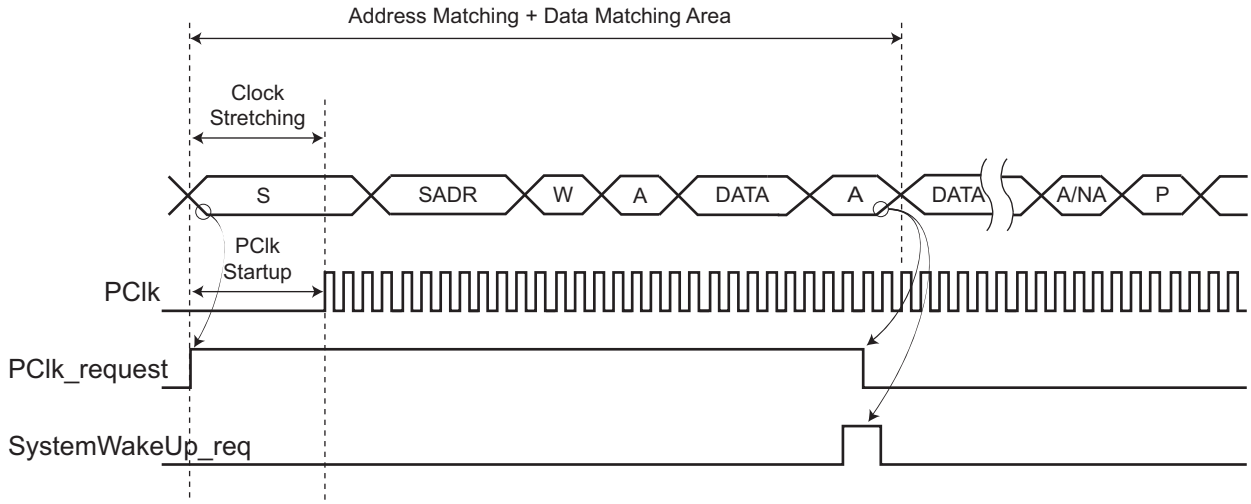
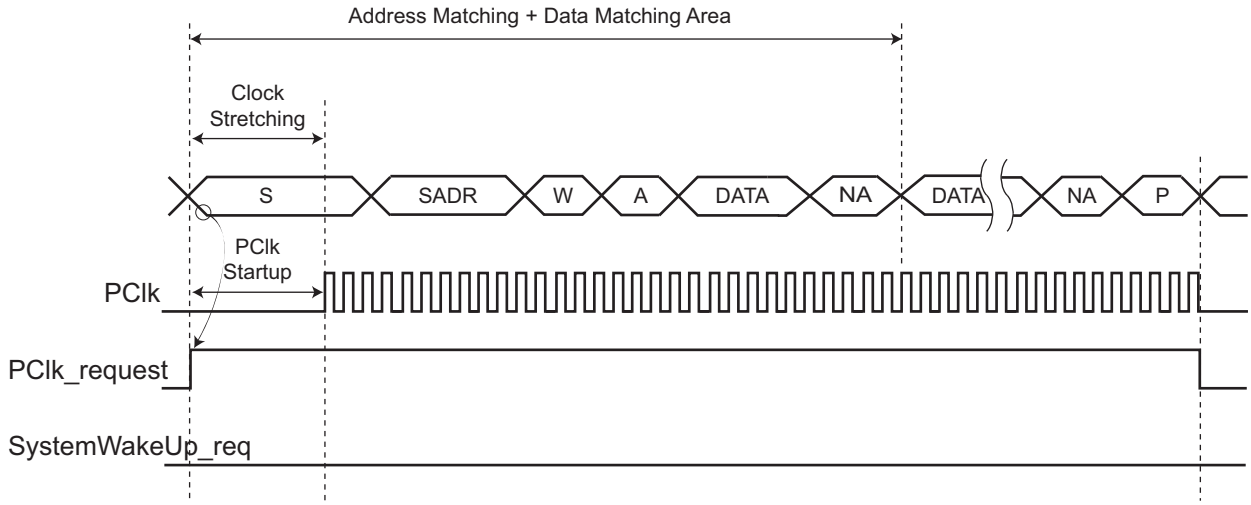



**Figure 43-47. Address Match and No Data Match (Data Matching Enabled)**




### 43.6.5.10 Slave Read Write Flowcharts

The flowchart shown in Figure 43-48 gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that TWIHS_IER be configured first.

**Figure 43-48. Read Write Flowchart in Slave Mode**

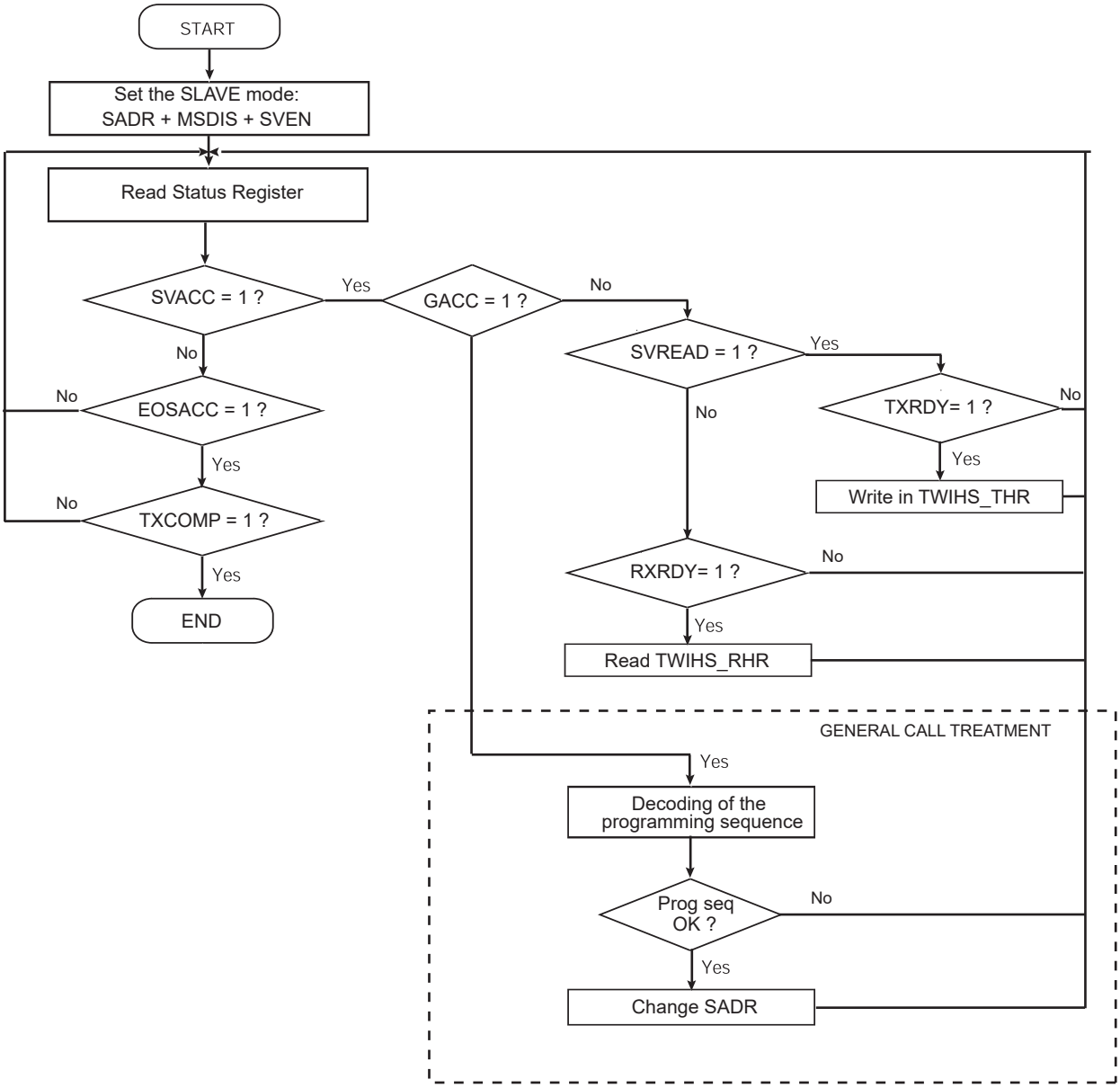


**Figure 43-49. Read Write Flowchart in Slave Mode with SMBus PEC**

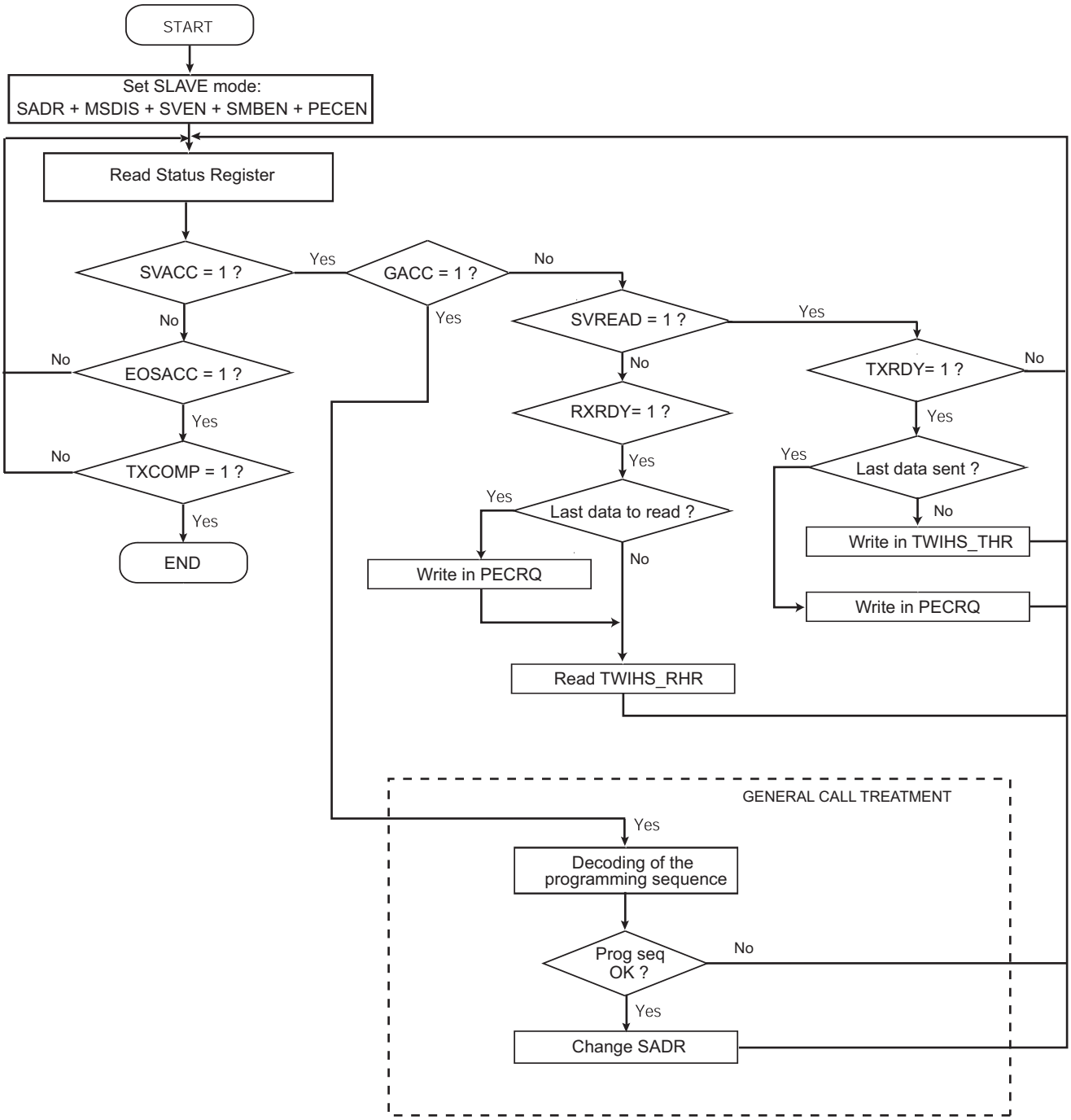
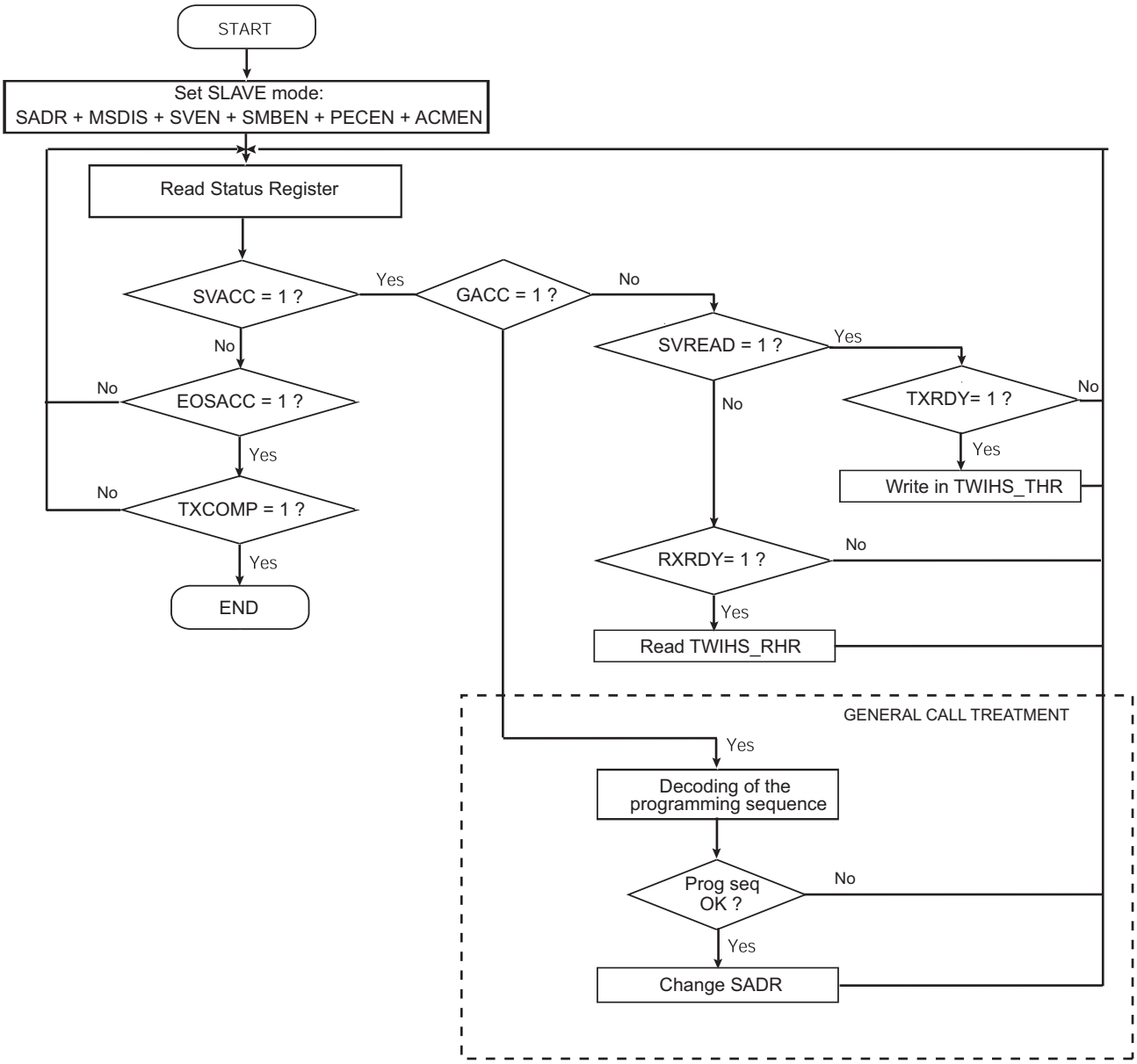


**Figure 43-50. Read Write Flowchart in Slave Mode with SMBus PEC and Alternative Command Mode**

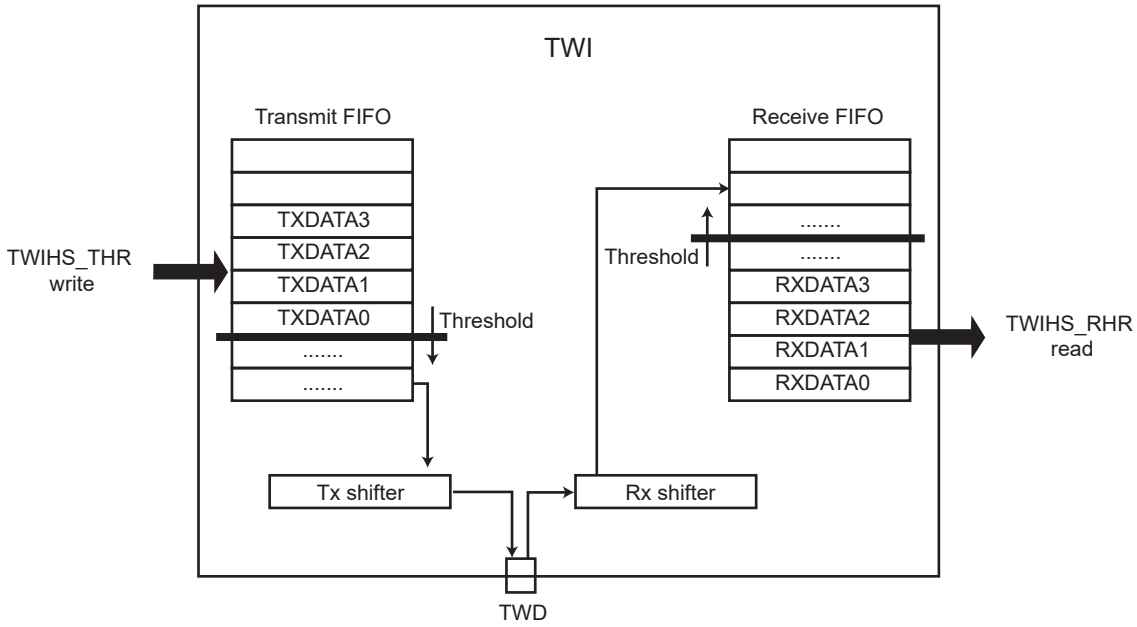



### 43.6.5.11 FIFOs

The TWIHS includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in the TWIHS_CR. It is recommended to disable both Master and Slave modes before enabling or disabling the FIFO (MSDIS and SVDIS bit in TWIHS_CR).

Writing the FIFOEN bit to ‘1’ will enable a 16-data Transmit FIFO and a 16-data Receive FIFO.

**Figure 43-51. FIFOs Block Diagram**




### Sending/Receiving Data with FIFO Enabled

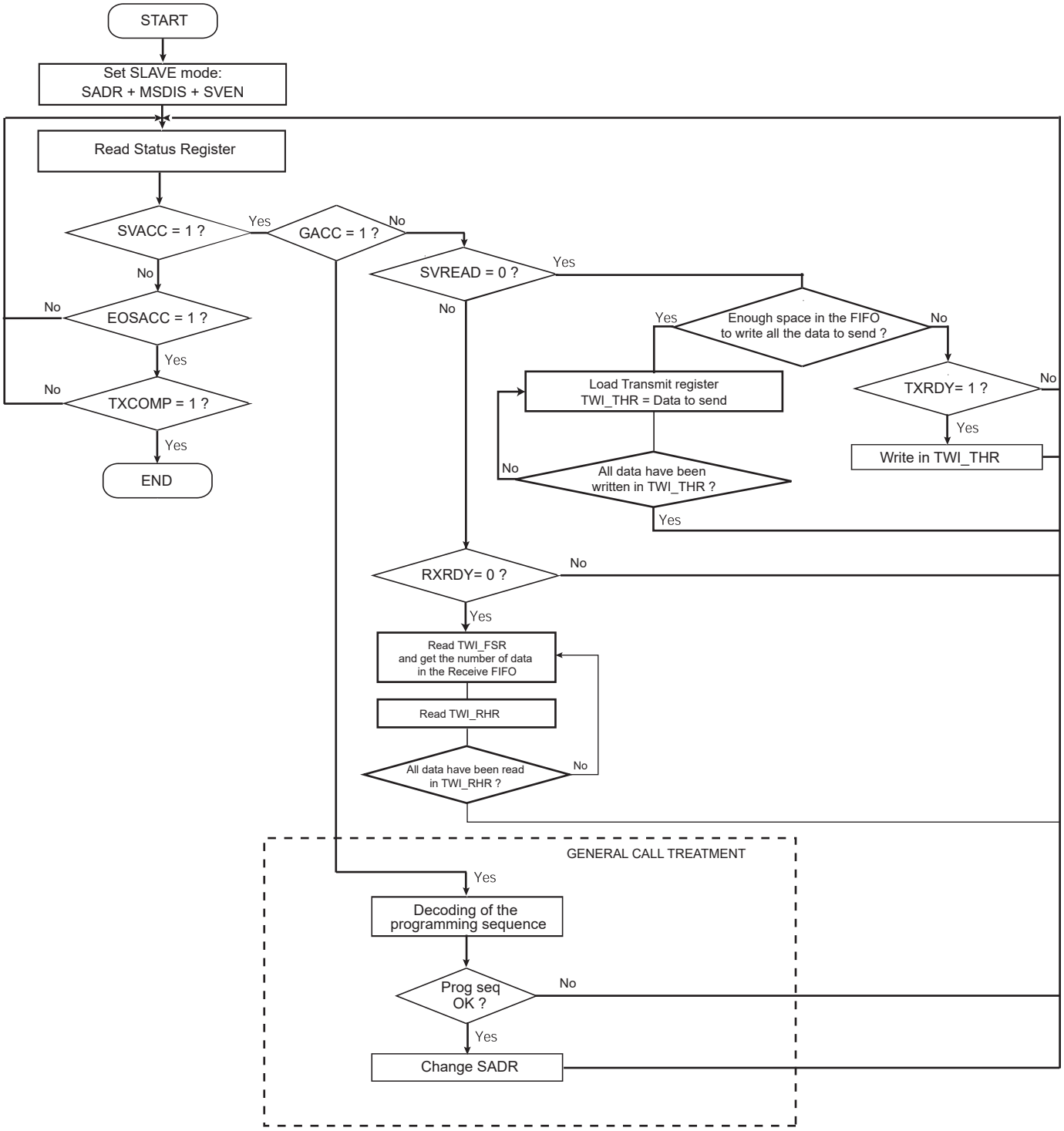
With the Transmit FIFO enabled, any write access to the TWIHS_THR will bring the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TXRDY flag state to send multiple data without DMAC.

Knowing the number of data to send and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in the TWIHS_THR without checking the TXRDY flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the TWIHS_FLR.

With Receive FIFO enabled, any read access on TWIHS_RHR will pull out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RXRDY flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RXRDY flag set to ‘1’), the exact number of data can be checked with the RXFL field in the TWIHS_FLR and all the data read successively in the TWIHS_RHR without checking the RXRDY flag between each access.

**Figure 43-52. Sending/Receiving Data with FIFO Flowchart**




### Clearing/Flushing FIFOs

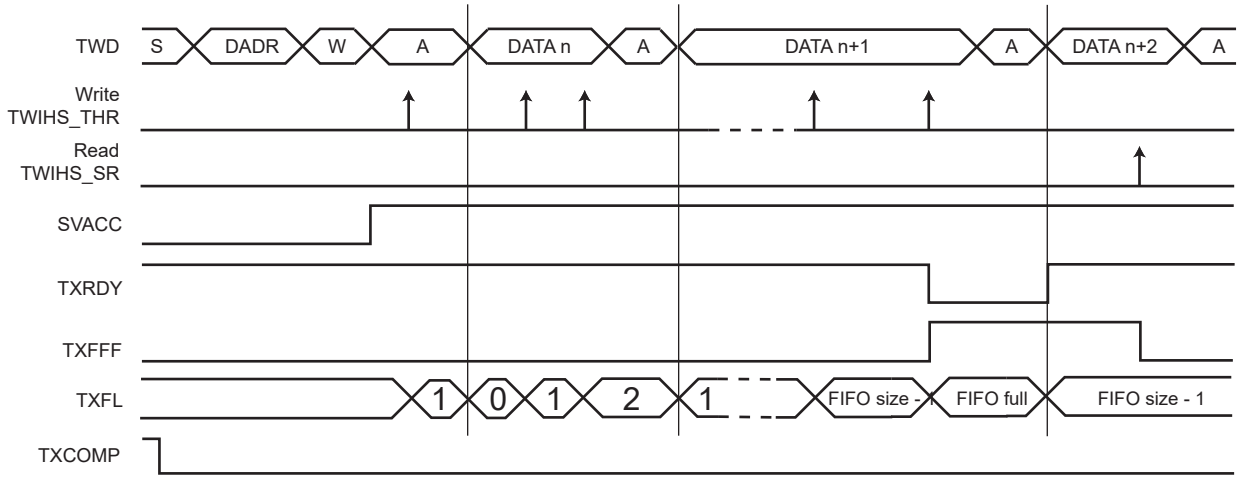
Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in the TWIHS_CR.

### TXRDY and RXRDY Behavior

If FIFOs are enabled, the behavior of the TXRDY and RXRDY flags will be slightly different.

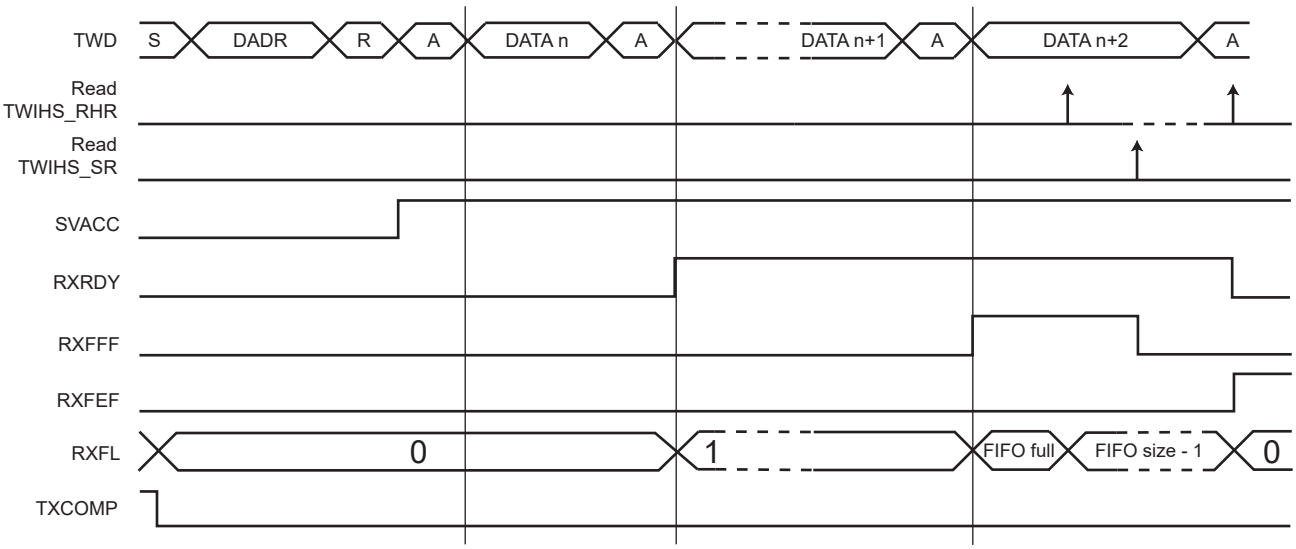
TXRDY will indicate if a data can be written in the Transmit FIFO. By default, the TXRDY flag will then stay at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 43-53. TXRDY in Single Data Mode and TXRDYM = 0x0**




RXRDY will indicate if an unread data is present in the Receive FIFO. By default, the RXRDY flag will then be at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 43-54. RXRDY in Single Data Mode and RXRDYM = 0x0**




TXRDY and RXRDY behavior can be modified using the TXRDYM and RXRDYM fields in the TWIHS_FMR. In Single Data mode, there is no need to modify the TXRDY and RXRDY behavior; however, it may be useful in Multiple Data mode.

### Slave Multiple Data Mode

When the FIFOs are enabled, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write/read up to four data in one TWIHS_THR/TWIHS_RHR register access.

The number of data to write/read is defined by the size of the register access. If the access is a byte-size register access, only one data will be written/read; if the access is a halfword-size register access, then two data will be written/read and finally, if the access is a word-size register access, then four data will be written/read.

Written/Read data are always right-aligned, as shown in Section 43.7.14 ”TWIHS Receive Holding Register (FIFO Enabled)” and Section 43.7.17 ”TWIHS Transmit Holding Register (FIFO Enabled)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six TWIHS_THR-byte write accesses.
- Perform three TWIHS_THR-halfword write accesses.
- Perform one TWIHS_THR-word write access and one TWIHS_THR-halfword write access.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six TWIHS_RHR-byte read accesses.
- Perform three TWIHS_RHR-halfword read accesses.
- Perform one TWIHS_RHR-word read access and one TWIHS_RHR-halfword read access.

Multiple Data mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

- TXRDY and RXRDY configuration

  In Multiple Data mode, the TXRDYM and RXRDYM fields in TWIHS_FMR become useful.

  As in Multiple Data mode, it is possible to write several data in the same access; it might be useful to configure the TXRDY flag behavior to indicate if 1, 2 or 4 data can be written in the FIFO depending on the access to perform on TWIHS_THR.

  If for instance four data are written each time in the TWIHS_THR it might be useful to configure the TXRDYM field to 0x2 value so that the TXRDY flag will be at ‘1’ only when at least four data can be written in the Transmit FIFO.

  In the same way, if four data are read each time in the TWIHS_RHR, it might be useful to configure the RXRDYM field to 0x2 value so that the RXRDY flag will be at ‘1’ only when at least four unread data are in the Receive FIFO.

- DMAC

  If DMAC transfer is used, it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

#### Transmit FIFO Lock

If a frame is terminated early due to a not-acknowledge error (NACK flag), SMBus timeout error (TOUT flag) or master code acknowledge error (MACK flag), a lock is set on the Transmit FIFO preventing any new frame from being sent until it is cleared. This allows clearing the FIFO if needed, reset DMAC channels, etc., without any risk.

The LOCK bit in the TWIHS_SR is used to check the state of the Transmit FIFO lock.

The Transmit FIFO lock can be cleared by setting the TXFLCLR bit to ‘1’ in the TWIHS_CR.

#### FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

  If the Transmit FIFO is full and a write access is performed on the TWIHS_THR it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in TWIHS_FSR.

  In Multiple Data mode, if the number of data written in the TWIHS_THR (according to the register access size) is bigger than the Transmit FIFO free space, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in TWIHS_FSR.

- Receive FIFO:

  In Multiple Data mode, if the number of data read in the TWIHS_RHR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it generates a Receive FIFO pointer error and sets the RXFPTEF flag in TWIHS_FSR.

  Pointer error should not happen if the FIFO state is checked before writing/reading in the TWIHS_THR/TWIHS_RHR registers. The FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a Transmit or Receive pointer error occurs, a software reset must be performed using the SWRST bit in the TWIHS_CR. Note that issuing a software while transmitting might leave a slave in an unknown state holding the TWD line. In such case, a Bus Clear Command will allow to make the slave release the TWD line (the first frame sent afterwards might not be received properly by the slave).

#### FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

  The Transmit FIFO threshold can be set using the TXFTHRES field in TWIHS_FMR. Each time the Transmit FIFO goes from the ‘above threshold’ to the ‘equal or below threshold’ state, the TXFTHF flag in TWIHS_FSR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

  The Receive FIFO threshold can be set using the RXFTHRES field in TWIHS_FMR. Each time the Receive FIFO goes from the ‘below threshold’ to the ‘equal or above threshold’ state, the RXFTHF flag in TWIHS_FSR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using TWIHS_FIER and TWIHS_FIDR.

#### FIFO Flags

FIFOs come with a set of flags which can be configured each to generate an interrupt through TWIHS_FIER and TWIHS_FIDR.

FIFO flags state can be read in TWIHS_FSR. They are cleared on TWIHS_FSR read.

### 43.6.6 TWIHS Comparison Function on Received Character

The comparison function differs if asynchronous partial wakeup (SleepWalking) is enabled or not.

If asynchronous partial wakeup is disabled (see the section “Power Management Controller (PMC)”), the TWIHS can extend the address matching on up to three slave addresses. The SADR1EN, SADR2EN and SADR3EN bits in TWIHS_SMR enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in the TWIHS_SWMR. The DATAMEN bit in the TWIHS_SMR has no effect.

The SVACC bit is set when there is a comparison match with the received slave address.

### 43.6.7 Register Write Protection

To prevent any single software error from corrupting TWIHS behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the TWIHS Write Protection Mode Register (TWIHS_WPMR).

If a write access to a write-protected register is detected, the WPVS bit in the TWIHS Write Protection Status Register (TWIHS_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading TWIHS_WPSR.

The following registers can be write-protected:

- TWIHS Slave Mode Register
- TWIHS Clock Waveform Generator Register
- TWIHS SMBus Timing Register
- TWIHS SleepWalking Matching Register

## 43.7 Two-wire Interface High Speed (TWIHS) User Interface

**Table 43-7. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | TWIHS_CR | Write-only | – |
| 0x04 | Master Mode Register | TWIHS_MMR | Read/Write | 0x00000000 |
| 0x08 | Slave Mode Register | TWIHS_SMR | Read/Write | 0x00000000 |
| 0x0C | Internal Address Register | TWIHS_IADR | Read/Write | 0x00000000 |
| 0x10 | Clock Waveform Generator Register | TWIHS_CWGR | Read/Write | 0x00000000 |
| 0x14–0x1C | Reserved | – | – | – |
| 0x20 | Status Register | TWIHS_SR | Read-only | 0x0300F009 |
| 0x24 | Interrupt Enable Register | TWIHS_IER | Write-only | – |
| 0x28 | Interrupt Disable Register | TWIHS_IDR | Write-only | – |
| 0x2C | Interrupt Mask Register | TWIHS_IMR | Read-only | 0x00000000 |
| 0x30 | Receive Holding Register | TWIHS_RHR | Read-only | 0x00000000 |
| 0x34 | Transmit Holding Register | TWIHS_THR | Write-only | 0x00000000 |
| 0x38 | SMBus Timing Register | TWIHS_SMBTR | Read/Write | 0x00000000 |
| 0x3C | Reserved | – | – | – |
| 0x40 | Alternative Command Register | TWIHS_ACR | Read/Write | – |
| 0x44 | Filter Register | TWIHS_FILTR | Read/Write | 0x00000000 |
| 0x48 | Reserved | – | – | – |
| 0x4C | SleepWalking Matching Register | TWIHS_SWMR | Read/Write | 0x00000000 |
| 0x50 | FIFO Mode Register | TWIHS_FMR | Read/Write | 0x00000000 |
| 0x54 | FIFO Level Register | TWIHS_FLR | Read-only | 0x00000000 |
| 0x58–0x5C | Reserved | – | – | – |
| 0x60 | FIFO Status Register | TWIHS_FSR | Read-only | 0x00000000 |
| 0x64 | FIFO Interrupt Enable Register | TWIHS_FIER | Write-only | – |
| 0x68 | FIFO Interrupt Disable Register | TWIHS_FIDR | Write-only | – |
| 0x6C | FIFO Interrupt Mask Register | TWIHS_FIMR | Read-only | 0x00000000 |
| 0x70–0xCC | Reserved | – | – | – |
| 0xD0 | Reserved | – | – | – |
| 0xD4–0xE0 | Reserved | – | – | – |
| 0xE4 | Write Protection Mode Register | TWIHS_WPMR | Read/Write | 0x00000000 |
| 0xE8 | Write Protection Status Register | TWIHS_WPSR | Read-only | 0x00000000 |
| 0xEC–0xFC[1] | Reserved | – | – | – |
| 0x100–0x128 | Reserved | – | – | – |

Note: 1. All unlisted offset values are considered as “reserved”.

### 43.7.1 TWIHS Control Register

Name: TWIHS_CR

**Address:** 0xF8028000 (0), 0xFC028000 (1)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | FIFODIS | FIFOEN | – | LOCKCLR | – | THRCLR |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | ACMDIS | ACMEN |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CLEAR | PECRQ | PECDIS | PECEN | SMBDIS | SMBEN | HSDIS | HSEN |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | QUICK | SVDIS | SVEN | MSDIS | MSEN | STOP | START |

- **START: Send a START Condition**

0: No effect.

1: A frame beginning with a START bit is transmitted according to the features defined in the TWIHS Master Mode Register (TWIHS_MMR).

This action is necessary when the TWIHS peripheral needs to read data from a slave. When configured in Master mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWIHS_THR).

- **STOP: Send a STOP Condition**

0: No effect.

1: STOP condition is sent just after completing the current byte transmission in Master Read mode.

  – In single data byte master read, both START and STOP must be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In Master Read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWIHS Master Mode Enabled**

0: No effect.

1: Enables the Master mode (MSDIS must be written to 0).

Note: Switching from Slave to Master mode is only permitted when TXCOMP = 1.

- **MSDIS: TWIHS Master Mode Disabled**

0: No effect.

1: The Master mode is disabled, all pending data is transmitted. The shifter and holding characters (if it contains data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SVEN: TWIHS Slave Mode Enabled**

0: No effect.

1: Enables the Slave mode (SVDIS must be written to 0).

Note: Switching from Master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWIHS Slave Mode Disabled**

0: No effect.

1: The Slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBus Quick Command**

0: No effect.

1: If Master mode is enabled, a SMBus Quick Command is sent.

- **SWRST: Software Reset**

0: No effect.

1: Equivalent to a system reset.

- **HSEN: TWIHS High-Speed Mode Enabled**

0: No effect.

1: High-speed mode enabled.

- **HSDIS: TWIHS High-Speed Mode Disabled**

0: No effect.

1: High-speed mode disabled.

- **SMBEN: SMBus Mode Enabled**

0: No effect.

1: If SMBDIS = 0, SMBus mode enabled.

- **SMBDIS: SMBus Mode Disabled**

0: No effect.

1: SMBus mode disabled.

- **PECEN: Packet Error Checking Enable**

0: No effect.

1: SMBus PEC (CRC) generation and check enabled.

- **PECDIS: Packet Error Checking Disable**

0: No effect.

1: SMBus PEC (CRC) generation and check disabled.

- **PECRQ: PEC Request**

0: No effect.

1: A PEC check or transmission is requested.

- **CLEAR: Bus CLEAR Command**

0: No effect.

1: If Master mode is enabled, sends a bus clear command.

- **ACMEN: Alternative Command Mode Enable**

0: No effect.

1: Alternative Command mode enabled.

- **ACMDIS: Alternative Command Mode Disable**

0: No effect.

1: Alternative Command mode disabled.

- **THRCLR: Transmit Holding Register Clear**

0: No effect.

1: Clears the Transmit Holding Register and set TXRDY, TXCOMP flags.

- **LOCKCLR: Lock Clear**

0: No effect.

1: Clears the TWIHS FSM lock.

- **FIFOEN: FIFO Enable**

0: No effect.

1: Enables the Transmit and Receive FIFOs

- **FIFODIS: FIFO Disable**

0: No effect.

1: Disables the Transmit and Receive FIFOs

### 43.7.2 TWIHS Master Mode Register

**Name:** TWIHS_MMR

**Address:** 0xF8028004 (0), 0xFC028004 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | DADR | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | MREAD | – | – | IADRSZ | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

- **IADRSZ: Internal Device Address Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | No internal device address |
| 1 | 1_BYTE | One-byte internal device address |
| 2 | 2_BYTE | Two-byte internal device address |
| 3 | 3_BYTE | Three-byte internal device address |

- **MREAD: Master Read Direction**

0: Master write direction.

1: Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in Read or Write mode. These bits are only used in Master mode.

### 43.7.3 TWIHS Slave Mode Register

**Name:** TWIHS_SMR

**Address:** 0xF8028008 (0), 0xFC028008 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DATAMEN | SADR3EN | SADR2EN | SADR1EN | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | SADR | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | MASK | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | SCLWSDIS | – | – | SMHH | SMDA | – | NACKEN |

This register can only be written if the WPEN bit is cleared in the TWIHS Write Protection Mode Register.

- **NACKEN: Slave Receiver Data Phase NACK enable**

0: Normal value to be returned in the ACK cycle of the data phase in Slave Receiver mode.

1: NACK value to be returned in the ACK cycle of the data phase in Slave Receiver mode.

- **SMDA: SMBus Default Address**

0: Acknowledge of the SMBus default address disabled.

1: Acknowledge of the SMBus default address enabled.

- **SMHH: SMBus Host Header**

0: Acknowledge of the SMBus host header disabled.

1: Acknowledge of the SMBus host header enabled.

- **SCLWSDIS: Clock Wait State Disable**

0: No effect.

1: Clock stretching disabled in Slave mode, OVRE and UNRE indicate an overrun/underrun.

- **MASK: Slave Address Mask**

A mask can be applied on the slave device address in Slave mode in order to allow multiple address answer. For each bit of the MASK field set to 1, the corresponding SADR bit is masked.

If the MASK field value is 0, no mask is applied to the SADR field.

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in Read or Write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

- **SADR1EN: Slave Address 1 Enable**

0: Slave address 1 matching is disabled.

1: Slave address 1 matching is enabled.

- **SADR2EN: Slave Address 2 Enable**

0: Slave address 2 matching is disabled.

1: Slave address 2 matching is enabled.

- **SADR3EN: Slave Address 3 Enable**

0: Slave address 3 matching is disabled.

1: Slave address 3 matching is enabled.

- **DATAMEN: Data Matching Enable**

0: Data matching on first received data is disabled.

1: Data matching on first received data is enabled.

### 43.7.4 TWIHS Internal Address Register

**Name:** TWIHS_IADR

**Address:** 0xF802800C (0), 0xFC02800C (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IADR | | | | | | | |

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

### 43.7.5 TWIHS Clock Waveform Generator Register

**Name:** TWIHS_CWGR

**Address:** 0xF8028010 (0), 0xFC028010 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | HOLD | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | CKSRC | – | CKDIV | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CHDIV | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CLDIV | | | | | | | |

This register can only be written if the WPEN bit is cleared in the TWIHS Write Protection Mode Register.

TWIHS_CWGR is used in Master mode only.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

If CKSRC = 0

$t_{low} = ((CLDIV \times 2^{CKDIV}) + 3) \times t_{peripheral\ clock}$

If CKSRC = 1

$t_{low} = (CLDIV \times 2^{CKDIV}) \times t_{external\ clock}$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

If CKSRC = 0

$t_{high} = ((CHDIV \times 2^{CKDIV}) + 3) \times t_{peripheral\ clock}$

If CKSRC = 1

$t_{high} = (CHDIV \times 2^{CKDIV}) \times t_{external\ clock}$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

- **HOLD: TWD Hold Time Versus TWCK Falling**

If High-speed mode is selected TWD is internally modified on the TWCK falling edge to meet the I2C specified maximum hold time, else if High-speed mode is not configured TWD is kept unchanged after TWCK falling edge for a period of $(HOLD + 3) \times t_{peripheral\ clock}$.

- **CKSRC: Transfer Rate Clock Source**

| Value | Name | Description |
|---|---|---|
| 0 | PERIPH_CK | Peripheral clock is used to generate the TWIHS baud rate. |
| 1 | GCLK | GCLK is used to generate the TWIHS baud rate. |

### 43.7.6 TWIHS Status Register

**Name:** TWIHS_SR

**Address:** 0xF8028020 (0), 0xFC028020 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | SDA | SCL |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| LOCK | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | EOSACC | SCLWS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | SVREAD | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed (cleared by writing TWIHS_THR)**

TXCOMP used in Master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

*TXCOMP behavior in Master mode* can be seen in Figure 43-6 and in Figure 43-8.

TXCOMP used in Slave mode:

0: As soon as a START is detected.

1: After a STOP or a REPEATED START + an address different from SADR is detected.

*TXCOMP behavior in Slave mode* can be seen in Figure 43-39, Figure 43-40, Figure 43-41 and Figure 43-42.

- **RXRDY: Receive Holding Register Ready (cleared by reading TWIHS_RHR)**

0: No character has been received since the last TWIHS_RHR read operation.

1: A byte has been received in the TWIHS_RHR since the last read.

*RXRDY behavior in Master mode* can be seen in Figure 43-7, Figure 43-8 and Figure 43-9.

*RXRDY behavior in Slave mode* can be seen in Figure 43-37, Figure 43-40, Figure 43-41 and Figure 43-42.

- **TXRDY: Transmit Holding Register Ready (cleared by writing TWIHS_THR)**

TXRDY used in Master mode:

0: The transmit holding register has not been transferred into the internal shifter. Set to 0 when writing into TWIHS_THR.

1: As soon as a data byte is transferred from TWIHS_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enables TWIHS).

*TXRDY behavior in Master mode* can be seen in Figure 43-4, Figure 43-5 and Figure 43-6.

TXRDY used in Slave mode:

0: As soon as data is written in the TWIHS_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1: Indicates that the TWIHS_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission is stopped. Thus when TRDY = NACK = 1, the user must not fill TWIHS_THR to avoid losing it.

*TXRDY behavior in Slave mode* can be seen in Figure 43-36, Figure 43-39, Figure 43-41 and Figure 43-42.

- **SVREAD: Slave Read**

This bit is used in Slave mode only. When SVACC is low (no slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a master.

1: Indicates that a read access is performed by a master.

*SVREAD behavior* can be seen in Figure 43-36, Figure 43-37, Figure 43-41 and Figure 43-42.

- **SVACC: Slave Access**

This bit is used in Slave mode only.

0: TWIHS is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

*SVACC behavior* can be seen in Figure 43-36, Figure 43-37, Figure 43-41 and Figure 43-42.

- **GACC: General Call Access (cleared on read)**

This bit is used in Slave mode only.

0: No general call has been detected.

1: A general call has been detected. After the detection of general call, if need be, the user may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

*GACC behavior* can be seen in Figure 43-38.

- **OVRE: Overrun Error (cleared on read)**

This bit is used only if clock stretching is disabled.

0: TWIHS_RHR has not been loaded while RXRDY was set.

1: TWIHS_RHR has been loaded while RXRDY was set. Reset by read in TWIHS_SR when TXCOMP is set.

- **UNRE: Underrun Error (cleared on read)**

This bit is used only if clock stretching is disabled.

0: TWIHS_THR has been filled on time.

1: TWIHS_THR has not been filled on time.

- **NACK: Not Acknowledged (cleared on read)**

NACK used in Master mode:

0: Each data byte has been correctly received by the far-end side TWIHS slave component.

1: A data or address byte has not been acknowledged by the slave component. Set at the same time as TXCOMP.

NACK used in Slave Read mode:

0: Each data byte has been correctly received by the master.

1: In Read mode, a data byte has not been acknowledged by the master. When NACK is set, the user must not fill TWIHS_THR even if TXRDY is set, because it means that the master stops the data transfer or reinitiate it.

Note: in Slave Write mode, all data are acknowledged by the TWIHS.

- **ARBLST: Arbitration Lost (cleared on read)**

This bit is used in Master mode only.

0: Arbitration won.

1: Arbitration lost. Another master of the TWIHS bus has won the multimaster arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State**

This bit is used in Slave mode only.

0: The clock is not stretched.

1: The clock is stretched. TWIHS_THR / TWIHS_RHR buffer is not filled / emptied before the transmission / reception of a new character.

*SCLWS behavior* can be seen in Figure 43-39 and Figure 43-40.

- **EOSACC: End Of Slave Access (cleared on read)**

This bit is used in Slave mode only.

0: A slave access is being performing.

1: The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

*EOSACC behavior* can be seen in Figure 43-41 and Figure 43-42.

- **MCACK: Master Code Acknowledge (cleared on read)**

MACK used in Slave mode:

0: No Master Code has been received since the last read of TWIHS_SR.

1: A Master Code has been received since the last read of TWIHS_SR.

- **TOUT: Timeout Error (cleared on read)**

0: No SMBus timeout occurred since the last read of TWIHS_SR.

1: An SMBus timeout occurred since the last read of TWIHS_SR.

- **PECERR: PEC Error (cleared on read)**

0: No SMBus PEC error occurred since the last read of TWIHS_SR.

1: An SMBus PEC error occurred since the last read of TWIHS_SR.

- **SMBDAM: SMBus Default Address Match (cleared on read)**

0: No SMBus Default Address received since the last read of TWIHS_SR.

1: An SMBus Default Address was received since the last read of TWIHS_SR.

- **SMBHHM: SMBus Host Header Address Match (cleared on read)**

0: No SMBus Host Header Address received since the last read of TWIHS_SR.

1: An SMBus Host Header Address was received since the last read of TWIHS_SR.

- **LOCK: TWIHS Lock due to Frame Errors (cleared by writing a one to bit LOCKCLR in TWIHS_CR)**

0: The TWIHS is not locked or LOCKCLR command issued in TWIHS_CR.

1: The TWIHS is locked due to frame errors (see Section 43.6.3.13 “Handling Errors in Alternative Command”).

- **SCL: SCL Line Value**

0: SCL line sampled value is ‘0’.

1: SCL line sampled value is ‘1.’

- **SDA: SDA Line Value**

0: SDA line sampled value is ‘0’.

1: SDA line sampled value is ‘1’.

### 43.7.7 TWIHS SMBus Timing Register

**Name:** TWIHS_SMBTR

**Address:** 0xF8028038 (0), 0xFC028038 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| THMAX | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| TLOWM | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TLOWS | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PRESC | | | |

This register can only be written if the WPEN bit is cleared in the TWIHS Write Protection Mode Register.

- **PRESC: SMBus Clock Prescaler**

Used to specify how to prescale the TLOWS, TLOWM and THMAX counters in SMBTR. Counters are prescaled according to the following formula:

$$f_{Prescaled} = \frac{f_{\text{peripheral clock}}}{2^{(PRESC+1)}}$$

- **TLOWS: Slave Clock Stretch Maximum Cycles**

0: TLOW:SEXT timeout check disabled.

1–255: Clock cycles in slave maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:SEXT.

- **TLOWM: Master Clock Stretch Maximum Cycles**

0: TLOW:MEXT timeout check disabled.

1–255: Clock cycles in master maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:MEXT.

- **THMAX: Clock High Maximum Cycles**

Clock cycles in clock high maximum count. Prescaled by PRESC. Used for bus free detection. Used to time THIGH:MAX.

### 43.7.8 TWIHS Alternative Command Register

**Name:** TWIHS_ACR

**Address:** 0xF8028040 (0), 0xFC028040 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | NPEC | NDIR |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| NDATAL | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | PEC | DIR |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DATAL | | | | | | | |

- **DATAL: Data Length**

0: No data to send (see Section 43.6.3.12 “Alternative Command”).

1–255: Number of bytes to send during the transfer.

- **DIR: Transfer Direction**

0: Write direction.

1: Read direction.

- **PEC: PEC Request (SMBus Mode only)**

0: The transfer does not use a PEC byte.

1: The transfer uses a PEC byte.

- **NDATAL: Next Data Length**

0: No data to send (see Section 43.6.3.12 “Alternative Command”.

1–255: Number of bytes to send for the next transfer.

- **NDIR: Next Transfer Direction**

0: Write direction.

1: Read direction.

- **NPEC: Next PEC Request (SMBus Mode only)**

0: The next transfer does not use a PEC byte.

1: The next transfer uses a PEC byte.

### 43.7.9 TWIHS Filter Register

**Name:** TWIHS_FILTR

**Address:** 0xF8028044 (0), 0xFC028044 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | THRES | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | PADFCFG | PADFEN | FILT |

- **FILT: RX Digital Filter**

0: No filtering applied on TWIHS inputs.

1: TWIHS input filtering is active (only in Standard and Fast modes)

Note: TWIHS digital input filtering follows a majority decision based on three samples from SDA/SCL lines at peripheral clock frequency.

- **PADFEN: PAD Filter Enable**

0: PAD analog filter is disabled.

1: PAD analog filter is enabled. (The analog filter must be enabled if High-speed mode is enabled.)

- **PADFCFG: PAD Filter Config**

See the electrical characteristics section for filter configuration details.

- **THRES: Digital Filter Threshold**

0: No filtering applied on TWIHS inputs.

1–7: Maximum pulse width of spikes to be suppressed by the input filter, defined in peripheral clock cycles.

### 43.7.10 TWIHS Interrupt Enable Register

**Name:** TWIHS_IER

**Address:** 0xF8028024 (0), 0xFC028024 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **TXCOMP: Transmission Completed Interrupt Enable**
- **RXRDY: Receive Holding Register Ready Interrupt Enable**
- **TXRDY: Transmit Holding Register Ready Interrupt Enable**
- **SVACC: Slave Access Interrupt Enable**
- **GACC: General Call Access Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **UNRE: Underrun Error Interrupt Enable**
- **NACK: Not Acknowledge Interrupt Enable**
- **ARBLST: Arbitration Lost Interrupt Enable**
- **SCL_WS: Clock Wait State Interrupt Enable**
- **EOSACC: End Of Slave Access Interrupt Enable**
- **MCACK: Master Code Acknowledge Interrupt Enable**
- **TOUT: Timeout Error Interrupt Enable**
- **PECERR: PEC Error Interrupt Enable**
- **SMBDAM: SMBus Default Address Match Interrupt Enable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Enable**

### 43.7.11 TWIHS Interrupt Disable Register

**Name:** TWIHS_IDR

**Address:** 0xF8028028 (0), 0xFC028028 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **TXCOMP: Transmission Completed Interrupt Disable**
- **RXRDY: Receive Holding Register Ready Interrupt Disable**
- **TXRDY: Transmit Holding Register Ready Interrupt Disable**
- **SVACC: Slave Access Interrupt Disable**
- **GACC: General Call Access Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **UNRE: Underrun Error Interrupt Disable**
- **NACK: Not Acknowledge Interrupt Disable**
- **ARBLST: Arbitration Lost Interrupt Disable**
- **SCL_WS: Clock Wait State Interrupt Disable**
- **EOSACC: End Of Slave Access Interrupt Disable**
- **MCACK: Master Code Acknowledge Interrupt Disable**
- **TOUT: Timeout Error Interrupt Disable**
- **PECERR: PEC Error Interrupt Disable**
- **SMBDAM: SMBus Default Address Match Interrupt Disable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Disable**

### 43.7.12 TWIHS Interrupt Mask Register

**Name:** TWIHS_IMR

**Address:** 0xF802802C (0), 0xFC02802C (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | EOSACC | SCL_WS | ARBLST | NACK |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **TXCOMP: Transmission Completed Interrupt Mask**
- **RXRDY: Receive Holding Register Ready Interrupt Mask**
- **TXRDY: Transmit Holding Register Ready Interrupt Mask**
- **SVACC: Slave Access Interrupt Mask**
- **GACC: General Call Access Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **UNRE: Underrun Error Interrupt Mask**
- **NACK: Not Acknowledge Interrupt Mask**
- **ARBLST: Arbitration Lost Interrupt Mask**
- **SCL_WS: Clock Wait State Interrupt Mask**
- **EOSACC: End Of Slave Access Interrupt Mask**
- **MCACK: Master Code Acknowledge Interrupt Mask**
- **TOUT: Timeout Error Interrupt Mask**
- **PECERR: PEC Error Interrupt Mask**
- **SMBDAM: SMBus Default Address Match Interrupt Mask**
- **SMBHHM: SMBus Host Header Address Match Interrupt Mask**

### 43.7.13 TWIHS Receive Holding Register

**Name:** TWIHS_RHR

**Address:** 0xF8028030 (0), 0xFC028030 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA | | | | | | | |

- **RXDATA: Master or Slave Receive Holding Data**

### 43.7.14 TWIHS Receive Holding Register (FIFO Enabled)

Name: TWIHS_RHR (FIFO_ENABLED)

**Address:** 0xF8028030 (0), 0xFC028030 (1)

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXDATA3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RXDATA2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXDATA1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in TWIHS_CR), refer to Section "Master Multiple Data Mode" for details.

- **RXDATA0: Master or Slave Receive Holding Data 0**
- **RXDATA1: Master or Slave Receive Holding Data 1**
- **RXDATA2: Master or Slave Receive Holding Data 2**
- **RXDATA3: Master or Slave Receive Holding Data 3**

### 43.7.15 TWIHS SleepWalking Matching Register

**Name:** TWIHS_SWMR

**Address:** 0xF802804C (0), 0xFC02804C (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DATAM | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | SADR3 | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | SADR2 | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | SADR1 | | | | | | |

This register can only be written if the WPEN bit is cleared in the TWIHS Write Protection Mode Register.

- **SADR1: Slave Address 1**

Slave address 1. The TWIHS module matches on this additional address if SADR1EN bit is enabled.

- **SADR2: Slave Address 2**

Slave address 2. The TWIHS module matches on this additional address if SADR2EN bit is enabled.

- **SADR3: Slave Address 3**

Slave address 3. The TWIHS module matches on this additional address if SADR3EN bit is enabled.

- **DATAM: Data Match**

The TWIHS module extends the SleepWalking matching process to the first received data, comparing it with DATAM if DATAMEN bit is enabled.

### 43.7.16 TWIHS Transmit Holding Register

**Name:** TWIHS_THR

**Address:** 0xF8028034 (0), 0xFC028034 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA | | | | | | | |

- **TXDATA: Master or Slave Transmit Holding Data**

### 43.7.17 TWIHS Transmit Holding Register (FIFO Enabled)

Name: TWIHS_THR (FIFO_ENABLED)

**Address:** 0xF8028034 (0), 0xFC028034 (1)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TXDATA3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TXDATA2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXDATA1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in TWIHS_CR), refer to Section "Master Multiple Data Mode" for details.

- **TXDATA0: Master or Slave Transmit Holding Data 02**
- **TXDATA1: Master or Slave Transmit Holding Data 1**
- **TXDATA2: Master or Slave Transmit Holding Data 2**
- **TXDATA3: Master or Slave Transmit Holding Data 3**

### 43.7.18 TWIHS FIFO Mode Register

**Name:** TWIHS_FMR

**Address:** 0xF8028050 (0), 0xFC028050 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | RXFTHRES | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | TXFTHRES | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | RXRDYM | | – | – | TXRDYM | |

- **TXRDYM: Transmitter Ready Mode**

If FIFOs are enabled, the TXRDY flag (in TWIHS_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | TXRDY will be at level ‘1’ when at least one data can be written in the Transmit FIFO |
| 0x1 | TWO_DATA | TXRDY will be at level ‘1’ when at least two data can be written in the Transmit FIFO |
| 0x2 | FOUR_DATA | TXRDY will be at level ‘1’ when at least four data can be written in the Transmit FIFO |

- **RXRDYM: Receiver Ready Mode**

If FIFOs are enabled, the RXRDY flag (in TWIHS_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | RXRDY will be at level ‘1’ when at least one unread data is in the Receive FIFO |
| 0x1 | TWO_DATA | RXRDY will be at level ‘1’ when at least two unread data are in the Receive FIFO |
| 0x2 | FOUR_DATA | RXRDY will be at level ‘1’ when at least four unread data are in the Receive FIFO |

- **TXFTHRES: Transmit FIFO Threshold**

0–16: Defines the Transmit FIFO threshold value (number of data). TXFTH flag in TWIHS_FSR will be set when Transmit FIFO goes from “above” threshold state to “equal or below” threshold state.

- **RXFTHRES: Receive FIFO Threshold**

0–16: Defines the Receive FIFO threshold value (number of data). RXFTH flag in TWIHS_FSR will be set when Receive FIFO goes from “below” threshold state to “equal or above” threshold state.

### 43.7.19 TWIHS FIFO Level Register

**Name:** TWIHS_FLR

**Address:** 0xF8028054 (0), 0xFC028054 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFL | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TXFL | | | | | |

- **TXFL: Transmit FIFO Level**

0: There is no data in the Transmit FIFO

1–16: Indicates the number of DATA in the Transmit FIFO

- **RXFL: Receive FIFO Level**

0: There is no unread data in the Receive FIFO

1–16: Indicates the number of unread DATA in the Receive FIFO

### 43.7.20 TWIHS FIFO Status Register

**Name:** TWIHS_FSR

**Address:** 0xF8028060 (0), 0xFC028060 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: Transmit FIFO Empty Flag (cleared on read)**

0: Transmit FIFO is not empty.

1: Transmit FIFO has been emptied since the last read of TWIHS_FSR.

- **TXFFF: Transmit FIFO Full Flag (cleared on read)**

0: Transmit FIFO is not full.

1: Transmit FIFO has been filled since the last read of TWIHS_FSR.

- **TXFTHF: Transmit FIFO Threshold Flag (cleared on read)**

0: Number of DATA in Transmit FIFO is above TXFTHRES threshold.

1: Number of DATA in Transmit FIFO has reached TXFTHRES threshold since the last read of TWIHS_FSR.

- **RXFEF: Receive FIFO Empty Flag**

0: Receive FIFO is not empty.

1: Receive FIFO has been emptied since the last read of TWIHS_FSR.

- **RXFFF: Receive FIFO Full Flag**

0: Receive FIFO is not empty.

1: Receive FIFO has been filled since the last read of TWIHS_FSR.

- **RXFTHF: Receive FIFO Threshold Flag**

0: Number of unread DATA in Receive FIFO is below RXFTHRES threshold.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES threshold since the last read of TWIHS_FSR.

- **TXFPTEF: Transmit FIFO Pointer Error Flag**

0: No Transmit FIFO pointer occurred

1: Transmit FIFO pointer error occurred. Transceiver must be reset

See Section ”FIFO Pointer Error” for details.

- **RXFPTEF: Receive FIFO Pointer Error Flag**

0: No Receive FIFO pointer occurred

1: Receive FIFO pointer error occurred. Receiver must be reset

See Section ”FIFO Pointer Error” for details.

### 43.7.21 TWIHS FIFO Interrupt Enable Register

**Name:** TWIHS_FIER

**Address:** 0xF8028064 (0), 0xFC028064 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **TXFEF: TXFEF Interrupt Enable**
- **TXFFF: TXFFF Interrupt Enable**
- **TXFTHF: TXFTHF Interrupt Enable**
- **RXFEF: RXFEF Interrupt Enable**
- **RXFFF: RXFFF Interrupt Enable**
- **RXFTHF: RXFTHF Interrupt Enable**
- **TXFPTEF: TXFPTEF Interrupt Enable**
- **RXFPTEF: RXFPTEF Interrupt Enable**

### 43.7.22 TWIHS FIFO Interrupt Disable Register

**Name:** TWIHS_FIDR

**Address:** 0xF8028068 (0), 0xFC028068 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **TXFEF: TXFEF Interrupt Disable**
- **TXFFF: TXFFF Interrupt Disable**
- **TXFTHF: TXFTHF Interrupt Disable**
- **RXFEF: RXFEF Interrupt Disable**
- **RXFFF: RXFFF Interrupt Disable**
- **RXFTHF: RXFTHF Interrupt Disable**
- **TXFPTEF: TXFPTEF Interrupt Disable**
- **RXFPTEF: RXFPTEF Interrupt Disable**

### 43.7.23 TWIHS FIFO Interrupt Mask Register

**Name:** TWIHS_FIMR

**Address:** 0xF802806C (0), 0xFC02806C (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **TXFEF: TXFEF Interrupt Mask**
- **TXFFF: TXFFF Interrupt Mask**
- **TXFTHF: TXFTHF Interrupt Mask**
- **RXFEF: RXFEF Interrupt Mask**
- **RXFFF: RXFFF Interrupt Mask**
- **RXFTHF: RXFTHF Interrupt Mask**
- **TXFPTEF: TXFPTEF Interrupt Mask**
- **RXFPTEF: RXFPTEF Interrupt Mask**

### 43.7.24 TWIHS Write Protection Mode Register

**Name:** TWIHS_WPMR

**Address:** 0xF80280E4 (0), 0xFC0280E4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

See Section 43.6.7 “Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x545749 | PASSWD | Writing any other value in this field aborts the write operation of the WPEN bit.<br>Always reads as 0 |

### 43.7.25 TWIHS Write Protection Status Register

**Name:** TWIHS_WPSR

**Address:** 0xF80280E8 (0), 0xFC0280E8 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the TWIHS_WPSR.

1: A write protection violation has occurred since the last read of the TWIHS_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

# 44. Flexible Serial Communication Controller (FLEXCOM)

## 44.1 Description

The Flexible Serial Communication Controller (FLEXCOM) offers several serial communication protocols that are managed by the three submodules USART, SPI, and TWI.

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver timeout enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

The USART features three test modes: Remote Loopback, Local Loopback and Automatic Echo.

The USART supports specific operating modes providing interfaces on RS485, LIN, and SPI, with ISO7816 T = 0 or T = 1 smart card slots, and infrared transceivers. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

The USART supports the connection to the DMA Controller, which enables data transfers to the transmitter and from the receiver. The DMAC provides chained buffer management without any intervention of the processor.

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a Shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves” which have data shifted into and out by the master. Different CPUs can take turn being masters (multiple master protocol, contrary to single master protocol where one CPU is always the master while all of the others are always slaves). One master can simultaneously shift data into multiple slaves. However, only one slave can drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI)—This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO)—This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK)—This control line is driven by the master and regulates the flow of the data bits. The master can transmit data at a variety of baud rates; there is one SPCK pulse for each bit that is transmitted.
- Slave Select (NSS)—This control line allows slaves to be turned on and off by hardware.

The Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 kbits per second in Fast mode and up to 3.4 Mbits per second in High-speed Slave mode only, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and $I^2C$-compatible devices, such as a Real-Time Clock (RTC), Dot Matrix/Graphic LCD Controller and temperature sensor. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported.

Arbitration of the bus is performed internally and puts the TWI in Slave mode automatically if the bus arbitration is lost.

A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

Table 44-1 lists the compatibility level of the Atmel Two-wire Interface in Master mode and a full $I^2C$ compatible device.

**Table 44-1. Atmel TWI Compatibility with $I^2C$ Standard**

| $I^2C$ Standard | Atmel TWI |
|---|---|
| Standard Mode Speed (100 kHz) | Supported |
| Fast Mode Speed (400 kHz) | Supported |
| High-speed Mode (Slave only, 3.4 MHz) | Supported |
| 7- or 10-bit(1) Slave Addressing | Supported |
| Repeated Start (Sr) Condition | Supported |
| ACK and NACK Management | Supported |
| Input Filtering | Supported |
| Slope Control | Not Supported |
| Clock Stretching | Supported |
| Multi Master Capability | Supported |

Notes: 1. 10-bit support in Master mode only

## 44.2 Embedded Characteristics

### 44.2.1 USART/UART Characteristics

- 32-byte Transmit and Receive FIFOs
- Programmable Baud Rate Generator
- Baud Rate can be Independent of the Processor/Peripheral Clock
- Comparison Function on Received Character
- 5-bit to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - Digital Filter on Receive Line
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by 16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Receiver Timeout and Transmitter Timeguard
  - Optional Multidrop Mode with Address Generation and Detection
- RS485 with Driver Control Signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 kbit/s
- SPI Mode
  - MASTER or Slave

  - Serial Clock Programmable Phase and Polarity
  - SPI Serial Clock (SCK) Frequency up to $f_{peripheral\ clock}/6$
- LIN Mode
  - Compliant with LIN 1.3 and LIN 2.0 specifications
  - Master or Slave
  - Processing of Frames with Up to 256 Data Bytes
  - Response Data Length can be Configurable or Defined Automatically by the Identifier
  - Self-synchronization in Slave Node Configuration
  - Automatic Processing and Verification of the “Synch Break” and the “Synch Field”
  - “Synch Break” Detection Even When Partially Superimposed with a Data Byte
  - Automatic Identifier Parity Calculation/Sending and Verification
  - Parity Sending and Verification Can be Disabled
  - Automatic Checksum Calculation/sending and Verification
  - Checksum Sending and Verification Can be Disabled
  - Support Both “Classic” and “Enhanced” Checksum Types
  - Full LIN Error Checking and Reporting
  - Frame Slot Mode: Master Allocates Slots to the Scheduled Frames Automatically
  - Generation of the Wakeup Signal
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of:
  - Two DMA Controller (DMAC) Channels
  - Offers Buffer Transfer without Processor Intervention
- Register Write Protection

### 44.2.2 SPI Characteristics

- 32-byte Transmit and Receive FIFOs
- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit programmable data length per chip select
  - Programmable phase and polarity per chip select
  - Programmable transfer delay between consecutive transfers and delay before SPI clock per chip select
  - Programmable delay between chip selects
- Selectable mode fault detection
- Master Mode can drive SPCK up to Peripheral Clock
- Master Mode Bit Rate can be Independent of the Processor/Peripheral Clock
- Slave mode operates on SPCK, asynchronously with core and bus clock
- Two chip selects with external decoder support allow communication with up to 3 peripherals
- Communication with Serial External Devices Supported
  - Serial memories, such as DataFlash and 3-wire EEPROMs
  - Serial peripherals, such as ADCs, DACs, LCD controllers, CAN controllers and sensors
  - External coprocessors
- Connection to DMA Channel Capabilities, Optimizing Data Transfers
  - One channel for the receiver

- One channel for the transmitter
- Register Write Protection

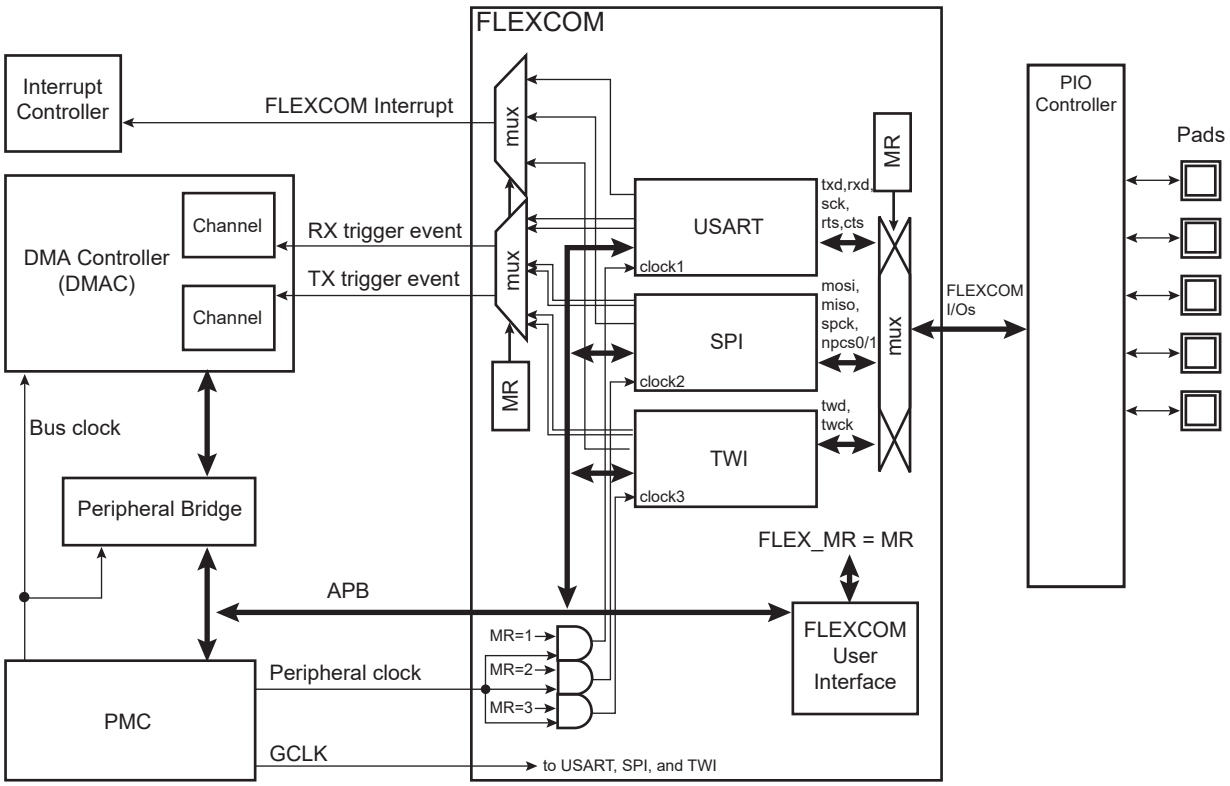
### 44.2.3 TWI/SMBus Characteristics

- 16-byte Transmit and Receive FIFOs
- Bit Rate: Up to 400 kbit/s in Fast Mode and 3.4 Mbit/s in High-Speed Mode (Slave Only)
- Bit Rate can be Independent of the Processor/Peripheral Clock
- SMBus support
- Compatible with Atmel Two-wire Interface Serial Memory and $I^2C$ Compatible Devices(1)
- Master and Multi-Master Operation (Standard and Fast Mode Only)
- Slave Mode Operation (Standard, Fast and High-Speed Mode)
- One, Two or Three Bytes for Slave Address
- Sequential Read/Write Operations
- General Call Supported in Slave Mode
- Connection to DMA Controller Channels Optimizes Data Transfers
  - One Channel for the Receiver
  - One Channel for the Transmitter
- Register Write Protection

Note: 1. See Table 44-1 for details on compatibility with $I^2C$ Standard.

## 44.3 Block Diagram

**Figure 44-1. FLEXCOM Block Diagram**



## 44.4 I/O Lines Description

**Table 44-2. I/O Lines Description**

| Name | Description | | | Type |
|---|---|---|---|---|
| | USART/UART | SPI | TWI | |
| FLEXCOM_IO0 | TXD | MOSI | TWD | I/O |
| FLEXCOM_IO1 | RXD | MISO | TWCK | I/O |
| FLEXCOM_IO2 | SCK | SPCK | – | I/O |
| FLEXCOM_IO3 | CTS | NPCS0/NSS | – | I/O |
| FLEXCOM_IO4 | RTS | NPCS1 | – | O |

## 44.5 Product Dependencies

### 44.5.1 I/O Lines

The pins used for interfacing the FLEXCOM are multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired FLEXCOM pins to their peripheral function. If I/O lines of the FLEXCOM are not used by the application, they can be used for other purposes by the PIO Controller.

**Table 44-3. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| FLEXCOM0 | FLEXCOM0_IO0 | PB28 | C |
| FLEXCOM0 | FLEXCOM0_IO1 | PB29 | C |
| FLEXCOM0 | FLEXCOM0_IO2 | PB30 | C |
| FLEXCOM0 | FLEXCOM0_IO3 | PB31 | C |
| FLEXCOM0 | FLEXCOM0_IO4 | PC0 | C |
| FLEXCOM1 | FLEXCOM1_IO0 | PA24 | A |
| FLEXCOM1 | FLEXCOM1_IO1 | PA23 | A |
| FLEXCOM1 | FLEXCOM1_IO2 | PA22 | A |
| FLEXCOM1 | FLEXCOM1_IO3 | PA25 | A |
| FLEXCOM1 | FLEXCOM1_IO4 | PA26 | A |
| FLEXCOM2 | FLEXCOM2_IO0 | PA6 | E |
| FLEXCOM2 | FLEXCOM2_IO0 | PD26 | C |
| FLEXCOM2 | FLEXCOM2_IO1 | PA7 | E |
| FLEXCOM2 | FLEXCOM2_IO1 | PD27 | C |
| FLEXCOM2 | FLEXCOM2_IO2 | PA8 | E |
| FLEXCOM2 | FLEXCOM2_IO2 | PD28 | C |
| FLEXCOM2 | FLEXCOM2_IO3 | PA9 | E |
| FLEXCOM2 | FLEXCOM2_IO3 | PD29 | C |
| FLEXCOM2 | FLEXCOM2_IO4 | PA10 | E |
| FLEXCOM2 | FLEXCOM2_IO4 | PD30 | C |
| FLEXCOM3 | FLEXCOM3_IO0 | PA15 | E |

Table 44-3. I/O Lines (Continued)

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| FLEXCOM3 | FLEXCOM3_IO0 | PB23 | E |
| FLEXCOM3 | FLEXCOM3_IO0 | PC20 | E |
| FLEXCOM3 | FLEXCOM3_IO1 | PA13 | E |
| FLEXCOM3 | FLEXCOM3_IO1 | PB22 | E |
| FLEXCOM3 | FLEXCOM3_IO1 | PC19 | E |
| FLEXCOM3 | FLEXCOM3_IO2 | PA14 | E |
| FLEXCOM3 | FLEXCOM3_IO2 | PB21 | E |
| FLEXCOM3 | FLEXCOM3_IO2 | PC18 | E |
| FLEXCOM3 | FLEXCOM3_IO3 | PA16 | E |
| FLEXCOM3 | FLEXCOM3_IO3 | PB24 | E |
| FLEXCOM3 | FLEXCOM3_IO3 | PC21 | E |
| FLEXCOM3 | FLEXCOM3_IO4 | PA17 | E |
| FLEXCOM3 | FLEXCOM3_IO4 | PB25 | E |
| FLEXCOM3 | FLEXCOM3_IO4 | PC22 | E |
| FLEXCOM4 | FLEXCOM4_IO0 | PC28 | B |
| FLEXCOM4 | FLEXCOM4_IO0 | PD12 | B |
| FLEXCOM4 | FLEXCOM4_IO0 | PD21 | C |
| FLEXCOM4 | FLEXCOM4_IO1 | PC29 | B |
| FLEXCOM4 | FLEXCOM4_IO1 | PD13 | B |
| FLEXCOM4 | FLEXCOM4_IO1 | PD22 | C |
| FLEXCOM4 | FLEXCOM4_IO2 | PC30 | B |
| FLEXCOM4 | FLEXCOM4_IO2 | PD14 | B |
| FLEXCOM4 | FLEXCOM4_IO2 | PD23 | C |
| FLEXCOM4 | FLEXCOM4_IO3 | PC31 | B |
| FLEXCOM4 | FLEXCOM4_IO3 | PD15 | B |
| FLEXCOM4 | FLEXCOM4_IO3 | PD24 | C |
| FLEXCOM4 | FLEXCOM4_IO4 | PD0 | B |
| FLEXCOM4 | FLEXCOM4_IO4 | PD16 | B |
| FLEXCOM4 | FLEXCOM4_IO4 | PD25 | C |

### 44.5.2 Power Management

The peripheral clock is not continuously provided to the FLEXCOM. The programmer must first enable the FLEXCOM Clock in the Power Management Controller (PMC) before using the USART or SPI or TWI.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the FLEXCOM in the Sleepwalking Enable Register (PMC_SLPWK_ER). The peripheral clock can be automatically provided to the FLEXCOM, depending on the instructions (requests) provided by the FLEXCOM to the PMC.

### 44.5.3 Interrupt Sources

The FLEXCOM interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the FLEXCOM interrupt requires the Interrupt Controller to be programmed first.

**Table 44-4. Peripheral IDs**

| Instance | ID |
|---|---|
| FLEXCOM0 | 19 |
| FLEXCOM1 | 20 |
| FLEXCOM2 | 21 |
| FLEXCOM3 | 22 |
| FLEXCOM4 | 23 |

## 44.6 Register Accesses

Register accesses supports 8-bit, 16-bit and 32-bit accesses which means that only an 8-bit part of a 32-bit register can be written in one access for instance. For this the access must be done with the right size at the right address.

8-bit, 16-bit and 32-bit accesses are supported for register accesses however a field in a register cannot be partially written (e.g., if a field is bigger than 8 bits, the whole field must be written).

This feature helps avoiding a read-modify-write process if only a small part of the register is to be modified.

## 44.7 USART Functional Description

### 44.7.1 Baud Rate Generator

The baud rate generator provides the bit period clock named “baud rate clock” to both the receiver and the transmitter.

Configuring the USCLKS field in FLEX_US_MR selects the baud rate generator clock from one of the following sources:

- the peripheral clock
- a division of the peripheral clock, the divider being product dependent, but generally set to 8
- a fully programmable generic clock (GCLK) provided by PMC and independent of processor/peripheral clock
- the external clock, available on the SCK pin

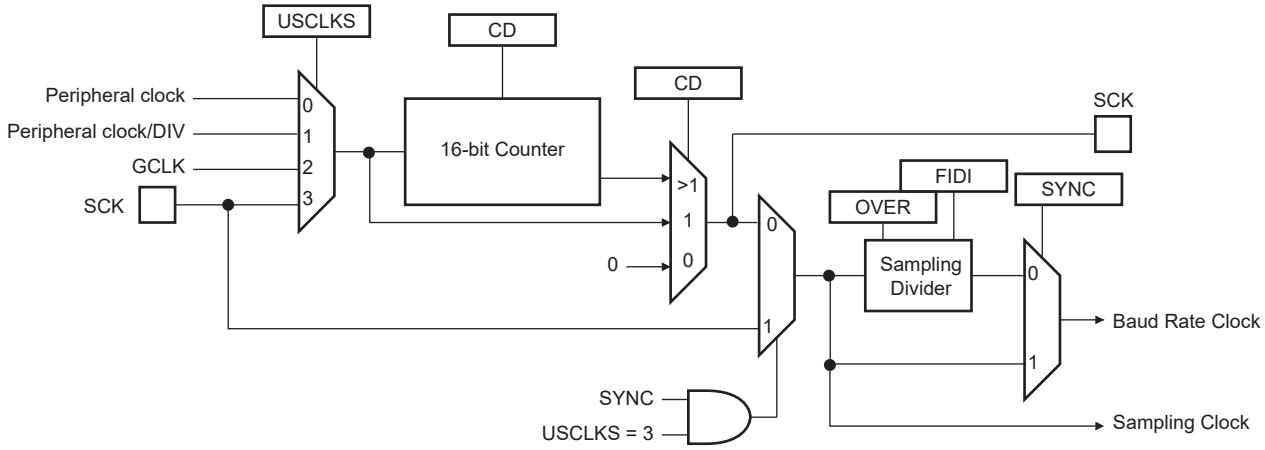
The baud rate generator is based upon a 16-bit divider, which is programmed with the CD field of the Baud Rate Generator Register (FLEX_US_BRGR). If a zero is written to CD, the baud rate generator does not generate any clock. If a one is written to CD, the divider is bypassed and becomes inactive.

If the external SCK clock is selected, the duration of the low and high levels of the signal provided on the SCK pin must be longer than a peripheral clock period. The frequency of the signal provided on SCK must be at least three times lower than peripheral clock in USART mode (field USART_MODE differs from 0xE or 0xF) or six times lower in SPI mode (field USART_MODE equals 0xE or 0xF).

If GCLK is selected, the baud rate is independent of the processor/peripheral clock and thus processor/peripheral clock frequency can be changed without affecting the USART transfer. The GCLK frequency must be at least three times lower than peripheral clock frequency.

If GCLK is selected (USCLKS = 2) and the SCK pin is driven (CLKO = 1), the CD field must be greater than 1.

**Figure 44-2. Baud Rate Generator**




### 44.7.1.1 Baud Rate in Asynchronous Mode

If the USART is programmed to operate in Asynchronous mode, the selected clock is first divided by CD, which is field-programmed in FLEX_US_BRGR. The resulting clock is provided to the receiver as a sampling clock and then divided by 16 or 8, depending on the programming of the OVER bit in FLEX_US_MR.

If OVER is set, the receiver sampling is eight times higher than the baud rate clock. If OVER is cleared, the sampling is performed at 16 times the baud rate clock.

The baud rate is calculated as per the following formula:

$$\text{Baud rate} = \frac{\text{Selected Clock}}{(8(2-OVER)CD)}$$

This gives a maximum baud rate of peripheral clock divided by 8, assuming that peripheral clock is the highest possible clock and that the OVER bit is set.

#### Baud Rate Calculation Example

Table 44-5 shows calculations of CD to obtain a baud rate at 38,400 bit/s for different source clock frequencies. This table also shows the actual resulting baud rate and the error.

**Table 44-5. Baud Rate Example (OVER = 0)**

| Source Clock (MHz) | Expected Baud Rate (bit/s) | Calculation Result | CD | Actual Baud Rate (bit/s) | Error |
|---|---|---|---|---|---|
| 3,686,400 | 38,400 | 6.00 | 6 | 38,400.00 | 0.00% |
| 4,915,200 | 38,400 | 8.00 | 8 | 38,400.00 | 0.00% |
| 5,000,000 | 38,400 | 8.14 | 8 | 39,062.50 | 1.70% |
| 7,372,800 | 38,400 | 12.00 | 12 | 38,400.00 | 0.00% |
| 8,000,000 | 38,400 | 13.02 | 13 | 38,461.54 | 0.16% |
| 12,000,000 | 38,400 | 19.53 | 20 | 37,500.00 | 2.40% |
| 12,288,000 | 38,400 | 20.00 | 20 | 38,400.00 | 0.00% |
| 14,318,180 | 38,400 | 23.30 | 23 | 38,908.10 | 1.31% |
| 14,745,600 | 38,400 | 24.00 | 24 | 38,400.00 | 0.00% |
| 18,432,000 | 38,400 | 30.00 | 30 | 38,400.00 | 0.00% |
| 24,000,000 | 38,400 | 39.06 | 39 | 38,461.54 | 0.16% |

**Table 44-5. Baud Rate Example (OVER = 0) (Continued)**

| Source Clock (MHz) | Expected Baud Rate (bit/s) | Calculation Result | CD | Actual Baud Rate (bit/s) | Error |
|---|---|---|---|---|---|
| 24,576,000 | 38,400 | 40.00 | 40 | 38,400.00 | 0.00% |
| 25,000,000 | 38,400 | 40.69 | 40 | 38,109.76 | 0.76% |
| 32,000,000 | 38,400 | 52.08 | 52 | 38,461.54 | 0.16% |
| 32,768,000 | 38,400 | 53.33 | 53 | 38,641.51 | 0.63% |
| 33,000,000 | 38,400 | 53.71 | 54 | 38,194.44 | 0.54% |
| 40,000,000 | 38,400 | 65.10 | 65 | 38,461.54 | 0.16% |
| 50,000,000 | 38,400 | 81.38 | 81 | 38,580.25 | 0.47% |

The baud rate is calculated with the following formula:

Baud rate = MCK / CD × 16

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

$$Error = 1 - \left(\frac{\text{Expected Baud Rate}}{\text{Actual Baud Rate}}\right)$$

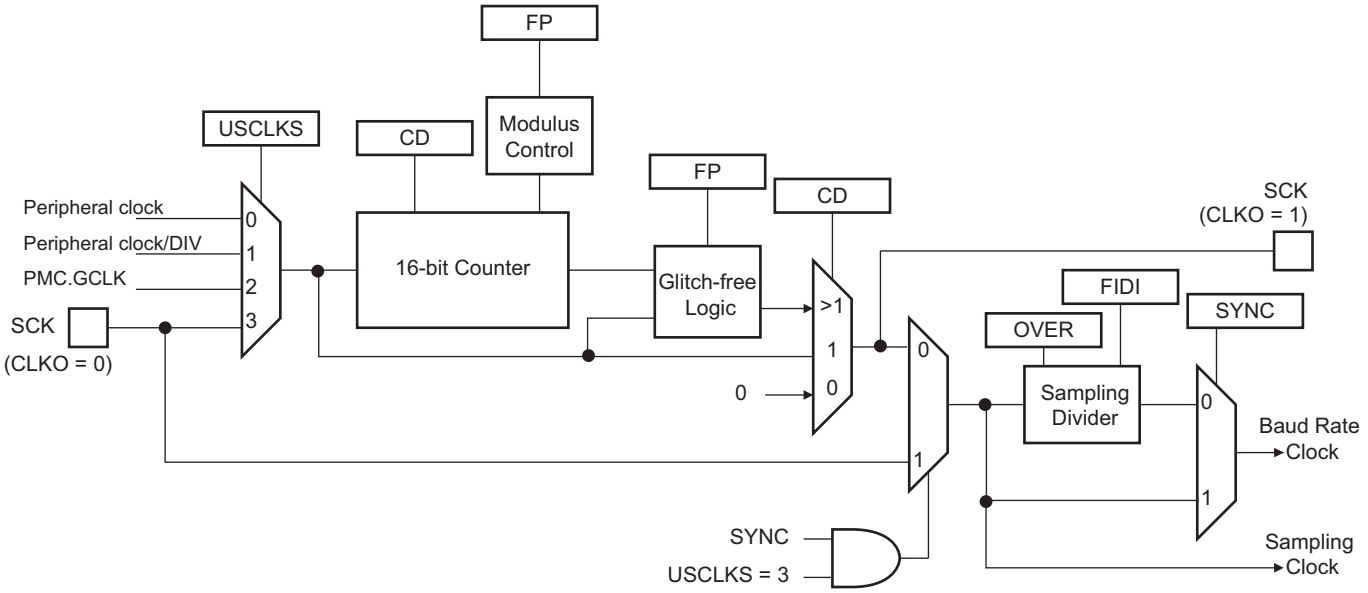
#### 44.7.1.2 Fractional Baud Rate in Asynchronous Mode

The baud rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain baud rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in FLEX_US_BRGR. If FP is not 0, the fractional part is activated. The resolution is one eighth of the clock divider. The fractional baud rate is calculated using the following formula:

$$\text{Baud rate} = \frac{\text{Selected Clock}}{\left(8(2 - OVER)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented in Figure 44-3.

**Figure 44-3. Fractional Baud Rate Generator**




**Warning:** When the value of field FP is greater than 0, the SCK (oversampling clock) generates non-constant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of the CD field.

#### 44.7.1.3 Baud Rate in Synchronous Mode or SPI Mode

If the USART is programmed to operate in Synchronous mode, the selected clock is simply divided by the CD field in FLEX_US_BRGR:

$$\text{Baud rate} = \frac{\text{Selected Clock}}{CD}$$

In Synchronous mode, if the external clock is selected (USCLKS = 3) and CLKO = 0 (Slave mode), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in FLEX_US_BRGR has no effect. The external clock frequency must be at least three times lower than the system clock. In Synchronous mode master (USCLKS = 0 or 1, CLKO = 1), the receive part limits the SCK maximum frequency to $f_{peripheral\ clock}/3$ in USART mode, or $f_{peripheral\ clock}/6$ in SPI mode.

When either the external clock SCK or the internal clock divided (peripheral clock/DIV or GCLK) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. If the peripheral clock is selected, the baud rate generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

#### 44.7.1.4 Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 44-6.

**Table 44-6. Binary and Decimal Values for Di**

| DI field | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 1000 | 1001 |
|---|---|---|---|---|---|---|---|---|
| Di (decimal) | 1 | 2 | 4 | 8 | 16 | 32 | 12 | 20 |

Fi is a binary value encoded on a 4-bit field, named FI, as represented in Table 44-7.

**Table 44-7. Binary and Decimal Values for Fi**

| FI field | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 1001 | 1010 | 1011 | 1100 | 1101 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fi (decimal) | 372 | 372 | 558 | 744 | 1116 | 1488 | 1860 | 512 | 768 | 1024 | 1536 | 2048 |

Table 44-8 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

**Table 44-8. Possible Values for the Fi/Di Ratio**

| Fi/Di | 372 | 558 | 744 | 1116 | 1488 | 1806 | 512 | 768 | 1024 | 1536 | 2048 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 372 | 558 | 744 | 1116 | 1488 | 1860 | 512 | 768 | 1024 | 1536 | 2048 |
| 2 | 186 | 279 | 372 | 558 | 744 | 930 | 256 | 384 | 512 | 768 | 1024 |
| 4 | 93 | 139.5 | 186 | 279 | 372 | 465 | 128 | 192 | 256 | 384 | 512 |
| 8 | 46.5 | 69.75 | 93 | 139.5 | 186 | 232.5 | 64 | 96 | 128 | 192 | 256 |
| 16 | 23.25 | 34.87 | 46.5 | 69.75 | 93 | 116.2 | 32 | 48 | 64 | 96 | 128 |
| 32 | 11.62 | 17.43 | 23.25 | 34.87 | 46.5 | 58.13 | 16 | 24 | 32 | 48 | 64 |
| 12 | 31 | 46.5 | 62 | 93 | 124 | 155 | 42.66 | 64 | 85.33 | 128 | 170.6 |
| 20 | 18.6 | 27.9 | 37.2 | 55.8 | 74.4 | 93 | 25.6 | 38.4 | 51.2 | 76.8 | 102.4 |

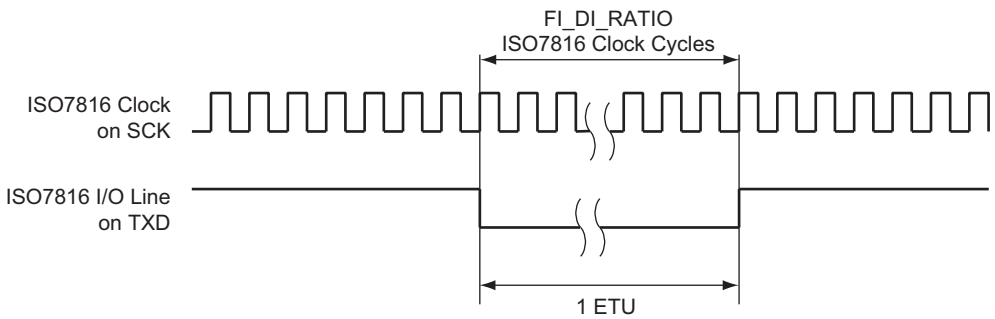
If the USART is configured in ISO7816 mode, the clock selected by the USCLKS field in FLEX_US_MR is first divided by the value programmed in field CD field in FLEX_US_BRGR. The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in FLEX_US_MR.

This clock is then divided by the value programmed in the FI_DI_RATIO field in the FI DI Ratio Register (FLEX_US_FIDI). This is performed by the Sampling Divider, which performs a division by up to 65535 in ISO7816 mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program the FI_DI_RATIO field to a value as close as possible to the expected value.

The FI_DI_RATIO field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate (Fi = 372, Di = 1).

Figure 44-4 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

**Figure 44-4. Elementary Time Unit (ETU)**



### 44.7.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the USART Control Register (FLEX_US_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in FLEX_US_CR. However, the transmitter registers can be programmed before being enabled.

The receiver and the transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in FLEX_US_CR. The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in FLEX_US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the USART Transmit Holding Register (FLEX_US_THR). If a timeguard is programmed, it is handled normally.

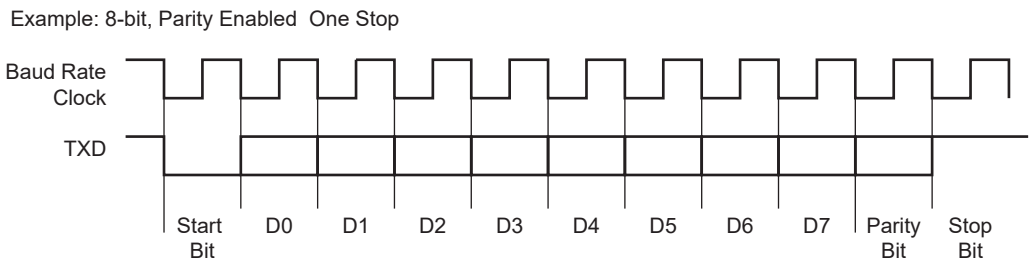
### 44.7.3 Synchronous and Asynchronous Modes

#### 44.7.3.1 Transmitter Operations

The transmitter performs the same in both Synchronous and Asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, 1 optional parity bit and up to 2 stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in FLEX_US_MR. Nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The parity bit is set according to the PAR field in FLEX_US_MR. The even, odd, space, marked or none parity bit can be configured. The MSBF bit in FLEX_US_MR configures which data bit is sent first. If written to 1, the most significant bit is sent first. If written to 0, the less significant bit is sent first. The number of stop bits is selected by the NBSTOP field in FLEX_US_MR. The 1.5 stop bit is supported in Asynchronous mode only.

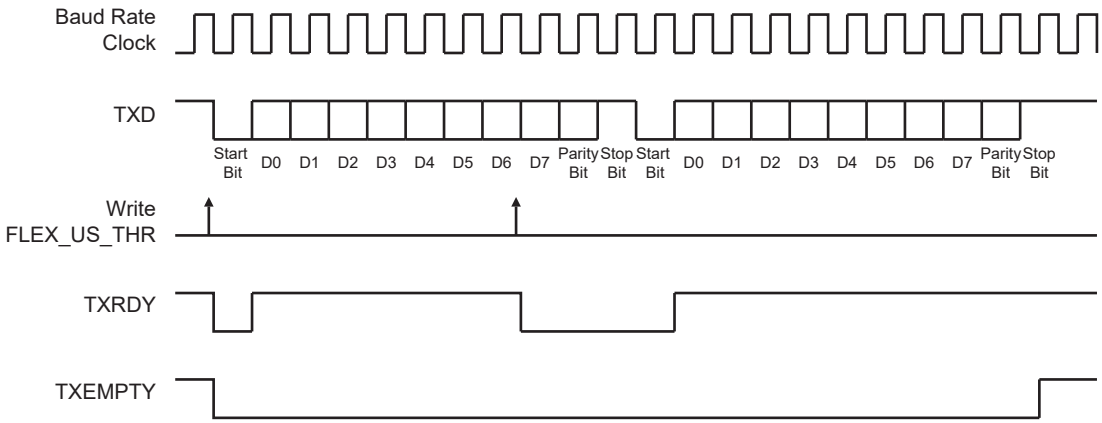
**Figure 44-5. Character Transmit**




The characters are sent by writing in FLEX_US_THR. The transmitter reports two status bits in the USART Channel Status Register (FLEX_US_CSR): TXRDY (Transmitter Ready), which indicates that FLEX_US_THR is empty and TXEMPTY, which indicates that all the characters written in FLEX_US_THR have been processed. When the current character processing is completed, the last character written in FLEX_US_THR is transferred into the Shift register of the transmitter and FLEX_US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in FLEX_US_THR while TXRDY is low has no effect and the written character is lost.

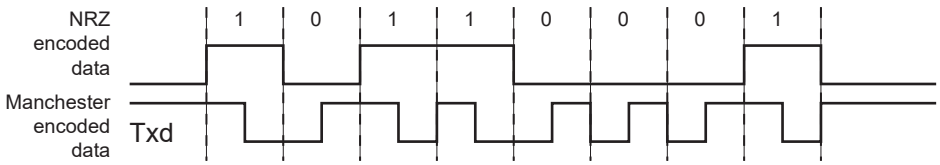
**Figure 44-6. Transmitter Status**




### 44.7.3.2 Manchester Encoder

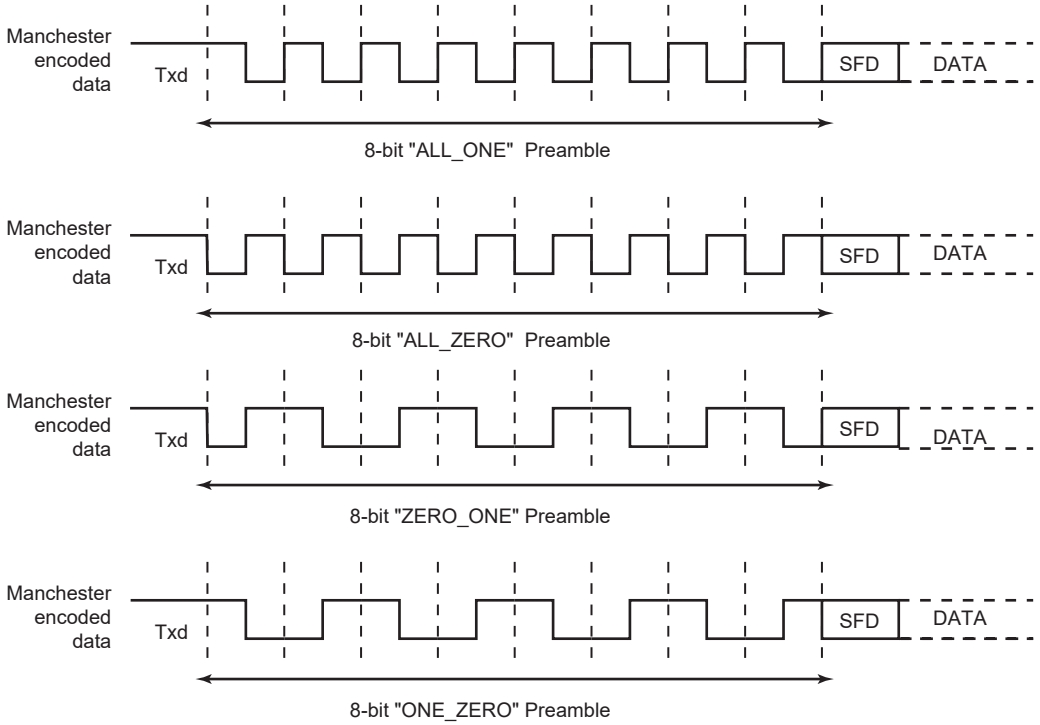
When the Manchester encoder is in use, characters transmitted through the USART are encoded based on biphase Manchester II format. To enable this mode, set the MAN bit in FLEX_US_MR to 1. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero or zero-to-one. Thus, a transition always occurs at the midpoint of each bit time. It consumes more bandwidth than the original NRZ signal (2x) but the receiver has more error control since the expected input must show a change at the center of a bit cell. An example of Manchester encoded sequence is: the byte 0xB1 or 10110001 encodes to 10 01 10 10 01 01 01 10, assuming the default polarity of the encoder. Figure 44-7 illustrates this coding scheme.

**Figure 44-7. NRZ to Manchester Encoding**



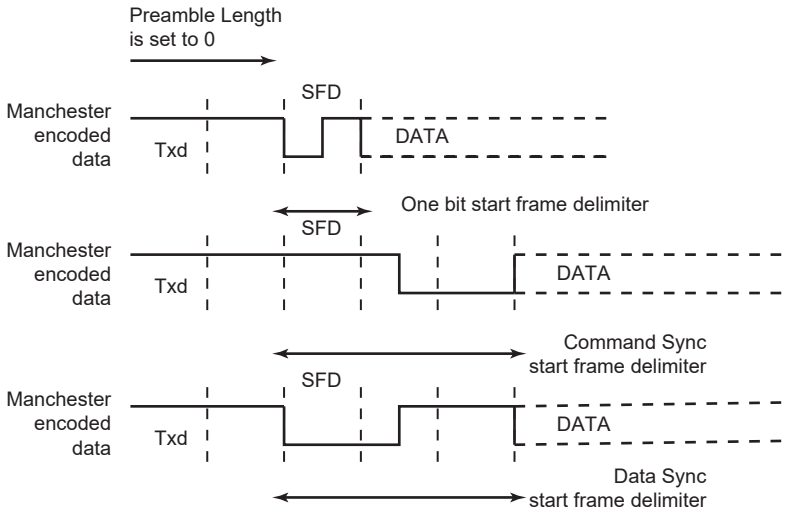

The Manchester encoded character can also be encapsulated by adding both a configurable preamble and a start frame delimiter pattern. Depending on the configuration, the preamble is a training sequence, composed of a predefined pattern with a programmable length from 1 to 15 bit times. If the preamble length is set to 0, the preamble waveform is not generated prior to any character. The preamble pattern is chosen among the following sequences: ALL_ONE, ALL_ZERO, ONE_ZERO or ZERO_ONE, writing the TX_PP field in FLEX_US_MAN register. The TX_PL field is used to configure the preamble length. Figure 44-8 illustrates and defines the valid patterns. To improve flexibility, the encoding scheme can be configured using the TX_MPOL bit in the FLEX_US_MAN register. If the TX_MPOL bit is set to zero (default), a logic zero is encoded with a zero-to-one transition and a logic one is encoded with a one-to-zero transition. If the TX_MPOL bit is set to one, a logic one is encoded with a one-to-zero transition and a logic zero is encoded with a zero-to-one transition.

**Figure 44-8. Preamble Patterns, Default Polarity Assumed**




A start frame delimiter is to be configured using the ONEBIT bit in FLEX_US_MR. It consists of a user-defined pattern that indicates the beginning of a valid data. Figure 44-9 illustrates these patterns. If the start frame delimiter, also known as the start bit, is one bit, (ONEBIT = 1), a logic zero is Manchester encoded and indicates that a new character is being sent serially on the line. If the start frame delimiter is a synchronization pattern also referred to as sync (ONEBIT = 0), a sequence of three bit times is sent serially on the line to indicate the start of a new character. The sync waveform is in itself an invalid Manchester waveform as the transition occurs at the middle of the second bit time. Two distinct sync patterns are used: the command sync and the data sync. The command sync has a logic one level for one and a half bit times, then a transition to logic zero for the second one and a half bit times. If the MODSYNC bit in FLEX_US_MR is set to 1, the next character is a command. If it is set to 0, the next character is a data. When direct memory access is used, the MODSYNC bit can be immediately updated with a modified character located in memory. To enable this mode, VAR_SYNC bit in FLEX_US_MR must be set. In this case, the MODSYNC bit in FLEX_US_MR is bypassed and the sync configuration is held in the TXSYNH in FLEX_US_THR. The USART character format is modified and includes sync information.

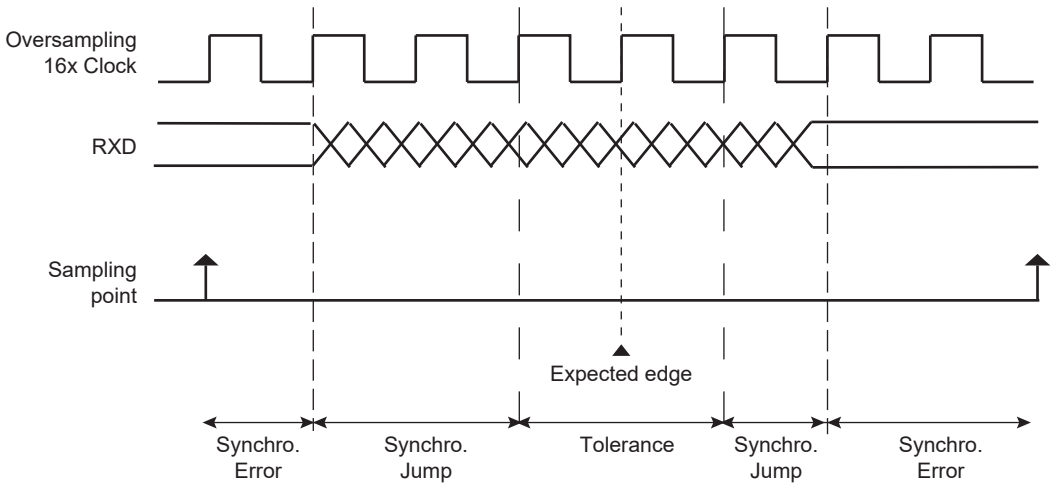
**Figure 44-9. Start Frame Delimiter**




### Drift Compensation

Drift compensation is available only in 16X Oversampling mode. An hardware recovery system allows a larger clock drift. To enable the hardware system, the bit in the USART_MAN register must be set. If the RXD edge is one 16X clock cycle from the expected edge, this is considered as normal jitter and no corrective actions is taken. If the RXD event is between 4 and 2 clock cycles before the expected edge, then the current period is shortened by one clock cycle. If the RXD event is between 2 and 3 clock cycles after the expected edge, then the current period is lengthened by one clock cycle. These intervals are considered to be drift and so corrective actions are automatically taken.

**Figure 44-10. Bit Resynchronization**




#### 44.7.3.3 Asynchronous Receiver

If the USART is programmed in Asynchronous operating mode (SYNC = 0), the receiver oversamples the RXD input line. The oversampling is either 16 or 8 times the baud rate clock, depending on the OVER bit in FLEX_US_MR.

The receiver samples the RXD line. If the line is sampled during one half of a bit time to 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

If the oversampling is 16 (OVER = 0), a start is detected at the eighth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 16 oversampling clock cycles. If the oversampling is 8 (OVER = 1), a start bit is detected at the fourth sample to 0. Data bits, parity bit and stop bit are assumed to have a duration corresponding to 8 oversampling clock cycles.

The number of data bits, first bit sent and Parity mode are selected by the same fields and bits as the transmitter, i.e., respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the NBSTOP field, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 44-11 and Figure 44-12 illustrate start detection and character reception when USART operates in Asynchronous mode.

**Figure 44-11. Asynchronous Start Detection**

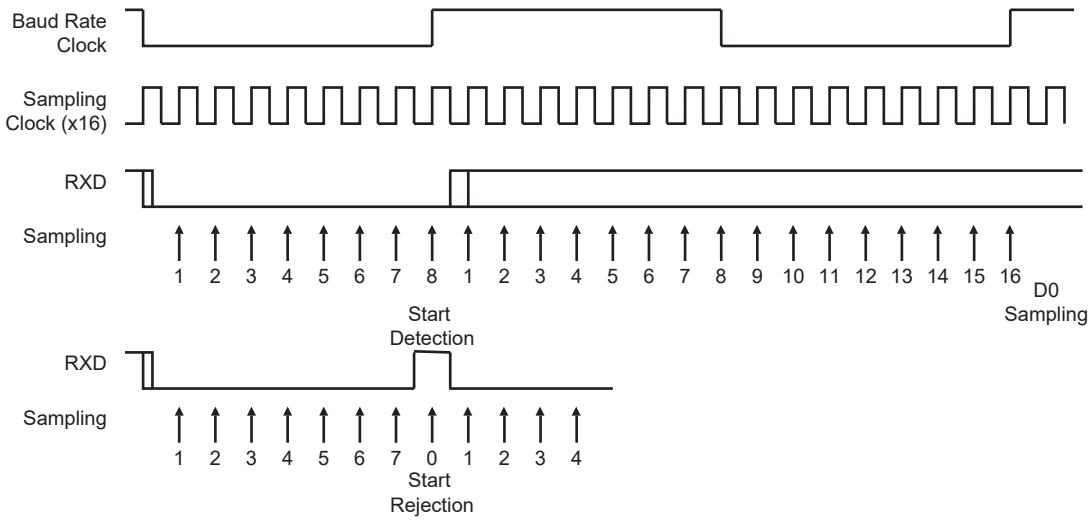
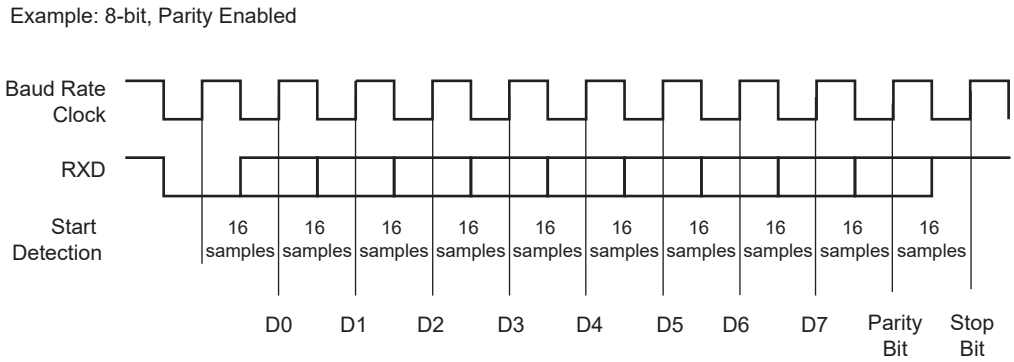



**Figure 44-12. Asynchronous Character Reception**




### 44.7.3.4 Manchester Decoder

When the MAN bit in FLEX_US_MR is set, the Manchester decoder is enabled. The decoder performs both preamble and start frame delimiter detection. One input line is dedicated to Manchester encoded input data.

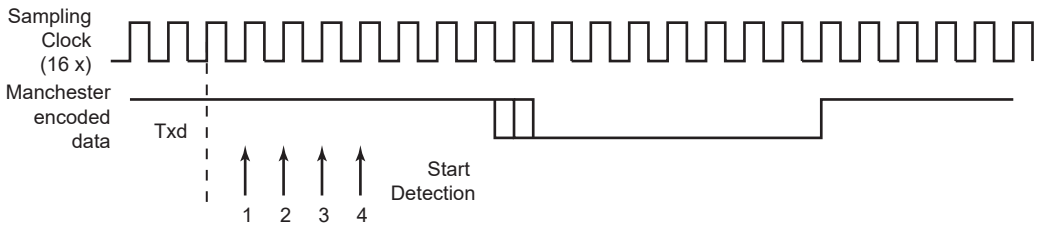
An optional preamble sequence can be defined. Its length is user-defined and totally independent of the transmitter side. Use RX_PL in FLEX_US_MAN register to configure the length of the preamble sequence. If the length is set to 0, no preamble is detected and the function is disabled. In addition, the polarity of the input stream

is programmable with RX_MPOL bit in FLEX_US_MAN register. Depending on the desired application the preamble pattern matching is to be defined via the RX_PP field in FLEX_US_MAN. See Figure 44-8 for available preamble patterns.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. So, if ONEBIT bit = 1, only a zero encoded Manchester can be detected as a valid start frame delimiter. If ONEBIT = 0, only a sync pattern is detected as a valid start frame delimiter. Decoder operates by detecting transition on incoming stream. If RXD is sampled during one quarter of a bit time to zero, a start bit is detected. See Figure 44-13. The sample pulse rejection mechanism applies.

The RXIDLEV bit in FLEX_US_MAN informs the USART of the receiver line idle state value (receiver line inactive). The user must define RXIDLEV to ensure reliable synchronization. By default, RXIDLEV is set to one (receiver line is at level 1 when there is no activity).

**Figure 44-13. Asynchronous Start Bit Detection**




The receiver is activated and starts preamble and frame delimiter detection, sampling the data at one quarter and then three quarters. If a valid preamble pattern or start frame delimiter is detected, the receiver continues decoding with the same synchronization. If the stream does not match a valid pattern or a valid start frame delimiter, the receiver resynchronizes on the next valid edge.The minimum time threshold to estimate the bit value is three quarters of a bit time.

If a valid preamble (if used) followed with a valid start frame delimiter is detected, the incoming stream is decoded into NRZ data and passed to USART for processing. Figure 44-14 illustrates Manchester pattern mismatch. When incoming data stream is passed to the USART, the receiver is also able to detect Manchester code violation. A code violation is a lack of transition in the middle of a bit cell. In this case, the MANE flag in FLEX_US_CSR is raised. It is cleared by writing a one to the RSTSTA bit in FLEX_US_CR. See Figure 44-15 for an example of Manchester error detection during the data phase.

**Figure 44-14. Preamble Pattern Mismatch**

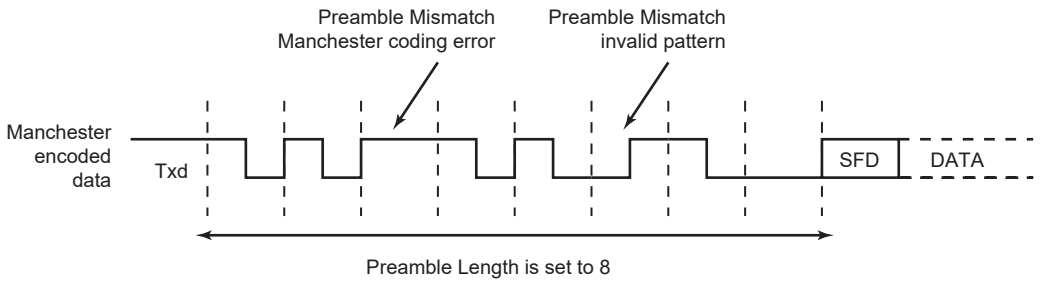
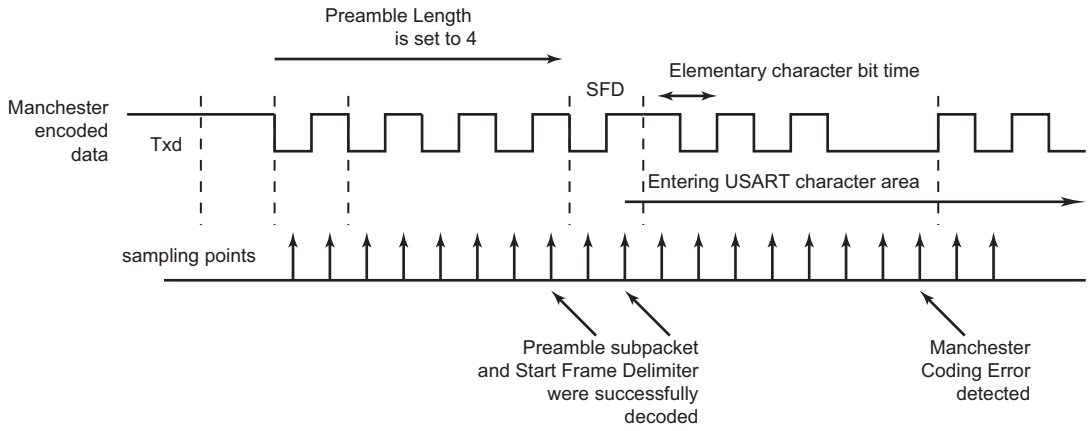


**Figure 44-15. Manchester Error Flag**




When the start frame delimiter is a sync pattern (ONEBIT = 0), both command and data delimiter are supported. If a valid sync is detected, the received character is written as RXCHR field in the Receive Holding Register (FLEX_US_RHR) and the RXSYNH is updated. RXCHR is set to 1 when the received character is a command, and it is set to 0 if the received character is a data. This mechanism alleviates and simplifies the direct memory access as the character contains its own sync field in the same register.

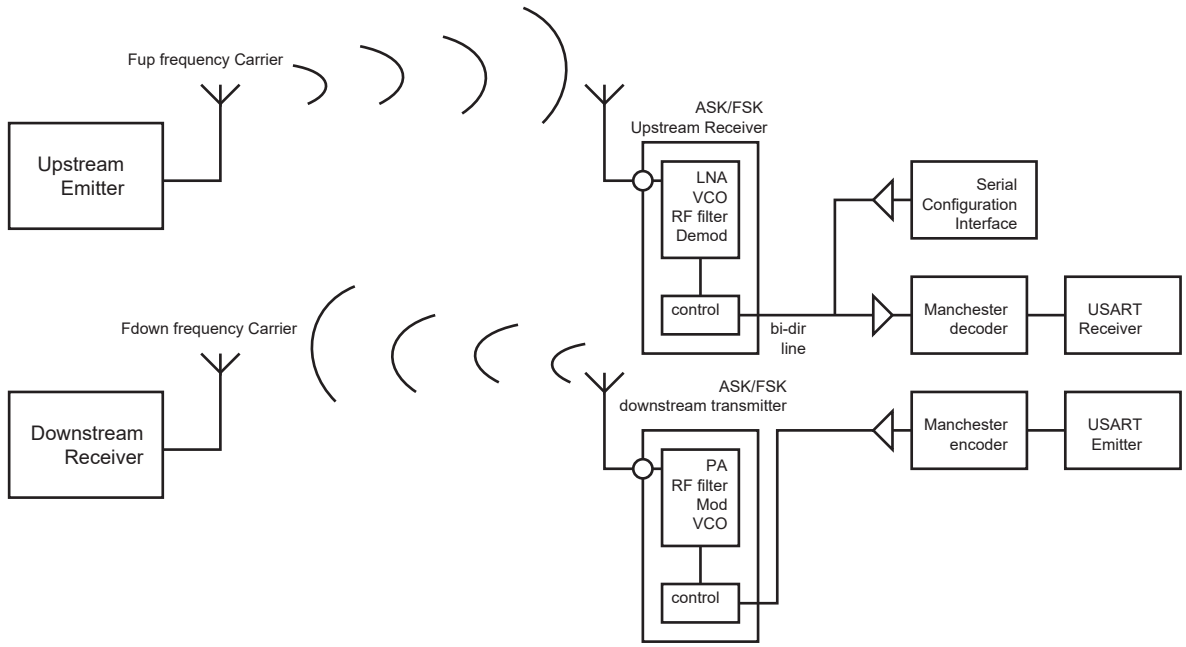
As the decoder is setup to be used in Unipolar mode, the first bit of the frame has to be a zero-to-one transition.

### 44.7.3.5 Radio Interface: Manchester Encoded USART Application

This section describes low data rate RF transmission systems and their integration with a Manchester encoded USART. These systems are based on transmitter and receiver ICs that support ASK and FSK modulation schemes.

The goal is to perform full duplex radio transmission of characters using two different frequency carriers. See the configuration in Figure 44-16.

**Figure 44-16. Manchester Encoded Characters RF Transmission**



The USART peripheral is configured as a Manchester encoder/decoder. Looking at the downstream communication channel, Manchester encoded characters are serially sent to the RF transmitter. This may also include a user defined preamble and a start frame delimiter. Mostly, preamble is used in the RF receiver to distinguish between a valid data from a transmitter and signals due to noise. The Manchester stream is then modulated. See Figure 44-17 for an example of ASK modulation scheme. When a logic one is sent to the ASK modulator, the power amplifier, referred to as PA, is enabled and transmits an RF signal at downstream frequency. When a logic zero is transmitted, the RF signal is turned off. If the FSK modulator is activated, two different frequencies are used to transmit data. When a logic 1 is sent, the modulator outputs an RF signal at frequency F0 and switches to F1 if the data sent is a 0. See Figure 44-18.

From the receiver side, another carrier frequency is used. The RF receiver performs a bit check operation examining demodulated data stream. If a valid pattern is detected, the receiver switches to Receiving mode. The demodulated stream is sent to the Manchester decoder. Because of bit checking inside RF IC, the data transferred to the microcontroller is reduced by a user-defined number of bits. The Manchester preamble length is to be defined in accordance with the RF IC configuration.

**Figure 44-17. ASK Modulator Output**

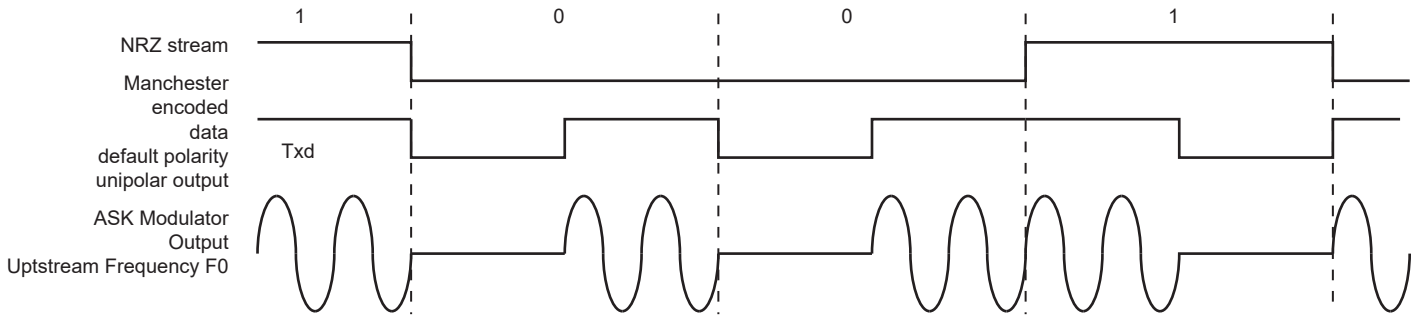
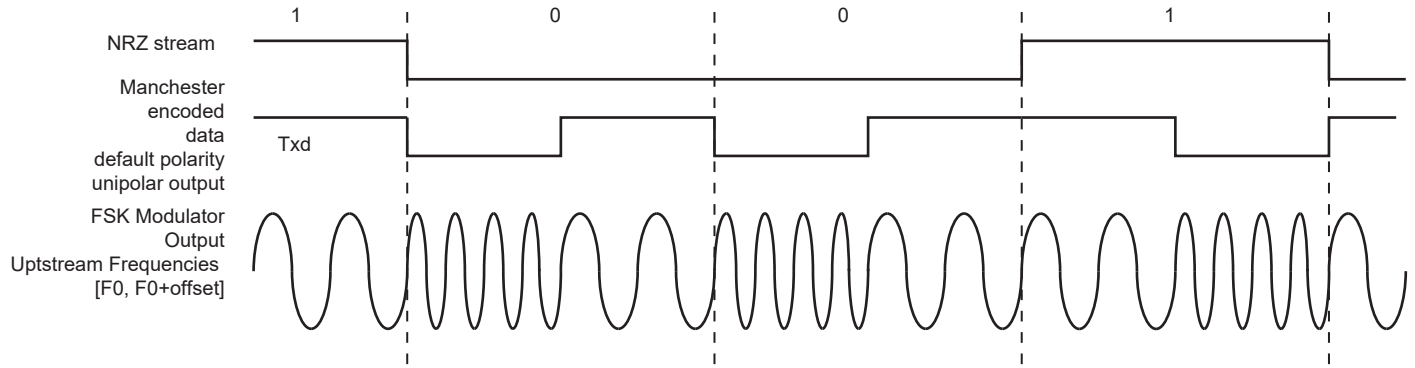



**Figure 44-18. FSK Modulator Output**



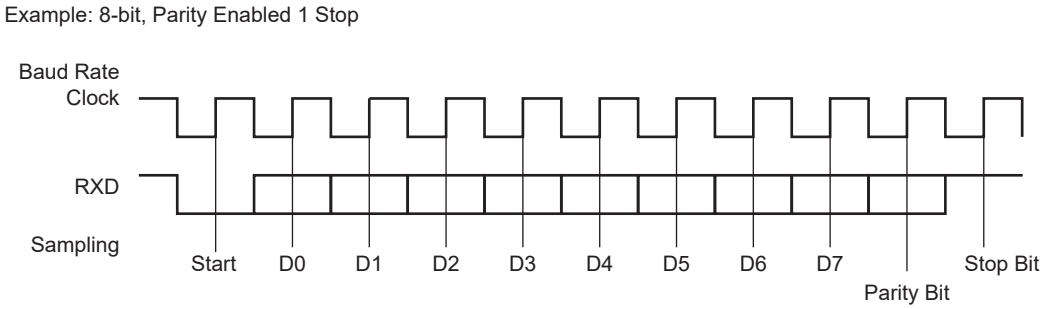

### 44.7.3.6 Synchronous Receiver

In Synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the baud rate clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high-speed transfer capability.

Configuration fields and bits are the same as in Asynchronous mode.

Figure 44-19 illustrates a character reception in Synchronous mode.

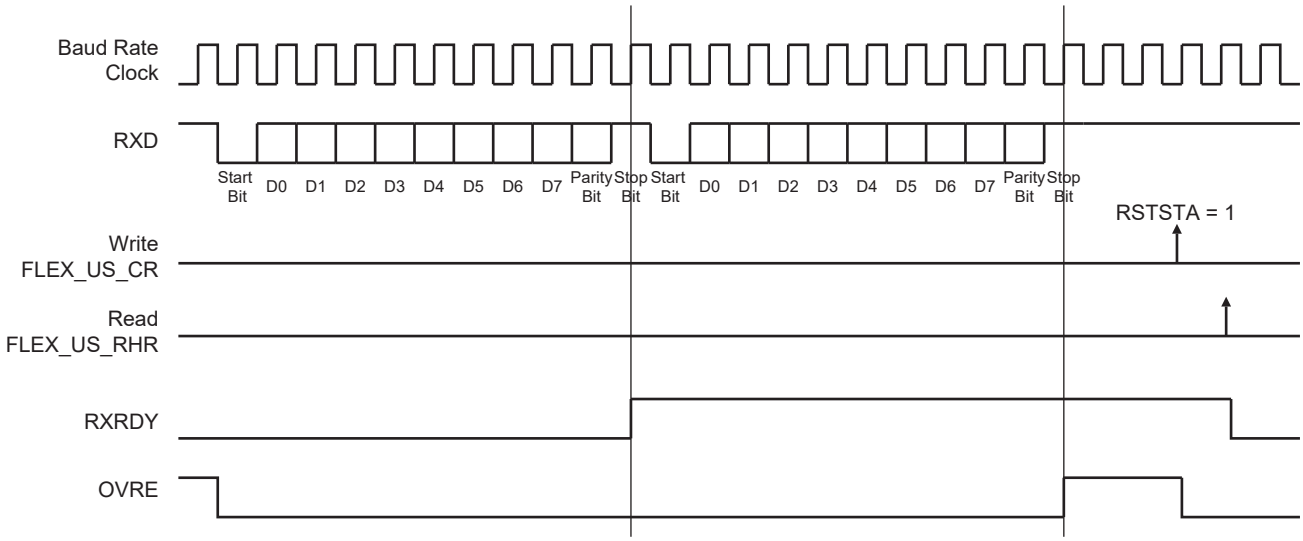
**Figure 44-19. Synchronous Mode Character Reception**




### 44.7.3.7 Receiver Operations

When a character reception is completed, it is transferred to the Receive Holding Register (FLEX_US_RHR) and the RXRDY bit in FLEX_US_CSR rises. If a character is completed while the RXRDY is set, the Overrun Error (OVRE) bit is set. The last character is transferred into FLEX_US_RHR and overwrites the previous one. The OVRE bit is cleared by writing a one to the Reset Status (RSTSTA) bit in FLEX_US_CR.

**Figure 44-20. Receiver Status**




### 44.7.3.8 Parity

The USART supports five parity modes that are selected by writing to the PAR field in FLEX_US_MR. The PAR field also enables the Multidrop mode (see Section 44.7.3.9 “Multidrop Mode”). Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit to 0 if a number of 1s in the character data bit is even, and to 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit to 1 if a number of 1s in the character data bit is even, and to 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit to 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 0. If the space parity is used, the parity generator of the transmitter drives the parity bit to 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled to 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

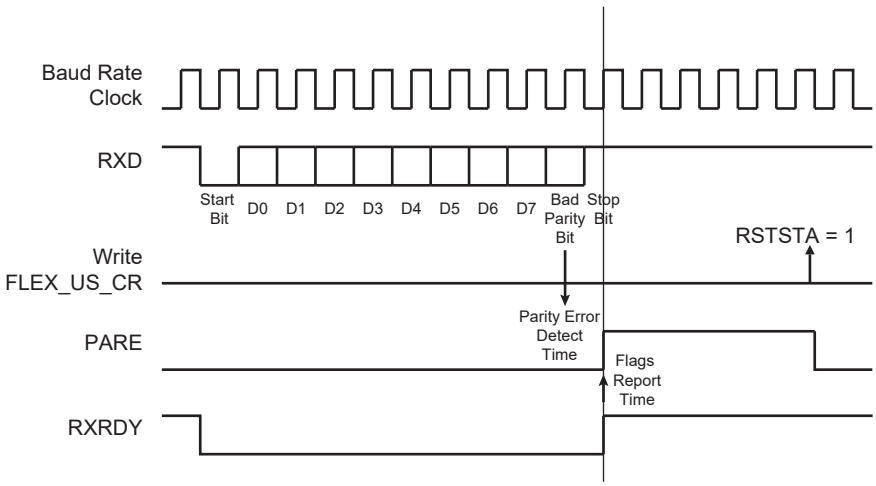
Table 44-9 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits set to 1 in the character value, the parity bit is set to 1 when the parity is odd, or configured to 0 when the parity is even.

**Table 44-9. Parity Bit Examples**

| Character | Hexadecimal | Binary | Parity Bit | Parity Mode |
|---|---|---|---|---|
| A | 0x41 | 0100 0001 | 1 | Odd |
| A | 0x41 | 0100 0001 | 0 | Even |
| A | 0x41 | 0100 0001 | 1 | Mark |
| A | 0x41 | 0100 0001 | 0 | Space |
| A | 0x41 | 0100 0001 | None | None |

When the receiver detects a parity error, it sets the PARE (Parity Error) bit in FLEX_US_CSR. The PARE bit can be cleared by writing a one to the RSTSTA bit in FLEX_US_CR. Figure 44-21 illustrates the parity bit status setting and clearing.

**Figure 44-21. Parity Error**




### 44.7.3.9 Multidrop Mode

If the value 0x6 or 0x07 is written to the PAR field in FLEX_US_MR, the USART runs in Multidrop mode. This mode differentiates the data characters and the address characters. Data are transmitted with the parity bit to 0 and addresses are transmitted with the parity bit to 1.

If the USART is configured in Multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when a one is written to the SENTA bit in FLEX_US_CR.

To handle parity error, the PARE bit is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

The transmitter sends an address byte (parity bit set) when SENDA is written to 1 in FLEX_US_CR. In this case, the next byte written to FLEX_US_THR is transmitted as an address. Any character written in FLEX_US_THR when the SENDA command is not written is transmitted normally with parity to 0.

### 44.7.3.10 Transmitter Timeguard

The timeguard feature enables the USART interface with slow remote devices.

The timeguard function enables the transmitter to insert an idle state on the TXD line between two characters. This idle state actually acts as a long stop bit.

The duration of the idle state is programmed in the TG field of the Transmitter Timeguard Register (FLEX_US_TTGR). When this field is written to zero no timeguard is generated. Otherwise, the transmitter holds a high level on TXD after each transmitted byte during the number of bit periods programmed in TG in addition to the number of stop bits.

As illustrated in Figure 44-22, the behavior of TXRDY and TXEMPTY status bits is modified by the programming of a timeguard. TXRDY rises only when the start bit of the next character is sent, and thus remains to 0 during the timeguard transmission if a character has been written in FLEX_US_THR. TXEMPTY remains low until the timeguard transmission is completed as the timeguard is part of the current character being transmitted.

**Figure 44-22. Timeguard Operations**

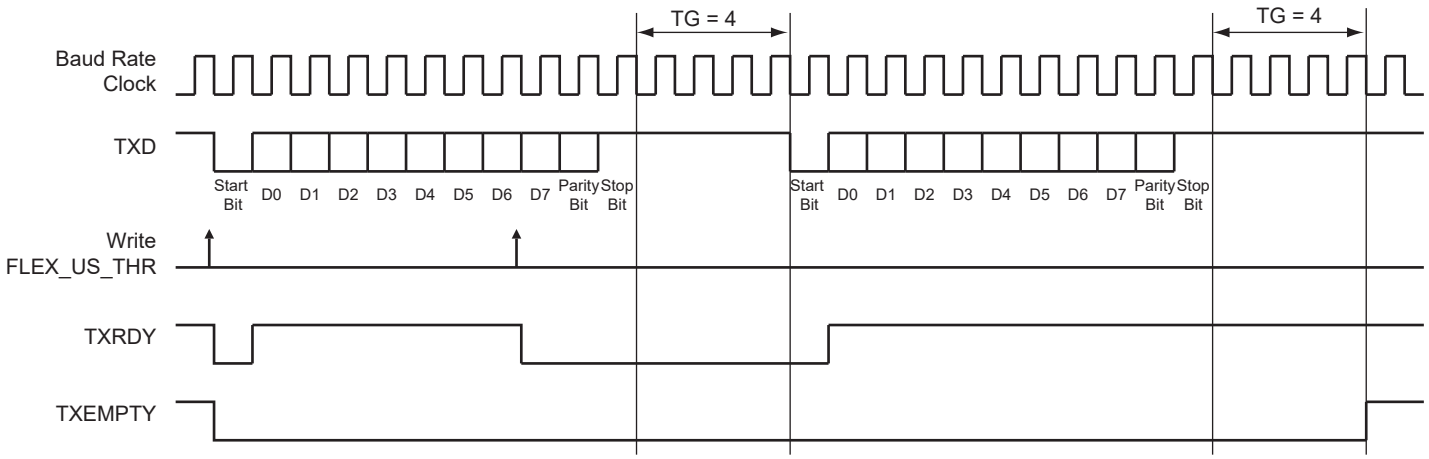



Table 44-10 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the baud rate.

**Table 44-10. Maximum Timeguard Length Depending on Baud Rate**

| Baud Rate (bit/s) | Bit Time (µs) | Timeguard (ms) |
|---|---|---|
| 1,200 | 833 | 212.50 |
| 9,600 | 104 | 26.56 |
| 14,400 | 69.4 | 17.71 |
| 19,200 | 52.1 | 13.28 |
| 28,800 | 34.7 | 8.85 |
| 38,400 | 26 | 6.63 |
| 56,000 | 17.9 | 4.55 |
| 57,600 | 17.4 | 4.43 |
| 115,200 | 8.7 | 2.21 |

#### 44.7.3.11 Receiver Timeout

The Receiver Timeout provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a timeout is detected, the TIMEOUT bit in FLEX_US_CSR rises and can generate an interrupt, thus indicating to the driver an end of frame.

The timeout delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Timeout Register (FLEX_US_RTOR). If the TO field is written to 0, the Receiver Timeout is disabled and no timeout is detected. The TIMEOUT bit in FLEX_US_CSR remains at 0. Otherwise, the receiver loads a 16-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time

a new character is received. If the counter reaches 0, the TIMEOUT bit in FLEX_US_CSR rises. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing a ‘1’ to FLEX_US_CR.STTTO. In this case, the idle state on RXD before a new character is received does not provide a timeout. This prevents having to handle an interrupt before a character is received and enables waiting for the next idle state on RXD after a frame is received.
- Obtain an interrupt while no character is received. This is performed by writing a ‘1’ to FLEX_US_CR.RETTO. In this case, the counter starts counting down immediately from the value TO. This generates a periodic interrupt so that a user timeout can be handled, for example when no key is pressed on a keyboard.

Figure 44-23 shows the block diagram of the Receiver Timeout feature.

**Figure 44-23. Receiver Timeout Block Diagram**

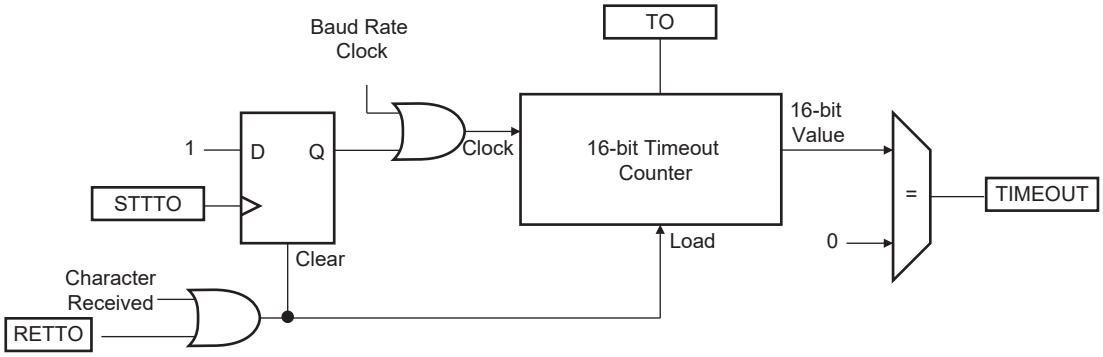



Table 44-11 gives the maximum timeout period for some standard baud rates.

**Table 44-11. Maximum Timeout Period**

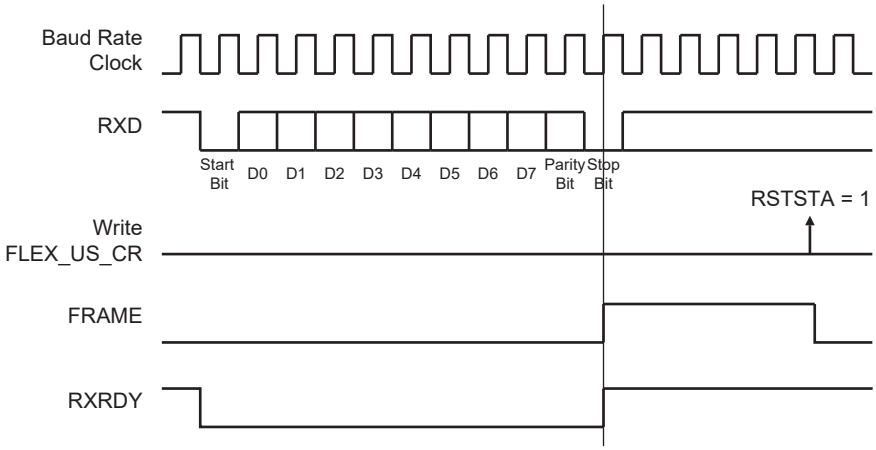
| Baud Rate (bit/s) | Bit Time (µs) | Timeout (ms) |
|---|---|---|
| 600 | 1,667 | 109,225 |
| 1,200 | 833 | 54,613 |
| 2,400 | 417 | 27,306 |
| 4,800 | 208 | 13,653 |
| 9,600 | 104 | 6,827 |
| 14,400 | 69 | 4,551 |
| 19,200 | 52 | 3,413 |
| 28,800 | 35 | 2,276 |
| 38,400 | 26 | 1,704 |
| 56,000 | 18 | 1,170 |
| 57,600 | 17 | 1,138 |
| 200,000 | 5 | 328 |

#### 44.7.3.12 Framing Error

The receiver is capable of detecting framing errors. A framing error happens when the stop bit of a received character is detected at level 0. This can occur if the receiver and the transmitter are fully desynchronized.

A framing error is reported on the FRAME bit of FLEX_US_CSR. The FRAME bit is asserted in the middle of the stop bit as soon as the framing error is detected. It is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

**Figure 44-24. Framing Error Status**




#### 44.7.3.13 Transmit Break

The user can request the transmitter to generate a break condition on the TXD line. A break condition drives the TXD line low during at least one complete character. It appears the same as a 0x00 character sent with the parity and the stop bits to 0. However, the transmitter holds the TXD line at least during one character until the user requests the break condition to be removed.

A break is transmitted by setting the STTBRK bit in FLEX_US_CR. This can be done at any time, either while the transmitter is empty (no character in either the Shift register or in FLEX_US_THR) or when a character is being transmitted. If a break is requested while a character is being shifted out, the character is first completed before the TXD line is held low.

Once the Start Break command is requested, further Start Break commands are ignored until the end of the break is completed.

The break condition is removed by setting the STPBRK bit in FLEX_US_CR. If the Stop Break command is requested before the end of the minimum break duration (one character, including start, data, parity and stop bits), the transmitter ensures that the break condition completes.

The transmitter considers the break as though it is a character, i.e., the Start Break and Stop Break commands are processed only if the TXRDY bit = 1 in FLEX_US_CSR and the start of the break condition clears the TXRDY and TXEMPTY bits as if a character is processed.

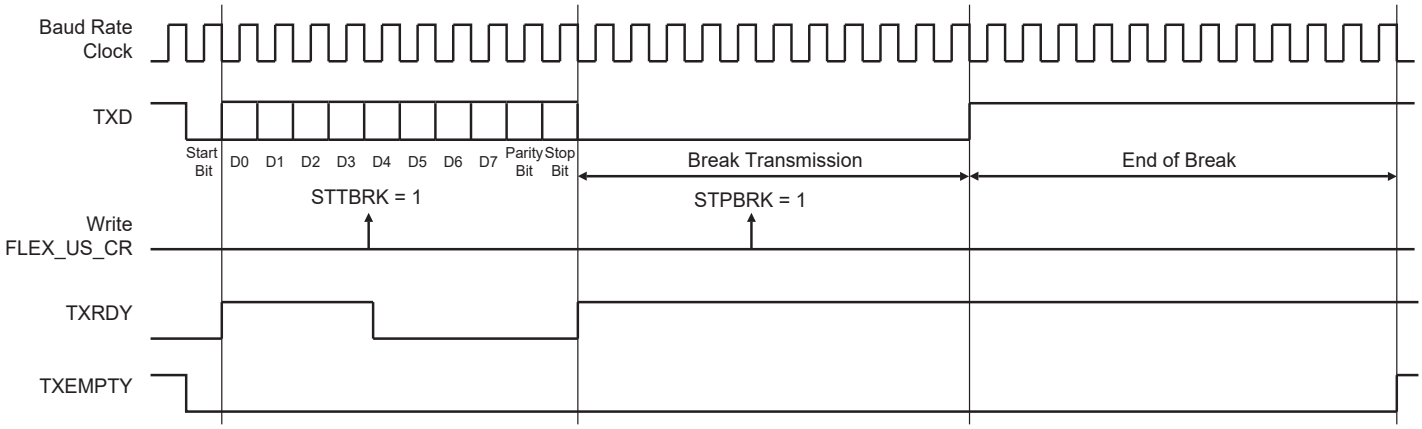
Setting both the STTBRK and STPBRK bits in FLEX_US_CR can lead to an unpredictable result. All Stop Break commands requested without a previous Start Break command are ignored. A byte written into the Transmit Holding register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

Figure 44-25 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

**Figure 44-25. Break Transmission**




#### 44.7.3.14 Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. This corresponds to detecting a framing error with data to 0x00, but FRAME remains low.

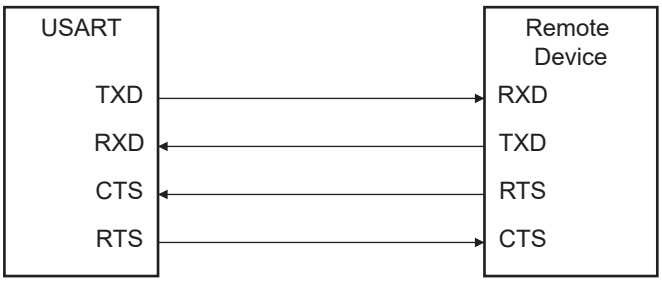
When the low stop bit is detected, the receiver asserts the RXBRK bit in FLEX_US_CSR. FLEX_US_CSR.RXBRK may be cleared by setting the RSTSTA bit in FLEX_US_CR.

An end of receive break is detected by a high level for at least 2/16ths of a bit period in Asynchronous operating mode or one sample at high level in Synchronous operating mode. The end of break detection also asserts the RXBRK bit.

#### 44.7.3.15 Hardware Handshaking

The USART features a hardware handshaking out-of-band flow control. The RTS and CTS pins are used to connect with the remote device, as shown in Figure 44-26.

**Figure 44-26. Connection with a Remote Device for Hardware Handshaking**




Setting the USART to operate with hardware handshaking is performed by writing the USART_MODE field in FLEX_US_MR to the value 0x2.

The USART behavior when hardware handshaking is enabled is the same as the behavior in standard Synchronous or Asynchronous mode, except that the receiver drives the RTS pin as described below and the level on the CTS pin modifies the behavior of the transmitter as described below. Using this mode requires using the DMAC channel for reception. The transmitter can handle hardware handshaking in any case.

**Figure 44-27. RTS line software control when FLEX_US_MR.USART_MODE = 2**

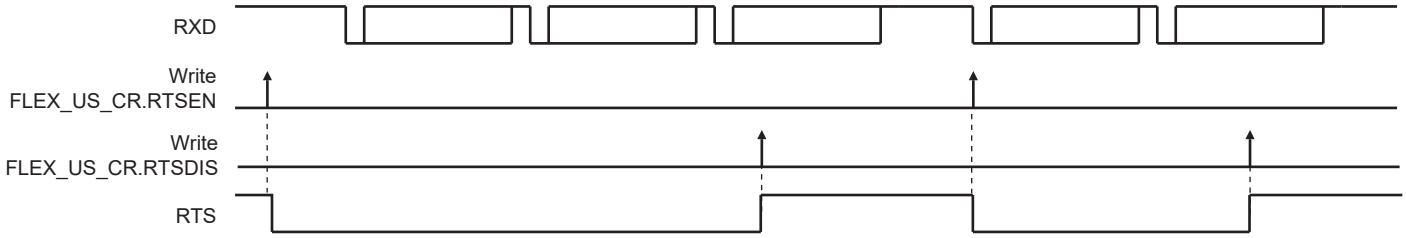
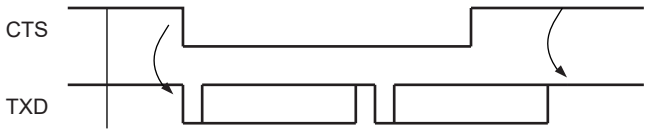


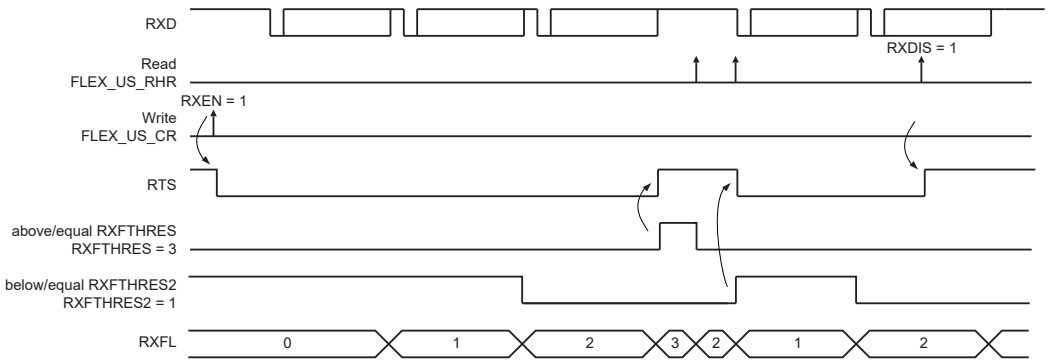

Figure 44-28 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processed, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

**Figure 44-28. Transmitter Behavior when Operating with Hardware Handshaking**




If USART FIFOs are enabled (FIFOEN bit in FLEX_US_CR), the RTS pin can be controlled by the USART Receive FIFO thresholds. The RTS pin control through Receive FIFO thresholds can be activated with the FRTSC bit in FLEX_US_FMR. Once activated, the RTS pin will be controlled by Receive FIFO thresholds, set to level ‘1’ each time RXFTHRES is reached and set to level ‘0’ each time RXFTHRES2 is reached (and RXFTHRES is not reached).

**Figure 44-29. Receiver Behavior When FIFO Enabled and FRTSC Set to ‘1’**




Note: In this mode, RXFTHRES must be > RXFTHRES2.

### 44.7.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

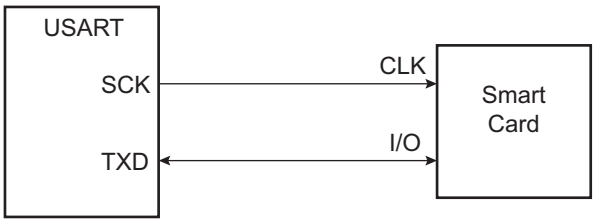
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in FLEX_US_MR to the value 0x4 for protocol T = 0 and to the value 0x6 for protocol T = 1.

### 44.7.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see Figure 44.7.1.1).

The USART connects to a smart card as shown in Figure 44-30. The TXD line becomes bidirectional and the baud rate generator feeds the ISO7816 clock on the SCK pin. As the TXD pin becomes bidirectional, its output remains driven by the output of the transmitter but only when the transmitter is active while its input is directed to the input of the receiver. The USART is considered as the master of the communication as it generates the clock.

**Figure 44-30. Connection of a Smart Card to the USART**




When operating in ISO7816, either in T = 0 or T = 1 modes, the character format is fixed. The configuration is 8 data bits, even parity and 1 or 2 stop bits, regardless of the values programmed in the CHRL, MODE9, PAR and CHMODE fields. MSBF can be used to transmit LSB or MSB first. Parity Bit (PAR) can be used to transmit in Normal or Inverse mode. Refer to Section 44.10.6 “USART Mode Register” and PAR: Parity Type.

The USART cannot operate concurrently in both Receiver and Transmitter modes as the communication is unidirectional at a time. It has to be configured according to the required mode by enabling or disabling either the receiver or the transmitter as desired. Enabling both the receiver and the transmitter at the same time in ISO7816 mode may lead to unpredictable results.

The ISO7816 specification defines an inverse transmission format. Data bits of the character must be transmitted on the I/O line at their negative value.

### 44.7.4.2 Protocol T = 0

In T = 0 protocol, a character is made up of 1 start bit, 8 data bits, 1 parity bit and 1 guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

If no parity error is detected, the I/O line remains at 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in Figure 44-31.

If a parity error is detected by the receiver, it drives the I/O line to 0 during the guard time, as shown in Figure 44-32. This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time which lasts 1 bit time.

When the USART is the receiver and it detects an error, it does not load the erroneous character in the Receive Holding Register (FLEX_US_RHR). It appropriately sets the PARE bit in the Status Register (FLEX_US_SR) so that the software can handle the error.

**Figure 44-31. T = 0 Protocol without Parity Error**

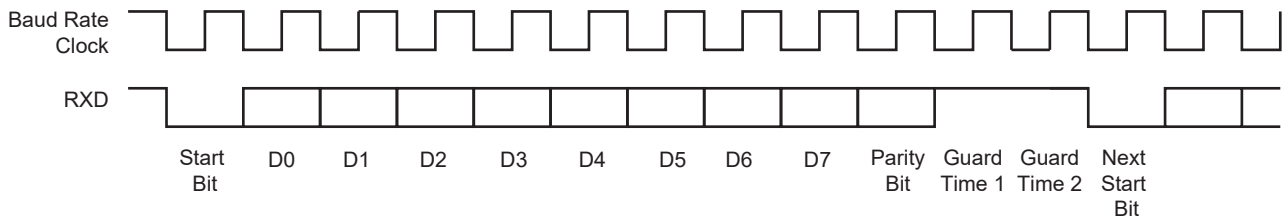
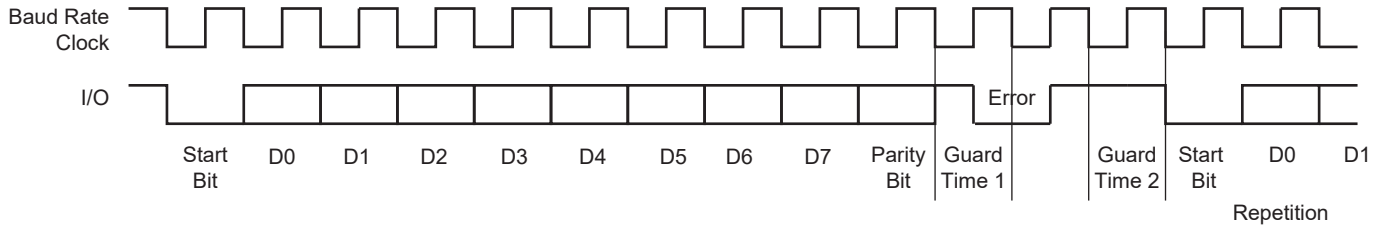


**Figure 44-32. T = 0 Protocol with Parity Error**




#### Receive Error Counter

The USART receiver also records the total number of errors. This can be read in the Number of Error (FLEX_US_NER) register. The NB_ERRORS field can record up to 255 errors. Reading FLEX_US_NER automatically clears the NB_ERRORS field.

#### Receive NACK Inhibit

The USART can be configured to inhibit an error. This is done by writing a '1' to FLEX_US_MR.INACK. In this case, no error signal is driven on the I/O line even if a parity bit is detected.

Moreover, if INACK = 1, the erroneous received character is stored in the Receive Holding register as if no error occurred, and the RXRDY bit rises.

#### Transmit Character Repetition

When the USART is transmitting a character and gets a NACK, it can automatically repeat the character before moving on to the next one. Repetition is enabled by writing the MAX_ITERATION field in FLEX_US_MR at a value higher than 0. Each character can be transmitted up to eight times; the first transmission plus seven repetitions.

If MAX_ITERATION does not equal zero, the USART repeats the character as many times as the value loaded in MAX_ITERATION.

When the USART repetition number reaches MAX_ITERATION, and the last repeated character is not acknowledged, the ITER bit is set in FLEX_US_CSR. If the repetition of the character is acknowledged by the receiver, the repetitions are stopped and the iteration counter is cleared.

The ITER bit in FLEX_US_CSR can be cleared by writing FLEX_US_CR with the RSTIT bit to 1.

#### Disable Successive Receive NACK

The receiver can limit the number of successive NACKs sent back to the remote transmitter. This is programmed by setting the DSNACK bit in FLEX_US_MR. The maximum number of NACKs transmitted is programmed in the MAX_ITERATION field. As soon as MAX_ITERATION is reached, no error signal is driven on the I/O line and the ITER bit in FLEX_US_CSR is set.

### 44.7.4.3 Protocol T = 1

When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in FLEX_US_CSR.

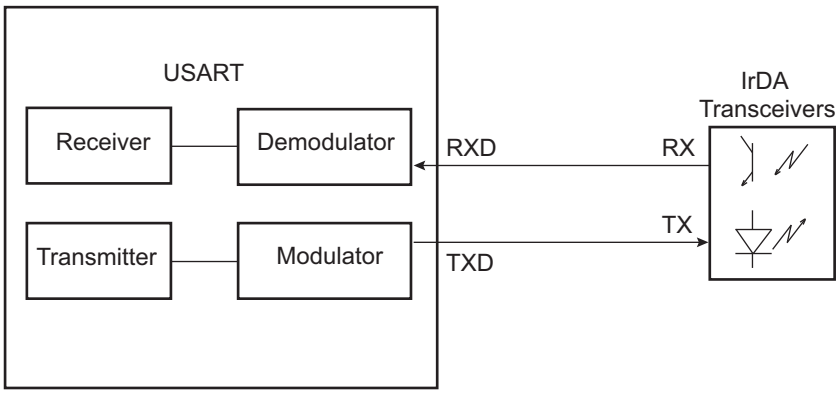
## 44.7.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in Figure 44-33. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 kbit/s to 115.2 kbit/s.

The USART IrDA mode is enabled by setting the USART_MODE field in FLEX_US_MR to the value 0x8. The IrDA Filter Register (FLEX_US_IF) allows configuring the demodulator filter. The USART transmitter and receiver

operate in a normal Asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

**Figure 44-33. Connection to IrDA Transceivers**




The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

To receive IrDA signals, the following needs to be done:

- Disable TX and Enable RX
- Configure the TXD pin as PIO and set it as an output to 0 (to avoid LED transmission). Disable the internal pull-up (better for power consumption).
- Receive data

#### 44.7.5.1 IrDA Modulation

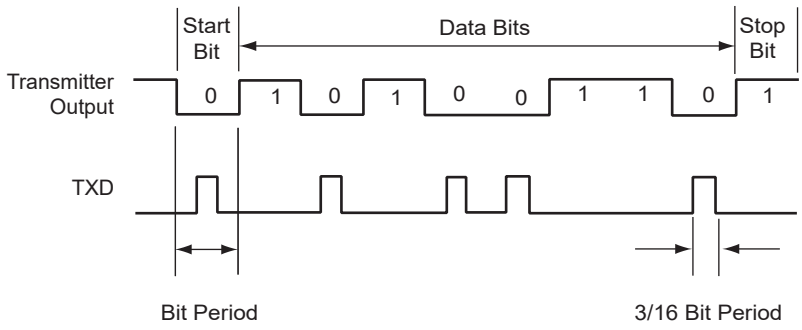
For baud rates up to and including 115.2 kbit/s, the RZI modulation scheme is used. “0” is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 44-12.

**Table 44-12. IrDA Pulse Duration**

| Baud Rate | Pulse Duration (3/16) |
|---|---|
| 2.4 kbit/s | 78.13 µs |
| 9.6 kbit/s | 19.53 µs |
| 19.2 kbit/s | 9.77 µs |
| 38.4 kbit/s | 4.88 µs |
| 57.6 kbit/s | 3.26 µs |
| 115.2 kbit/s | 1.63 µs |

Figure 44-34 shows an example of character transmission.

**Figure 44-34. IrDA Modulation**




### 44.7.5.2 IrDA Baud Rate

Table 44-13 gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of ±1.87% must be met.

**Table 44-13. IrDA Baud Rate Error**

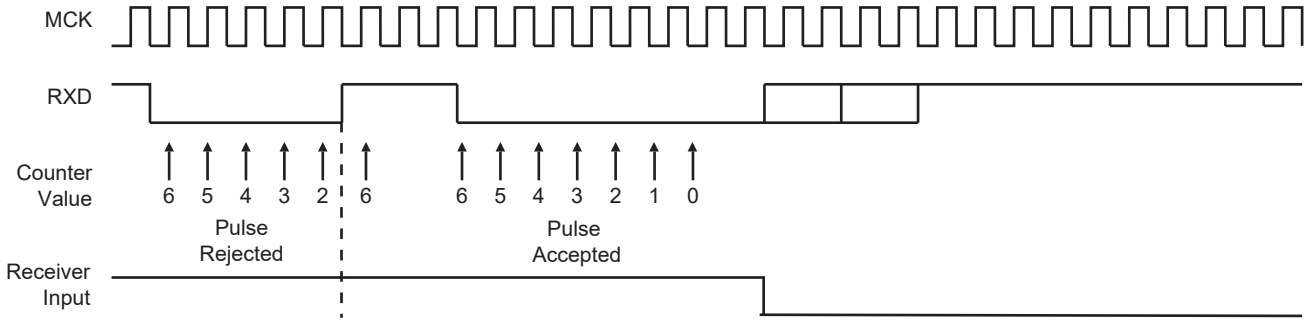
| Peripheral Clock | Baud Rate (bit/s) | CD | Baud Rate Error | Pulse Time (µs) |
|---|---|---|---|---|
| 3,686,400 | 115,200 | 2 | 0.00% | 1.63 |
| 20,000,000 | 115,200 | 11 | 1.38% | 1.63 |
| 32,768,000 | 115,200 | 18 | 1.25% | 1.63 |
| 40,000,000 | 115,200 | 22 | 1.38% | 1.63 |
| 3,686,400 | 57,600 | 4 | 0.00% | 3.26 |
| 20,000,000 | 57,600 | 22 | 1.38% | 3.26 |
| 32,768,000 | 57,600 | 36 | 1.25% | 3.26 |
| 40,000,000 | 57,600 | 43 | 0.93% | 3.26 |
| 3,686,400 | 38,400 | 6 | 0.00% | 4.88 |
| 20,000,000 | 38,400 | 33 | 1.38% | 4.88 |
| 32,768,000 | 38,400 | 53 | 0.63% | 4.88 |
| 40,000,000 | 38,400 | 65 | 0.16% | 4.88 |
| 3,686,400 | 19,200 | 12 | 0.00% | 9.77 |
| 20,000,000 | 19,200 | 65 | 0.16% | 9.77 |
| 32,768,000 | 19,200 | 107 | 0.31% | 9.77 |
| 40,000,000 | 19,200 | 130 | 0.16% | 9.77 |
| 3,686,400 | 9,600 | 24 | 0.00% | 19.53 |
| 20,000,000 | 9,600 | 130 | 0.16% | 19.53 |
| 32,768,000 | 9,600 | 213 | 0.16% | 19.53 |
| 40,000,000 | 9,600 | 260 | 0.16% | 19.53 |
| 3,686,400 | 2,400 | 96 | 0.00% | 78.13 |
| 20,000,000 | 2,400 | 521 | 0.03% | 78.13 |
| 32,768,000 | 2,400 | 853 | 0.04% | 78.13 |

#### 44.7.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in FLEX_US_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the peripheral clock speed. If a rising edge is detected on the RXD pin, the counter stops and is reloaded with FLEX_US_IF. If no rising edge is detected when the counter reaches 0, the input of the receiver is driven low during one bit time.

Figure 44-35 illustrates the operations of the IrDA demodulator.

**Figure 44-35. IrDA Demodulator Operations**




The programmed value in the FLEX_US_IF register must always meet the following criteria:

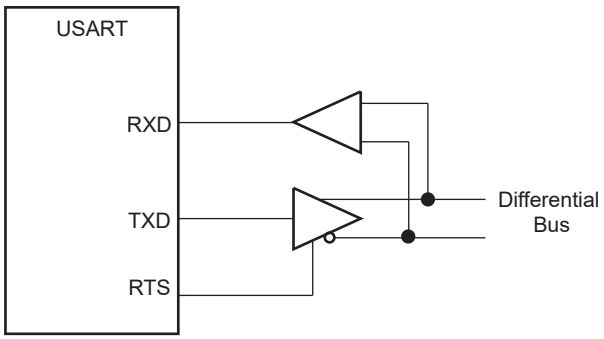
$$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$$

As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in FLEX_US_FIDI must be set to a value higher than 0 to make sure IrDA communications operate correctly.

### 44.7.6 RS485 Mode

The USART features the RS485 mode to enable line driver control. While operating in RS485 mode, the USART behaves as though in Asynchronous or Synchronous mode and configuration of all the parameters is possible. The difference is that the RTS pin is driven high when the transmitter is operating. The behavior of the RTS pin is controlled by the TXEMPTY bit. A typical connection of the USART to an RS485 bus is shown in Figure 44-36.

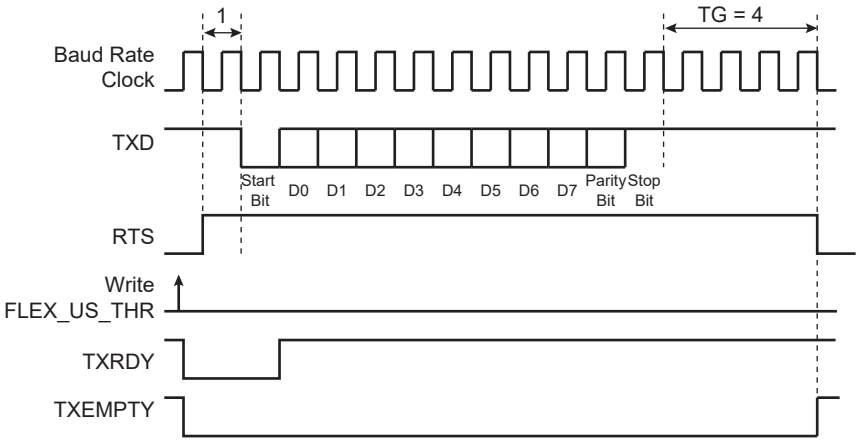
**Figure 44-36. Typical Connection to an RS485 Bus**




The USART is set in RS485 mode by writing the value 0x1 to the USART_MODE field in FLEX_US_MR.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. Figure 44-37 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

**Figure 44-37. Example of RTS Drive with Timeguard**




### 44.7.7 USART Comparison Function on Received Character

The CMP flag in FLEX_US_CSR is set when the received character matches the conditions programmed in FLEX_US_CMPR. The CMP flag is set as soon as FLEX_US_RHR is loaded with the new received character. The CMP flag is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

FLEX_US_CMPR (see Section 44.10.34 “USART Comparison Register”) can be programmed to provide different comparison methods:

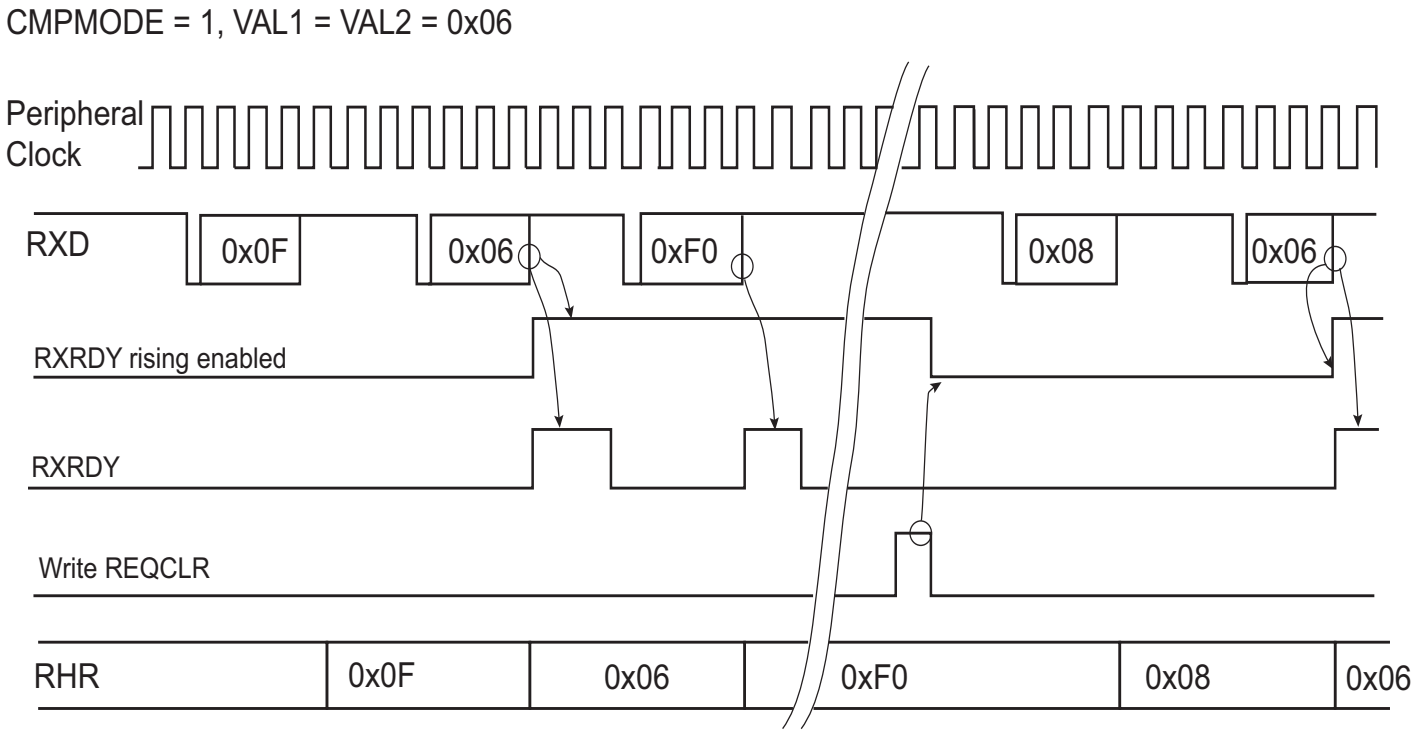
- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to 1 if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the flag CMP is set to 1 if any received character equals VAL1 or VAL2.

When the CMPMODE bit is set to FLAG_ONLY (value 0) in FLEX_US_CMPR, all received data are loaded in FLEX_US_RHR and the CMP flag provides the status of the comparison result.

By programming the CMPMODE bit to START_CONDITION (value 1), the comparison function result triggers the start of the loading of FLEX_US_RHR (see Figure 44-38). The trigger condition exists as soon as the received character value matches the condition defined by the programming of VAL1, VAL2 and CMPPAR in FLEX_US_CMPR. The comparison trigger event is restarted by writing a 1 to the REQCLR bit in FLEX_US_CR.

The value programmed in the VAL1 and VAL2 fields must not exceed the maximum value of the received character (see CHRL field in FLEX_US_MR).

**Figure 44-38. Receive Holding Register Management**




### 44.7.8 SPI Mode

The Serial Peripheral Interface (SPI) mode is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master” which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turns being masters and one master may simultaneously shift data into multiple slaves. (Multiple master protocol is the opposite of single master protocol, where one CPU is always the master while all of the others are always slaves.) However, only one slave may drive its output to write data back to the master at any given time.

A slave device is selected when its NSS signal is asserted by the master. The USART in SPI Master mode can address only one SPI slave because it can generate only one NSS signal.

The SPI system consists of two data lines and two control lines:

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input of the slave.
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master.
- Serial Clock (SCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of bit rates. The SCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows the master to select or deselect the slave.

#### 44.7.8.1 Modes of Operation

The USART can operate in SPI Master mode or in SPI Slave mode.

Operation in SPI Master mode is programmed by writing 0xE to the USART_MODE field in FLEX_US_MR. In this case the SPI lines must be connected as described below:

- The MOSI line is driven by the output pin TXD
- The MISO line drives the input pin RXD
- The SCK line is driven by the output pin SCK
- The NSS line is driven by the output pin RTS

Operation in SPI Slave mode is programmed by writing to 0xF the USART_MODE field in FLEX_US_MR. In this case the SPI lines must be connected as described below:

- The MOSI line drives the input pin RXD
- The MISO line is driven by the output pin TXD
- The SCK line drives the input pin SCK
- The NSS line drives the input pin CTS

In order to avoid unpredictable behavior, any change of the SPI mode must be followed by a software reset of the transmitter and of the receiver (except the initial configuration after a hardware reset). (See Section 44.7.8.4 “Receiver and Transmitter Control”.)

#### 44.7.8.2 Bit Rate

In SPI mode, the bit rate generator operates in the same way as in USART Synchronous mode: Section 44.7.1.3 “Baud Rate in Synchronous Mode or SPI Mode” However, there are some restrictions:

In SPI Master mode:

- The external clock SCK must not be selected (USCLKS ≠ 0x3), and the CLKO bit in FLEX_US_MR must be set in order to generate correctly the serial clock on the SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the value programmed in CD must be ≥ 6.
- If the divided peripheral clock is selected, the value programmed in CD must be even to ensure a 50:50 mark/space ratio on the SCK pin, this value can be odd if the peripheral clock is selected.

In SPI Slave mode:

- The external clock (SCK) selection is forced regardless of the value of the USCLKS field in FLEX_US_MR. Likewise, the value written in FLEX_US_BRGR has no effect, because the clock is provided directly by the signal on the USART SCK pin.
- To obtain correct behavior of the receiver and the transmitter, the external clock (SCK) frequency must be at least six times lower than the system clock.

#### 44.7.8.3 Data Transfer

Up to nine data bits are successively shifted out on the TXD pin at each rising or falling edge (depending of CPOL and CPHA) of the programmed serial clock. There is no Start bit, no Parity bit and no Stop bit.

The number of data bits is selected by the CHRL field and the MODE 9 bit in FLEX_US_MR. The nine bits are selected by setting the MODE 9 bit regardless of the CHRL field. The MSB data bit is always sent first in SPI mode (Master or Slave).

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in FLEX_US_MR. The clock phase is programmed with the CPHA bit. These two parameters determine the edges of the clock signal upon which data are driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

**Table 44-14. SPI Bus Protocol Mode**

| SPI Bus Protocol Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |

**Figure 44-39. SPI Transfer Format (CPHA = 1, 8 bits per transfer)**

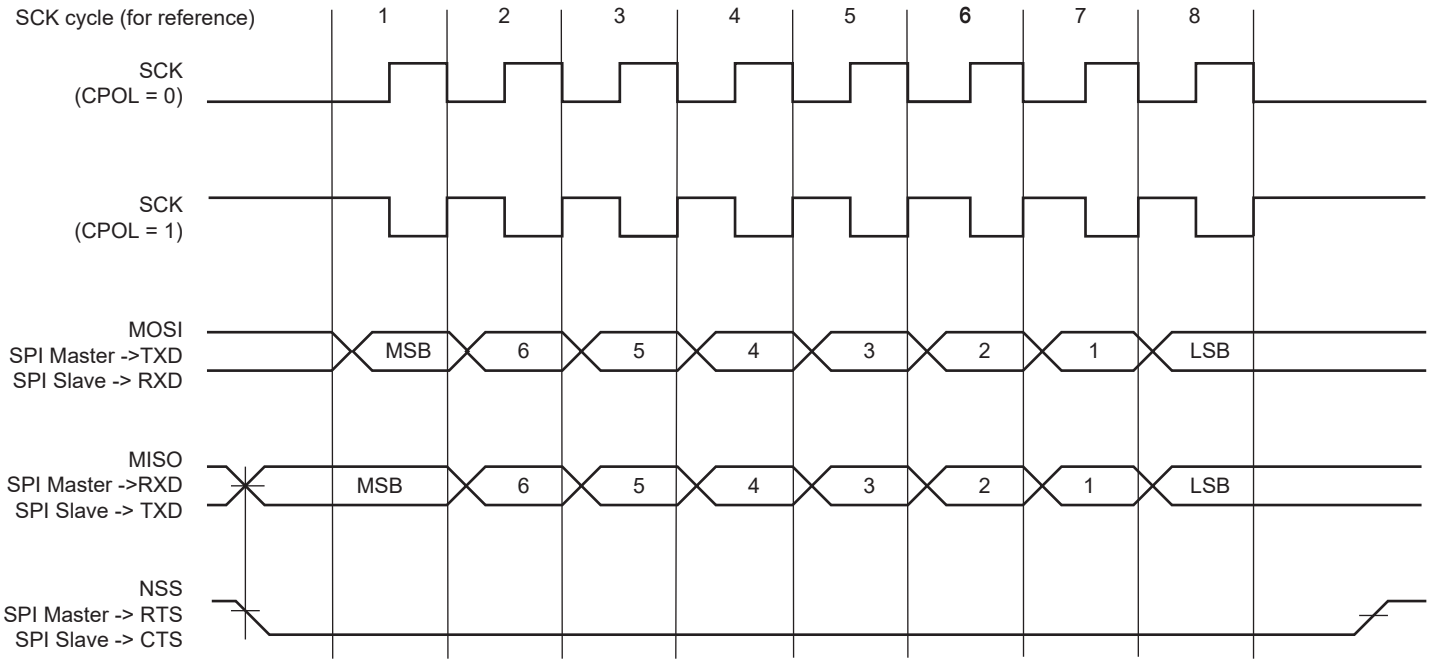
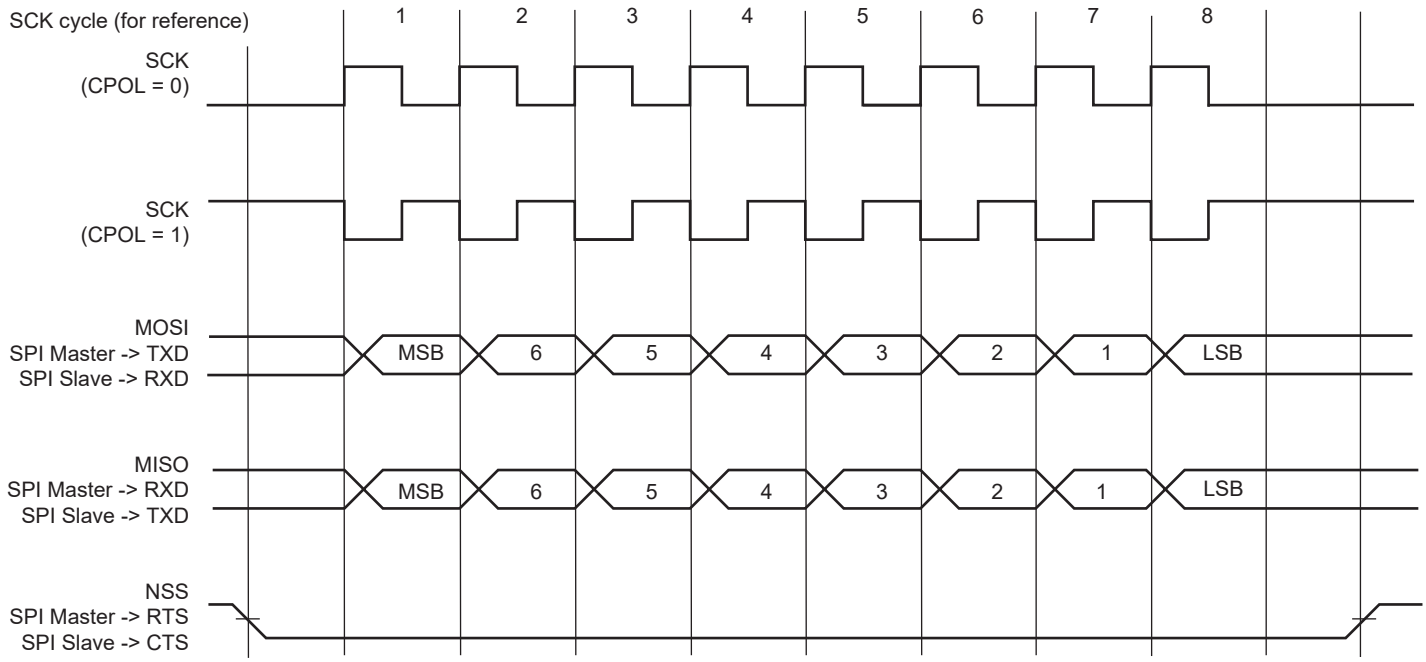


**Figure 44-40. SPI Transfer Format (CPHA = 0, 8 bits per transfer)**




#### 44.7.8.4 Receiver and Transmitter Control

See Section 44.7.2 “Receiver and Transmitter Control”.

#### 44.7.8.5 Character Transmission

The characters are sent by writing in the Transmit Holding Register (FLEX_US_THR). An additional condition for transmitting a character can be added when the USART is configured in SPI Master mode. In the “USART Mode Register (SPI_MODE)” (USART_MR), the value configured on the WRDBT bit can prevent any character transmission (even if FLEX_US_THR has been written) while the receiver side is not ready (character not read). When WRDBT = 0, the character is transmitted whatever the receiver status. If WRDBT = 1, the transmitter waits for the Receive Holding Register (FLEX_US_RHR) to be read before transmitting the character (RXRDY flag cleared), thus preventing any overflow (character loss) on the receiver side.

The chip select line is de-asserted for a period equivalent to 3 bits between the transmission of two data.

The transmitter reports two status bits in FLEX_US_CSR: TXRDY (Transmitter Ready), which indicates that FLEX_US_THR is empty and TXEMPTY, which indicates that all the characters written in FLEX_US_THR have been processed. When the current character processing is completed, the last character written in FLEX_US_THR is transferred into the Shift register of the transmitter and FLEX_US_THR becomes empty, thus TXRDY rises.

Both TXRDY and TXEMPTY bits are low when the transmitter is disabled. Writing a character in FLEX_US_THR while TXRDY is low has no effect and the written character is lost.

If the USART is in SPI Slave mode and if a character must be sent while FLEX_US_THR is empty, the UNRE (Underrun Error) bit is set. The TXD transmission line stays at high level during all this time. The UNRE bit is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

In SPI Master mode, the slave select line (NSS) is asserted at low level one $t_{bit}$ ($t_{bit}$ being the

nominal time required to transmit a bit) before the transmission of the MSB bit and released at high level one $t_{bit}$ after the transmission of the LSB bit. So, the slave select line (NSS) is always released between each character transmission and a minimum delay of three $t_{bit}$ always inserted. However, in order to address slave devices supporting the CSAAT mode (Chip Select Active After Transfer), the slave select line (NSS) can be forced at low level by writing a one to the RTSEN bit in FLEX_US_CR. The slave select line (NSS) can be released at high level

only by writing a one to the RTSDIS bit in FLEX_US_CR (for example, when all data have been transferred to the slave device).

In SPI Slave mode, the transmitter does not require a falling edge of the slave select line (NSS) to initiate a character transmission but only a low level. However, this low level must be present on the slave select line (NSS) at least one $t_{bit}$ before the first serial clock cycle corresponding to the MSB bit.

#### 44.7.8.6 Character Reception

When a character reception is completed, it is transferred to the Receive Holding Register (FLEX_US_RHR) and the RXRDY bit in the Status Register (FLEX_US_CSR) rises. If a character is completed while RXRDY is set, the OVRE (Overrun Error) bit is set. The last character is transferred into FLEX_US_RHR and overwrites the previous one. The OVRE bit is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

To ensure correct behavior of the receiver in SPI Slave mode, the master device sending the frame must ensure a minimum delay of one $t_{bit}$ between each character transmission. The receiver does not require a falling edge of the slave select line (NSS) to initiate a character reception but only a low level. However, this low level must be present on the slave select line (NSS) at least one $t_{bit}$ before the first serial clock cycle corresponding to the MSB bit.

#### 44.7.8.7 Receiver Timeout

Because the receiver bit rate clock is active only during data transfers in SPI mode, a receiver timeout is impossible in this mode, whatever the timeout value is (field TO) in FLEX_US_RTOR.

### 44.7.9 LIN Mode

The LIN mode provides master node and slave node connectivity on a LIN bus.

The LIN (Local Interconnect Network) is a serial communication protocol which efficiently supports the control of mechatronic nodes in distributed automotive applications.

The main properties of the LIN bus are:

- Single master/multiple slaves concept
- Low-cost silicon implementation based on common UART/SCI interface hardware, an equivalent in software, or as a pure state machine.
- Self synchronization without quartz or ceramic resonator in the slave nodes
- Deterministic signal transmission
- Low cost single-wire implementation
- Speed up to 20 kbit/s

LIN provides cost efficient bus communication where the bandwidth and versatility of CAN are not required.

The LIN mode enables processing LIN frames with a minimum of action from the microprocessor.

#### 44.7.9.1 Modes of Operation

The USART can act either as a LIN master node or as a LIN slave node.

The node configuration is chosen by setting the USART_MODE field in the USART Mode Register (FLEX_US_MR):

- LIN master node (USART_MODE = 0xA)
- LIN slave node (USART_MODE = 0xB)

In order to avoid unpredictable behavior, any change of the LIN node configuration must be followed by a software reset of the transmitter and of the receiver (except the initial node configuration after a hardware reset). (See Section 44.7.9.3 “Receiver and Transmitter Control”.)

#### 44.7.9.2 Baud Rate Configuration

See Section 44.7.1.1 “Baud Rate in Asynchronous Mode”.

- LIN master node: The baud rate is configured in FLEX_US_BRGR.
- LIN slave node: The initial baud rate is configured in FLEX_US_BRGR. This configuration is automatically copied in the LIN Baud Rate Register (FLEX_US_LINBRR) when writing FLEX_US_BRGR. After the synchronization procedure, the baud rate is updated in FLEX_US_LINBRR.

#### 44.7.9.3 Receiver and Transmitter Control

See Section 44.7.2 “Receiver and Transmitter Control”.

#### 44.7.9.4 Character Transmission

See Section 44.7.3.1 “Transmitter Operations”.

#### 44.7.9.5 Character Reception

See Section 44.7.3.7 “Receiver Operations”.

#### 44.7.9.6 Header Transmission (Master Node Configuration)

All the LIN Frames start with a header which is sent by the master node and consists of a Synch Break Field, Synch Field and Identifier Field.

So in master node configuration, the frame handling starts with the sending of the header.

The header is transmitted as soon as the identifier is written in the LIN Identifier Register (FLEX_US_LINIR). At this moment the flag TXRDY falls.

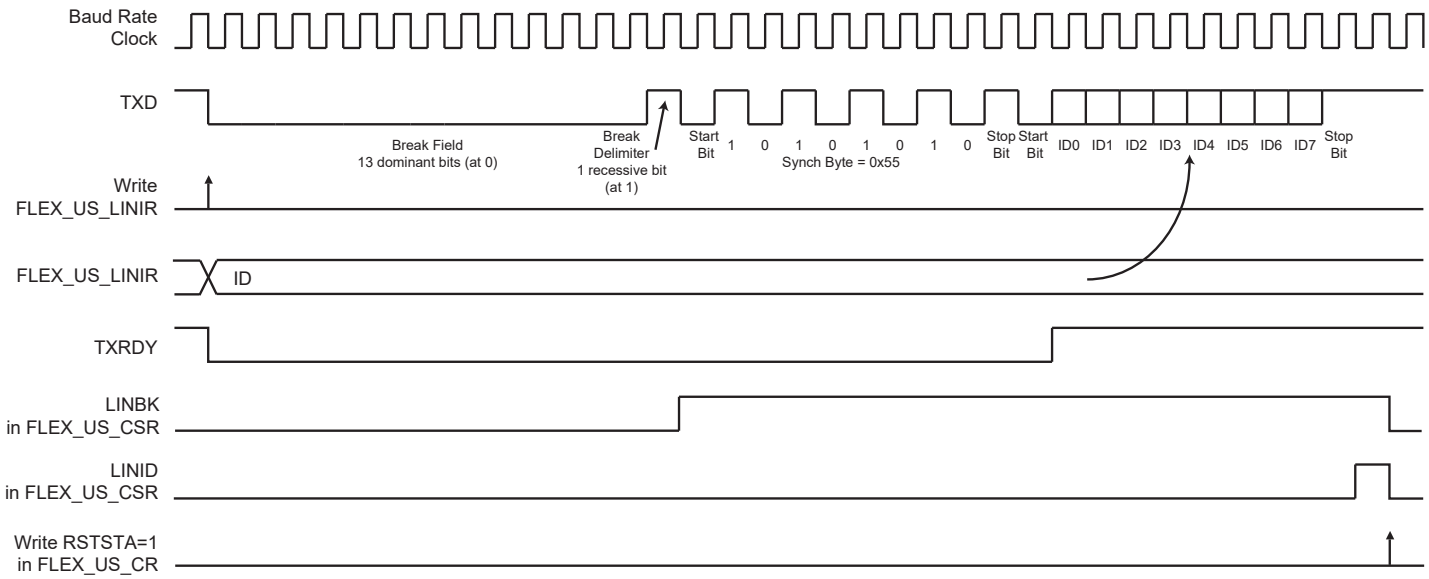
The Break Field, the Synch Field and the Identifier Field are sent automatically one after the other.

The Break Field consists of 13 dominant bits and 1 recessive bit, the Synch Field is the character 0x55 and the Identifier corresponds to the character written in the LIN Identifier Register (FLEX_US_LINIR). The Identifier parity bits can be automatically computed and sent (see Section 44.7.9.9).

The flag TXRDY rises when the identifier character is transferred into the Shift register of the transmitter.

As soon as the Synch Break Field is transmitted, the flag bit LINBK in FLEX_US_CSR is set. Likewise, as soon as the Identifier Field is sent, the flag bit LINID in FLEX_US_CSR is set. These flags are reset by writing a one to the RSTSTA bit in FLEX_US_CR.

**Figure 44-41. Header Transmission**



### 44.7.9.7 Header Reception (Slave Node Configuration)

All the LIN frames start with a header which is sent by the master node and consists of a Synch Break Field, Synch Field and Identifier Field.

In slave node configuration, the frame handling starts with the reception of the header.

The USART uses a break detection threshold of 11 nominal bit times at the actual baud rate. At any time, if 11 consecutive recessive bits are detected on the bus, the USART detects a Break Field. As long as a Break Field has not been detected, the USART stays idle and the received data are not taken in account.

When a Break Field has been detected, the flag LINBK in FLEX_US_CSR is set and the USART expects the Synch Field character to be 0x55. This field is used to update the actual baud rate in order to stay synchronized (see Section 44.7.9.8). If the received Synch character is not 0x55, an Inconsistent Synch Field error is generated (see Section 44.7.9.14).

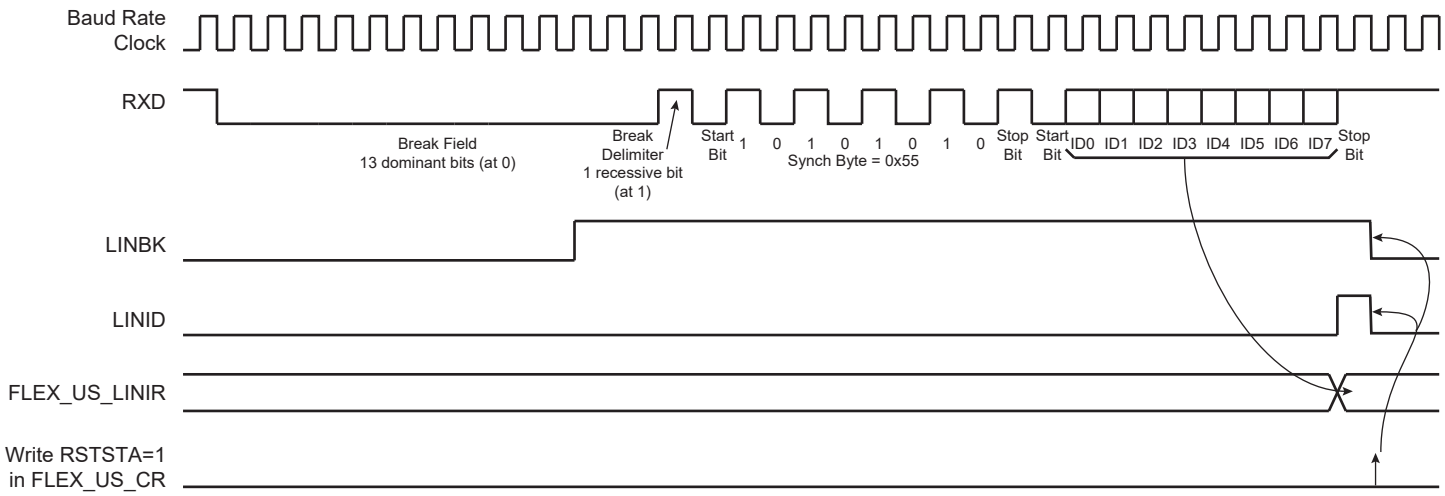
After receiving the Synch Field, the USART expects to receive the Identifier Field.

When the Identifier Field has been received, the flag bit LINID in FLEX_US_CSR is set. At this moment the IDCHR field in the LIN Identifier Register (FLEX_US_LINIR) is updated with the received character. The Identifier parity bits can be automatically computed and checked (see Section 44.7.9.9).

If the Header is not entirely received within the time given by the maximum length of the header $t_{Header\_Maximum}$, the error flag bit LINHTE in FLEX_US_CSR is set.

The flag bits LINID, LINBK and LINHTE are reset by writing a one to the RSTSTA bit in FLEX_US_CR.

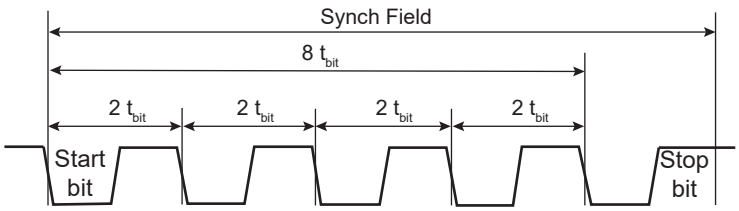
**Figure 44-42. Header Reception**




### 44.7.9.8 Slave Node Synchronization

The synchronization is done only in slave node configuration. The procedure is based on time measurement between falling edges of the Synch Field. The falling edges are available in distances of 2, 4, 6 and 8 bit times.

**Figure 44-43. Synch Field**




The time measurement is made by a 19-bit counter driven by the sampling clock (see Section 44.7.1).

When the start bit of the Synch Field is detected, the counter is reset. Then during the next eight $t_{bit}$ of the Synch Field, the counter is incremented. At the end of these eight $t_{bit}$, the counter is stopped. At this moment, the 16 most significant bits of the counter (value divided by 8) give the new clock divider (LINCD) and the 3 least significant bits of this value (the remainder) give the new fractional part (LINFP).

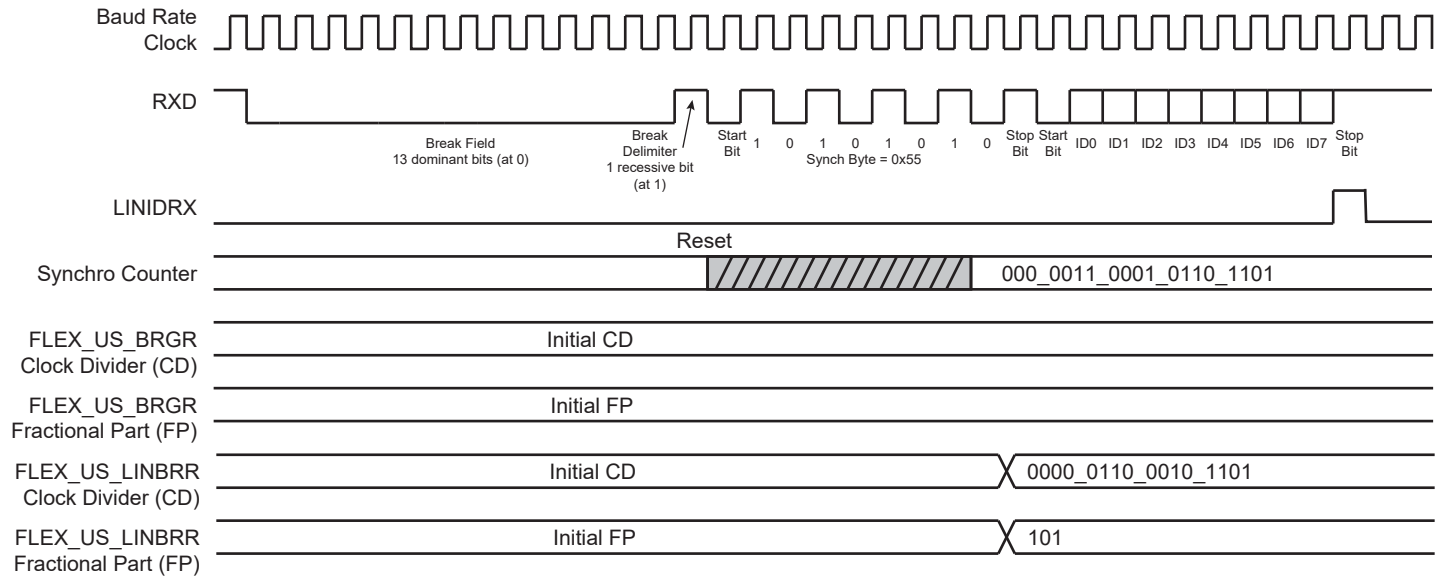
Once the Synch Field has been entirely received, the clock divider (LINCD) and the fractional part (LINFP) are updated in the LIN Baud Rate Register (FLEX_US_LINBRR) with the computed values, if the Synchronization is not disabled by the SYNCDIS bit in the LIN Mode Register (FLEX_US_LINMR).

After reception of the Synch Field:

- If it appears that the computed baud rate deviation compared to the initial baud rate is superior to the maximum tolerance FTol_Unsynch (±15%), then the clock divider (LINCD) and the fractional part (LINFP) are not updated, and the error flag bit LINSTE in FLEX_US_CSR is set.
- If it appears that the sampled Synch character is not equal to 0x55, then the clock divider (LINCD) and the fractional part (LINFP) are not updated, and the error flag bit LINISFE in FLEX_US_CSR is set.

Flags LINSTE and LINISFE are reset by writing a one to the RSTSTA bit in FLEX_US_CR.

**Figure 44-44. Slave Node Synchronization**




The accuracy of the synchronization depends on several parameters:

- The nominal clock frequency ($f_{Nom}$) (the theoretical slave node clock frequency)
- The Baud Rate
- The oversampling (OVER = 0 => 16X or OVER = 1 => 8X)

The following formula is used to compute the deviation of the slave bit rate relative to the master bit rate after synchronization ($f_{SLAVE}$ is the real slave node clock frequency).

$$\text{Baud rate deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baud rate}}{8 \times f_{SLAVE}}\right)\%$$

$$\text{Baud rate deviation} = \left(100 \times \frac{[\alpha \times 8 \times (2 - \text{Over}) + \beta] \times \text{Baud rate}}{8 \times \left(\frac{f_{TOL\_UNSYNCH}}{100}\right) \times f_{Nom}}\right)\%$$

$$-0.5 \le \alpha \le +0.5 \quad -1 < \beta < +1$$

$f_{TOL\_UNSYNCH}$ is the deviation of the real slave node clock from the nominal clock frequency. The LIN Standard imposes that it must not exceed ±15%. The LIN Standard imposes also that for communication between two nodes, their bit rate must not differ by more than ±2%. This means that the baud rate deviation must not exceed ±1%.

It follows from that, a minimum value for the nominal clock frequency:

$$f_{Nom}(\text{min}) = \left(100 \times \frac{[0.5 \times 8 \times (2 - \text{Over}) + 1] \times \text{Baud rate}}{8 \times \left(\frac{-15}{100} + 1\right) \times 1\%}\right)\text{Hz}$$

Examples:

- Baud rate = 20 kbit/s, OVER = 0 (Oversampling 16X) => $f_{Nom}$(min) = 2.64 MHz
- Baud rate = 20 kbit/s, OVER = 1 (Oversampling 8X) => $f_{Nom}$(min) = 1.47 MHz
- Baud rate = 1 kbit/s, OVER = 0 (Oversampling 16X) => $f_{Nom}$(min) = 132 kHz
- Baud rate = 1 kbit/s, OVER = 1 (Oversampling 8X) => $f_{Nom}$(min) = 74 kHz

#### 44.7.9.9 Identifier Parity

A protected identifier consists of two subfields; the identifier and the identifier parity. Bits 0 to 5 are assigned to the identifier and bits 6 and 7 are assigned to the parity.

The USART interface can generate/check these parity bits, but this feature can also be disabled. The user can choose between two modes by the PARDIS bit of FLEX_US_LINMR:

- PARDIS = 0:
  - During header transmission, the parity bits are computed and sent with the six least significant bits of the IDCHR field of the LIN Identifier Register (FLEX_US_LINIR). The bits 6 and 7 of this register are discarded.
  - During header reception, the parity bits of the identifier are checked. If the parity bits are wrong, an Identifier Parity error occurs (see Section 44.7.3.8). Only the six least significant bits of the IDCHR field are updated with the received Identifier. The bits 6 and 7 are stuck to 0.
- PARDIS = 1:
  - During header transmission, all the bits of the IDCHR field of the LIN Identifier Register (FLEX_US_LINIR) are sent on the bus.
  - During header reception, all the bits of the IDCHR field are updated with the received Identifier.

#### 44.7.9.10 Node Action

Depending on the identifier, the node is affected—or not—by the LIN response. Consequently, after sending or receiving the identifier, the USART must be configured. There are three possible configurations:

- PUBLISH: the node sends the response.
- SUBSCRIBE: the node receives the response.
- IGNORE: the node is not concerned by the response, it does not send and does not receive the response.

This configuration is made by the LIN Node Action (NACT) field in FLEX_US_LINMR (see Section 44.10.31).

Example: a LIN cluster that contains a master and two slaves:

- Data transfer from the master to slave 1 and to slave 2:

  NACT(master) = PUBLISH

  NACT(slave 1) = SUBSCRIBE

  NACT(slave 2) = SUBSCRIBE
- Data transfer from the master to slave 1 only:

  NACT(master) = PUBLISH

  NACT(slave 1) = SUBSCRIBE

  NACT(slave 2) = IGNORE
- Data transfer from slave 1 to the master:

  NACT(master) = SUBSCRIBE

  NACT(slave 1) = PUBLISH

  NACT(slave 2) = IGNORE
- Data transfer from slave 1 to slave 2:

  NACT(master) = IGNORE

  NACT(slave 1) = PUBLISH

  NACT(slave 2) = SUBSCRIBE
- Data transfer from slave 2 to the master and to slave 1:

  NACT(master) = SUBSCRIBE

  NACT(slave 1) = SUBSCRIBE

  NACT(slave 2) = PUBLISH

#### 44.7.9.11 Response Data Length

The LIN response data length is the number of data fields (bytes) of the response excluding the checksum.

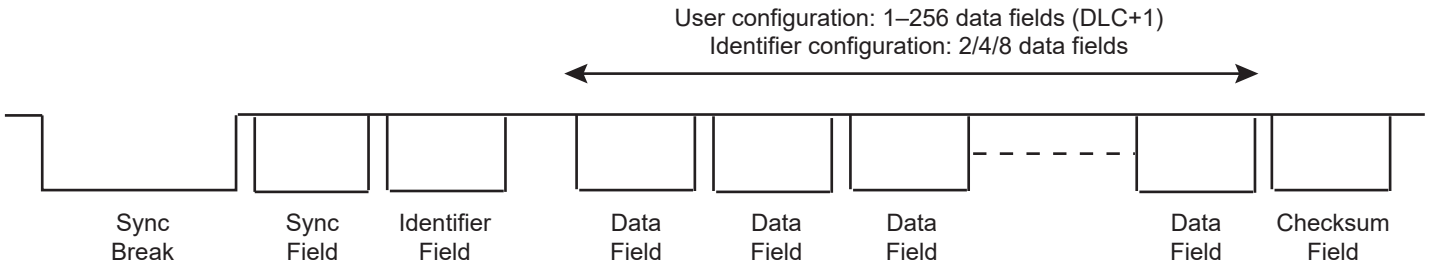
The response data length can either be configured by the user or be defined automatically by bits 4 and 5 of the Identifier (compatibility to LIN Specification 1.1). The user can choose between these two modes by the DLM bit of FLEX_US_LINMR:

- DLM = 0: The response data length is configured by the user via the DLC field of FLEX_US_LINMR. The response data length is equal to (DLC + 1) bytes. DLC can be programmed from 0 to 255, so the response can contain from 1 data byte up to 256 data bytes.
- DLM = 1: The response data length is defined by the Identifier (IDCHR in FLEX_US_LINIR) according to the table below. The DLC field of FLEX_US_LINMR is discarded. The response can contain 2 or 4 or 8 data bytes.

**Table 44-15. Response Data Length if DLM = 1**

| IDCHR[5] | IDCHR[4] | Response Data Length (bytes) |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 2 |
| 1 | 0 | 4 |
| 1 | 1 | 8 |

**Figure 44-45. Response Data Length**




### 44.7.9.12 Checksum

The last field of a frame is the checksum. The checksum contains the inverted 8- bit sum with carry, over all data bytes or all data bytes and the protected identifier. Checksum calculation over the data bytes only is called classic checksum and it is used for communication with LIN 1.3 slaves. Checksum calculation over the data bytes and the protected identifier byte is called enhanced checksum and it is used for communication with LIN 2.0 slaves.

The USART can be configured to:

- Send/Check an Enhanced checksum automatically (CHKDIS = 0 & CHKTYP = 0)
- Send/Check a Classic checksum automatically (CHKDIS = 0 & CHKTYP = 1)
- Not send/check a checksum (CHKDIS = 1)

This configuration is made by the Checksum Type (CHKTYP) and Checksum Disable (CHKDIS) bits of FLEX_US_LINMR.

If the checksum feature is disabled, the user can send it manually all the same, by considering the checksum as a normal data byte and by adding 1 to the response data length (see Section 44.7.9.11).

### 44.7.9.13 Frame Slot Mode

This mode is useful only for master nodes. It respects the following rule: each frame slot shall be longer than or equal to $t_{Frame\_Maximum}$.

If the Frame Slot mode is enabled (FSDIS = 0) and a frame transfer has been completed, the TXRDY flag is set again only after $t_{Frame\_Maximum}$ delay, from the start of frame. So the master node cannot send a new header if the frame slot duration of the previous frame is inferior to $t_{Frame\_Maximum}$.

If the Frame Slot mode is disabled (FSDIS = 1) and a frame transfer has been completed, the TXRDY flag is set again immediately.

The $t_{Frame\_Maximum}$ is calculated as follows:

If the Checksum is sent (CHKDIS = 0):

- $t_{Header\_Nominal} = 34 \times t_{bit}$
- $t_{Response\_Nominal} = 10 \times (NData + 1) \times t_{bit}$
- $t_{Frame\_Maximum} = 1.4 \times (t_{Header\_Nominal} + t_{Response\_Nominal} + 1)$ (1)
- $t_{Frame\_Maximum} = 1.4 \times (34 + 10 \times (DLC + 1 + 1) + 1) \times t_{bit}$

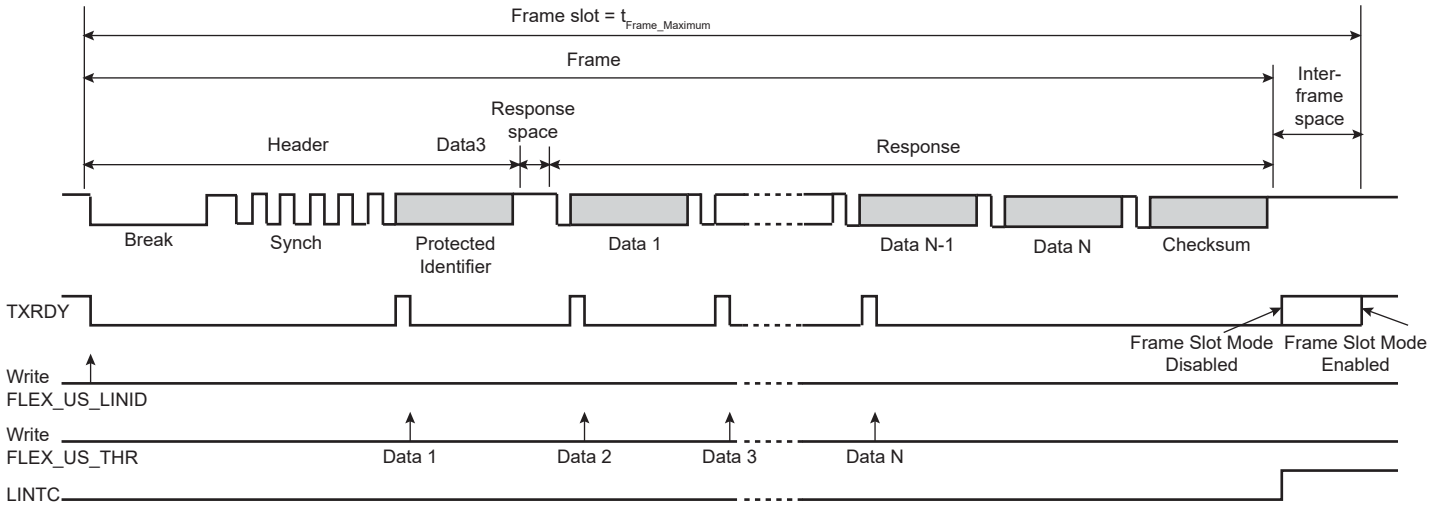
- $t_{Frame\_Maximum} = (77 + 14 \times DLC) \times t_{bit}$

If the Checksum is not sent (CHKDIS = 1):

- $t_{Header\_Nominal} = 34 \times t_{bit}$
- $t_{Response\_Nominal} = 10 \times NData \times t_{bit}$
- $t_{Frame\_Maximum} = 1.4 \times (t_{Header\_Nominal} + t_{Response\_Nominal} + 1)$(1)
- $t_{Frame\_Maximum} = 1.4 \times (34 + 10 \times (DLC + 1) + 1) \times t_{bit}$
- $t_{Frame\_Maximum} = (63 + 14 \times DLC) \times t_{bit}$

Note: 1. The term “+1” leads to an integer result for $t_{Frame\_Maximum}$ (LIN Specification 1.3).

**Figure 44-46. Frame Slot Mode**




### 44.7.9.14 LIN Errors

#### Bit Error

This error is generated in master of slave node configuration, when the USART is transmitting and if the transmitted value on the Tx line is different from the value sampled on the Rx line. If a bit error is detected, the transmission is aborted at the next byte border.

This error is reported by flag LINBE in FLEX_US_CSR.

#### Inconsistent Synch Field Error

This error is generated in slave node configuration, if the Synch Field character received is other than 0x55.

This error is reported by flag LINISFE in FLEX_US_CSR.

#### Identifier Parity Error

This error is generated in slave node configuration, if the parity of the identifier is wrong. This error can be generated only if the parity feature is enabled (PARDIS = 0).

This error is reported by flag LINIPE in FLEX_US_CSR.

#### Checksum Error

This error is generated in master of slave node configuration, if the received checksum is wrong. This flag can be set to 1 only if the checksum feature is enabled (CHKDIS = 0).

This error is reported by flag LINCE in FLEX_US_CSR.

#### Slave Not Responding Error

This error is generated in master of slave node configuration, when the USART expects a response from another node (NACT = SUBSCRIBE) but no valid message appears on the bus within the time given by the maximum length of the message frame, $t_{Frame\_Maximum}$ (see Section 44.7.9.13). This error is disabled if the USART does not expect any message (NACT = PUBLISH or NACT = IGNORE).

This error is reported by flag LINSNRE in FLEX_US_CSR.

#### Synch Tolerance Error

This error is generated in slave node configuration if, after the clock synchronization procedure, it appears that the computed baud rate deviation compared to the initial baud rate is superior to the maximum tolerance FTol_Unsynch (±15%).

This error is reported by flag LINSTE in FLEX_US_CSR.

#### Header Timeout Error

This error is generated in slave node configuration, if the Header is not entirely received within the time given by the maximum length of the Header, $t_{Header\_Maximum}$.

This error is reported by flag LINHTE in FLEX_US_CSR.

### 44.7.9.15 LIN Frame Handling

#### Master Node Configuration

- Write TXEN and RXEN in FLEX_US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in FLEX_US_MR to select the LIN mode and the master node configuration.
- Write CD and FP in FLEX_US_BRGR to configure the baud rate.
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM, FSDIS and DLC in FLEX_US_LINMR to configure the frame transfer.
- Check that TXRDY in FLEX_US_CSR is set to 1
- Write IDCHR in FLEX_US_LINIR to send the header

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the USART sends the response
  - Wait until TXRDY in FLEX_US_CSR rises
  - Write TCHR in FLEX_US_THR to send a byte
  - If all the data have not been written, redo the two previous steps
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors
- Case 2: NACT = SUBSCRIBE, the USART receives the response
  - Wait until RXRDY in FLEX_US_CSR rises
  - Read RCHR in FLEX_US_RHR
  - If all the data have not been read, redo the two previous steps
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors
- Case 3: NACT = IGNORE, the USART is not concerned by the response
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors

**Figure 44-47. Master Node Configuration, NACT = PUBLISH**

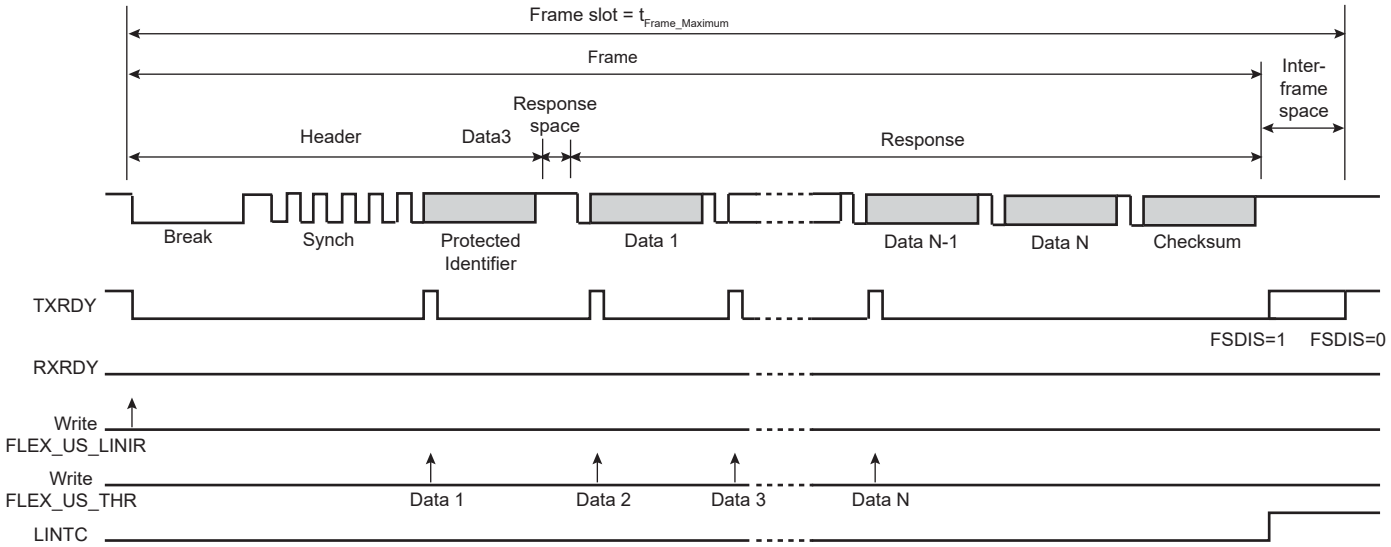



**Figure 44-48. Master Node Configuration, NACT = SUBSCRIBE**

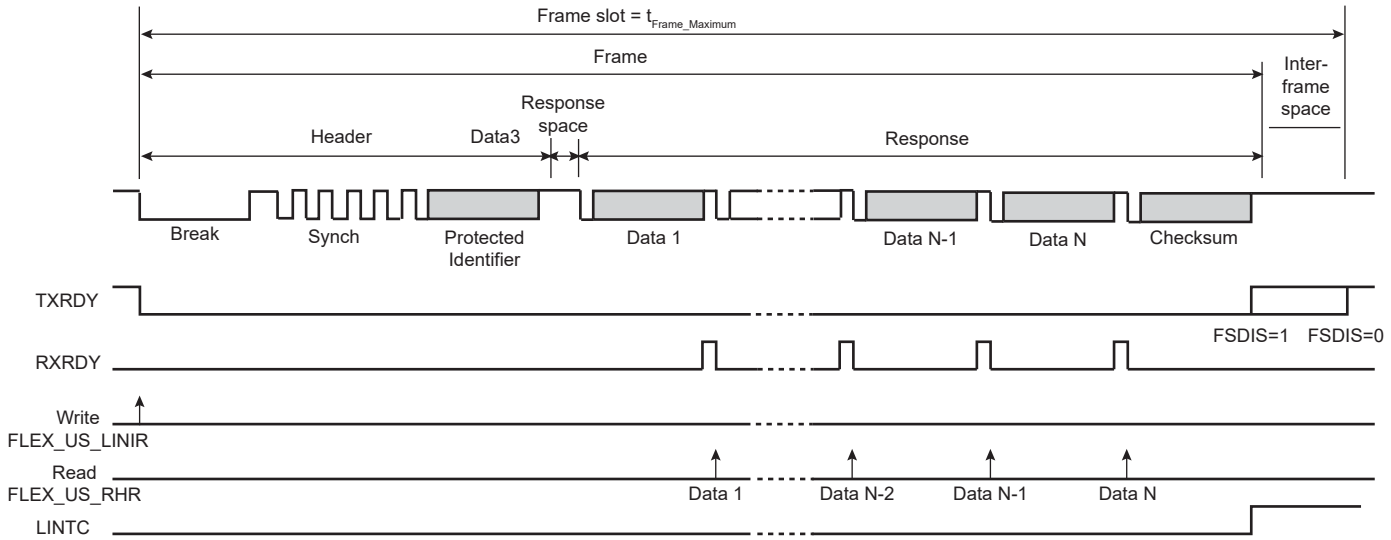
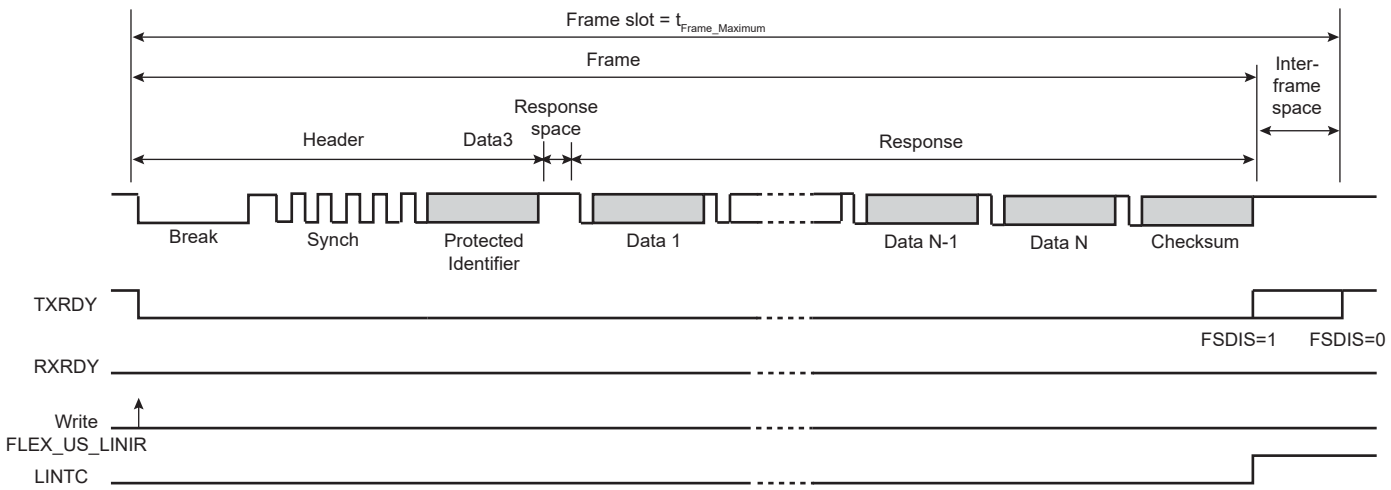



**Figure 44-49. Master Node Configuration, NACT = IGNORE**



### Slave Node Configuration

- Write TXEN and RXEN in FLEX_US_CR to enable both the transmitter and the receiver.
- Write USART_MODE in FLEX_US_MR to select the LIN mode and the slave node configuration.
- Write CD and FP in FLEX_US_BRGR to configure the baud rate.
- Wait until LINID in FLEX_US_CSR rises
- Check LINISFE and LINPE errors
- Read IDCHR in FLEX_US_RHR
- Write NACT, PARDIS, CHKDIS, CHKTYPE, DLCM and DLC in FLEX_US_LINMR to configure the frame transfer.

**IMPORTANT**: If the NACT configuration for this frame is PUBLISH, FLEX_US_LINMR must be written with NACT = PUBLISH even if this field is already correctly configured, in order to set the TXREADY flag and the corresponding write transfer request.

What comes next depends on the NACT configuration:

- Case 1: NACT = PUBLISH, the LIN controller sends the response
  - Wait until TXRDY in FLEX_US_CSR rises
  - Write TCHR in FLEX_US_THR to send a byte
  - If all the data have not been written, redo the two previous steps
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors
- Case 2: NACT = SUBSCRIBE, the USART receives the response
  - Wait until RXRDY in FLEX_US_CSR rises
  - Read RCHR in FLEX_US_RHR
  - If all the data have not been read, redo the two previous steps
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors
- Case 3: NACT = IGNORE, the USART is not concerned by the response
  - Wait until LINTC in FLEX_US_CSR rises
  - Check the LIN errors

**Figure 44-50. Slave Node Configuration, NACT = PUBLISH**

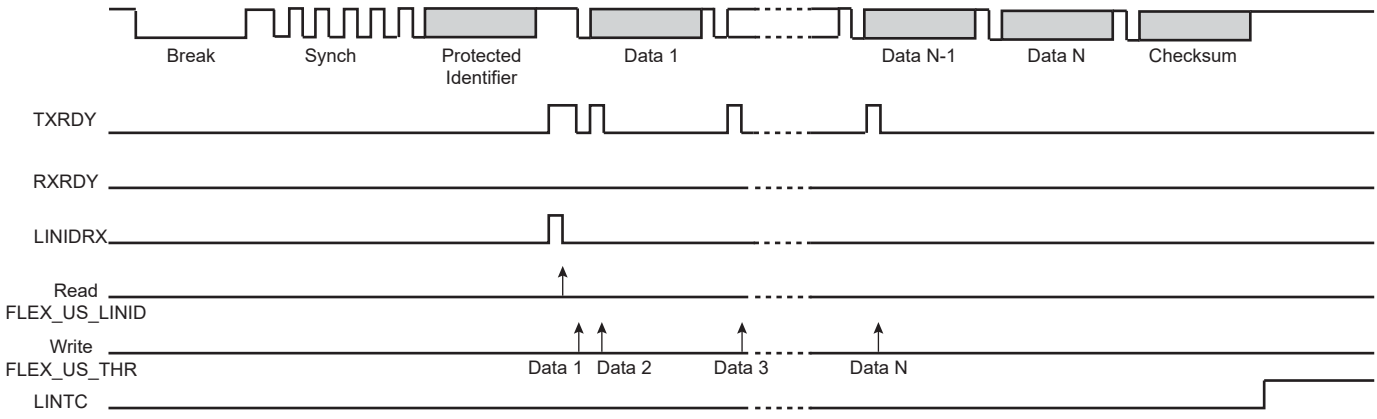


**Figure 44-51. Slave Node Configuration, NACT = SUBSCRIBE**

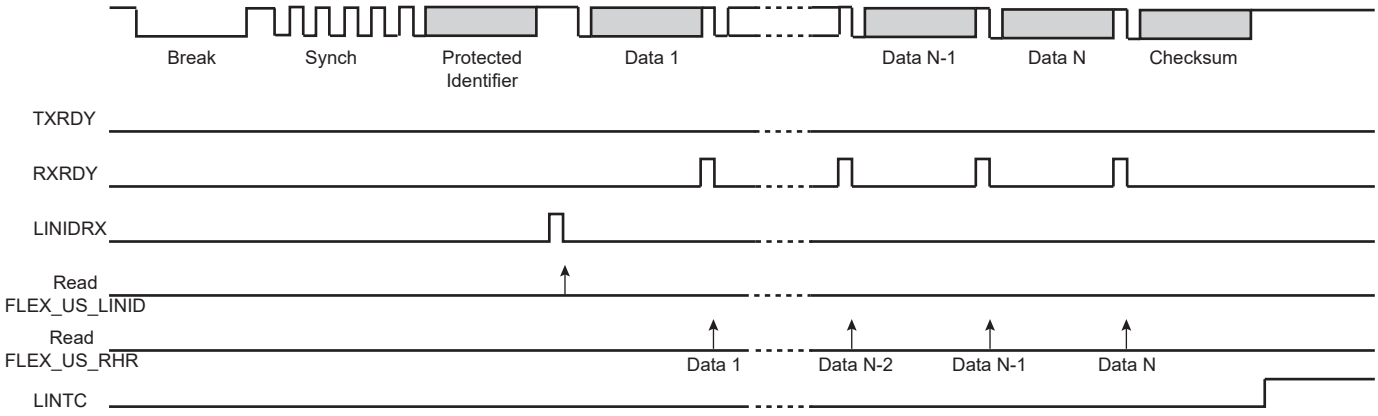
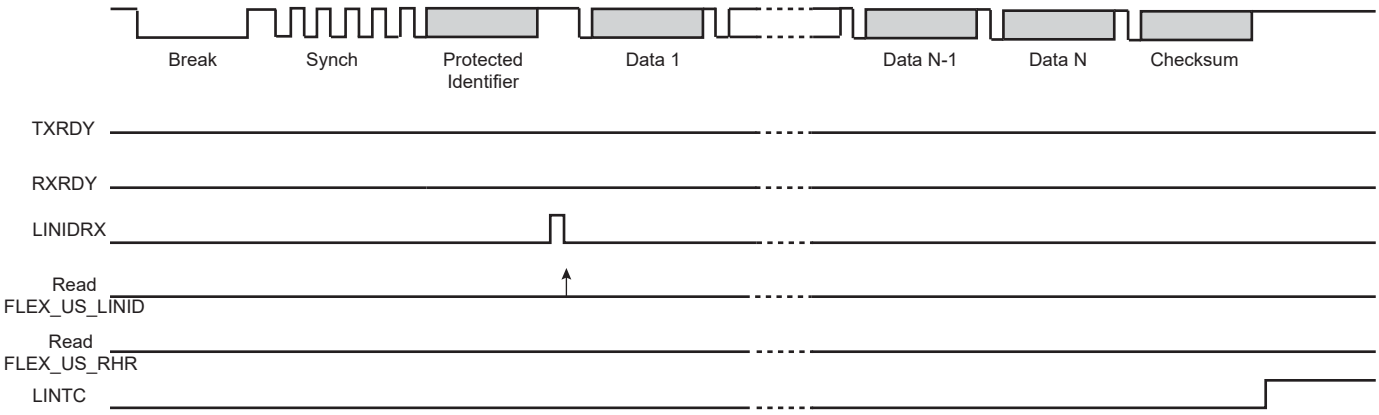



**Figure 44-52. Slave Node Configuration, NACT = IGNORE**




### 44.7.9.16 LIN Frame Handling With The DMAC

The USART can be used in association with the DMAC in order to transfer data directly into/from the on- and off-chip memories without any processor intervention.

The DMAC uses the trigger flags, TXRDY and RXRDY, to write or read into the USART. The DMAC always writes in the Transmit Holding Register (FLEX_US_THR) and it always reads in the Receive Holding Register (FLEX_US_RHR). The size of the data written or read by the DMAC in the USART is always a byte.

#### Master Node Configuration

The user can choose between two DMAC modes by configuring the PDCM bit in FLEX_US_LINMR:

- PDCM = 1: The LIN configuration is stored in the WRITE buffer and it is written by the DMAC in the Transmit Holding register FLEX_US_THR (instead of the LIN Mode register FLEX_US_LINMR). Because the DMAC transfer size is limited to a byte, the transfer is split into two accesses. During the first access the bits, NACT, PARDIS, CHKDIS, CHKTYP, DLM and FSDIS are written. During the second access the 8-bit DLC field is written.
- PDCM = 0: The LIN configuration is not stored in the WRITE buffer and it must be written by the user in FLEX_US_LINMR.

The WRITE buffer also contains the Identifier and the DATA, if the USART sends the response (NACT = PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT = SUBSCRIBE).

**Figure 44-53. Master Node with DMAC (PDCM = 1)**

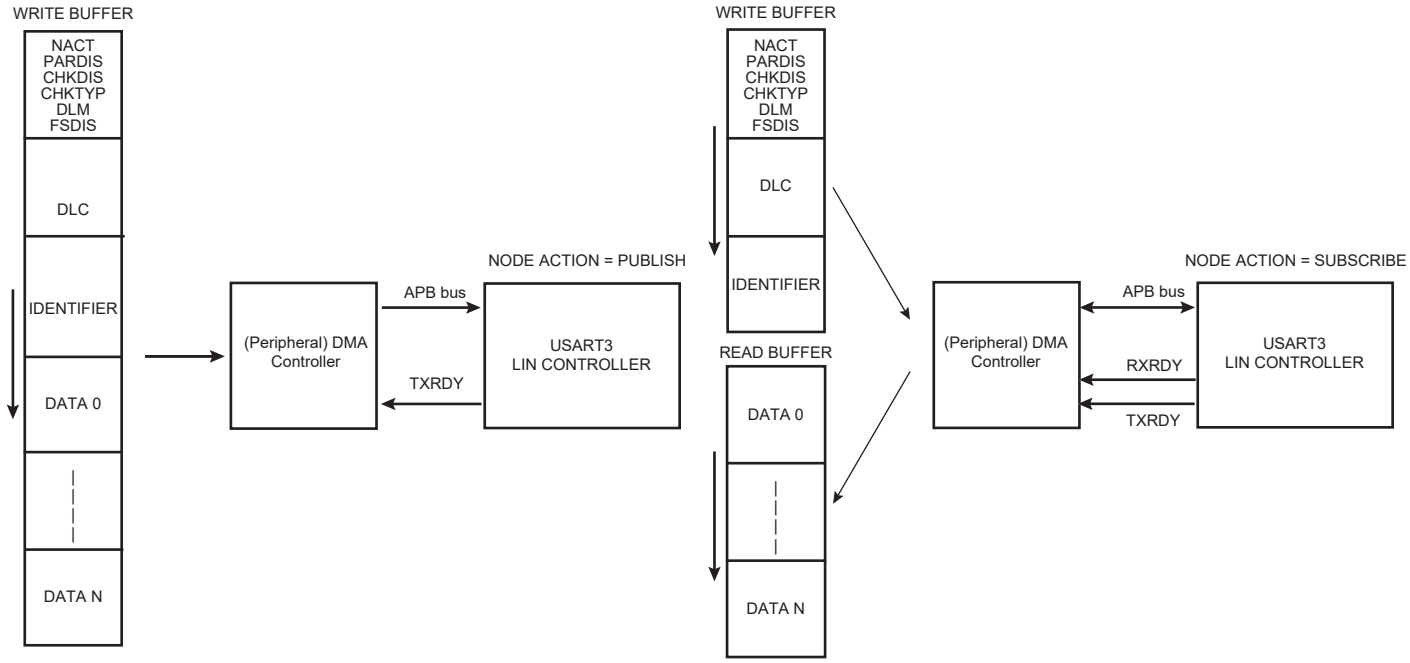
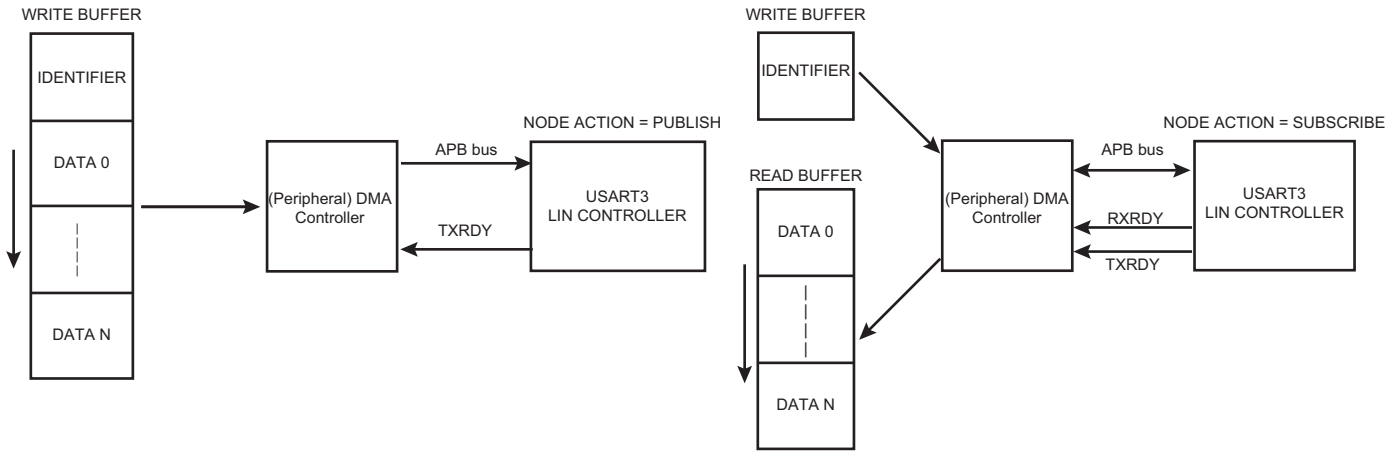



**Figure 44-54. Master Node with DMAC (PDCM = 0)**



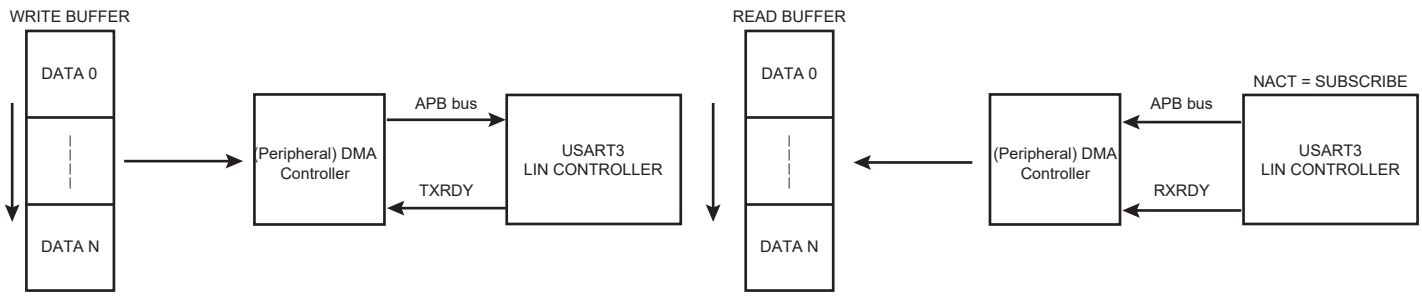

### Slave Node Configuration

In this configuration, the DMAC transfers only the DATA. The Identifier must be read by the user in the LIN Identifier Register (FLEX_US_LINIR). The LIN mode must be written by the user in FLEX_US_LINMR.

The WRITE buffer contains the DATA if the USART sends the response (NACT = PUBLISH).

The READ buffer contains the DATA if the USART receives the response (NACT = SUBSCRIBE).

**Figure 44-55. Slave Node with DMAC**



#### 44.7.9.17 Wakeup Request

Any node in a sleeping LIN cluster may request a wakeup.

In the LIN 2.0 specification, the wakeup request is issued by forcing the bus to the dominant state from 250 µs to 5 ms. For this, it is necessary to send the character 0xF0 in order to impose five successive dominant bits. Whatever the baud rate is, this character respects the specified timings.

- Baud rate min = 1 kbit/s -> $t_{bit}$ = 1 ms -> 5 $t_{bit}$ = 5 ms
- Baud rate max = 20 kbit/s -> $t_{bit}$ = 50 µs -> 5 $t_{bit}$ = 250 µs

In the LIN 1.3 specification, the wakeup request should be generated with the character 0x80 in order to impose eight successive dominant bits.

The user can choose by the WKUPTYP bit in FLEX_US_LINMR either to send a LIN 2.0 wakeup request (WKUPTYP = 0) or to send a LIN 1.3 wakeup request (WKUPTYP = 1).

A wakeup request is transmitted by writing FLEX_US_CR with the LINWKUP bit to 1. Once the transfer is completed, the LINTC flag is asserted in the Status Register (FLEX_US_SR). It is cleared by writing a one to the RSTSTA bit in FLEX_US_CR.

#### 44.7.9.18 Bus Idle Timeout

If the LIN bus is inactive for a certain duration, the slave nodes shall automatically enter in Sleep mode. In the LIN 2.0 specification, this timeout is defined as 4 seconds. In the LIN 1.3 specification, it is defined as 25,000 $t_{bit}$.

In slave Node configuration, the receiver timeout detects an idle condition on the RXD line. When a timeout is detected, the TIMEOUT bit in FLEX_US_CSR rises and can generate an interrupt, thus indicating to the driver to go into Sleep mode.

The timeout delay period (during which the receiver waits for a new character) is programmed in the TO field of FLEX_US_RTOR. If a zero is written to the TO field, the Receiver Timeout is disabled and no timeout is detected. The TIMEOUT bit in FLEX_US_CSR remains at 0. Otherwise, the receiver loads a 17-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit in FLEX_US_CSR rises.

If STTTO is performed, the counter clock is stopped until a first character is received.

If RETTO is performed, the counter starts counting down immediately from the value TO.

**Table 44-16. Receiver Timeout Programming**

| LIN Specification | Baud Rate | Timeout period | TO |
|---|---|---|---|
| 2.0 | 1,000 bit/s | 4s | 4,000 |
| | 2,400 bit/s | | 9,600 |
| | 9,600 bit/s | | 38,400 |
| | 19,200 bit/s | | 76,800 |
| | 20,000 bit/s | | 80,000 |
| 1.3 | – | 25,000 $t_{bit}$ | 25,000 |

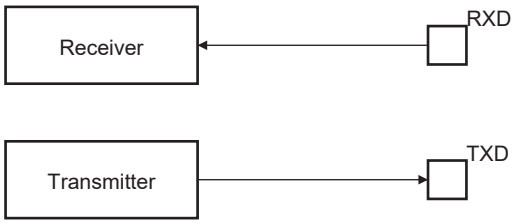
### 44.7.10 Test Modes

The USART can be programmed to operate in three different test modes. The internal loopback capability allows on-board diagnostics. In Loopback mode, the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

#### 44.7.10.1 Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

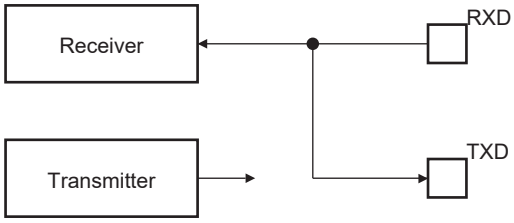
**Figure 44-56. Normal Mode Configuration**




### 44.7.10.2 Automatic Echo Mode

Automatic Echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 44-57. Programming the transmitter has no effect on the TXD pin. The RXD pin is still connected to the receiver input, thus the receiver remains active.

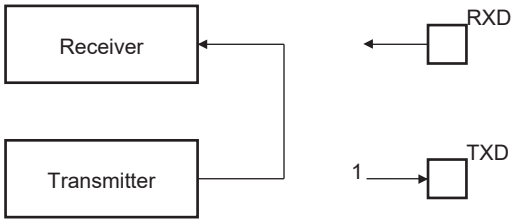
**Figure 44-57. Automatic Echo Mode Configuration**




### 44.7.10.3 Local Loopback Mode

Local Loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 44-58. The TXD and RXD pins are not used. The RXD pin has no effect on the receiver and the TXD pin is continuously driven high, as in idle state.

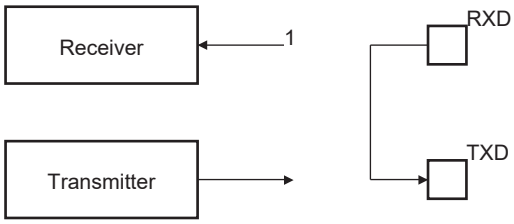
**Figure 44-58. Local Loopback Mode Configuration**




### 44.7.10.4 Remote Loopback Mode

Remote Loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 44-59. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

**Figure 44-59. Remote Loopback Mode Configuration**

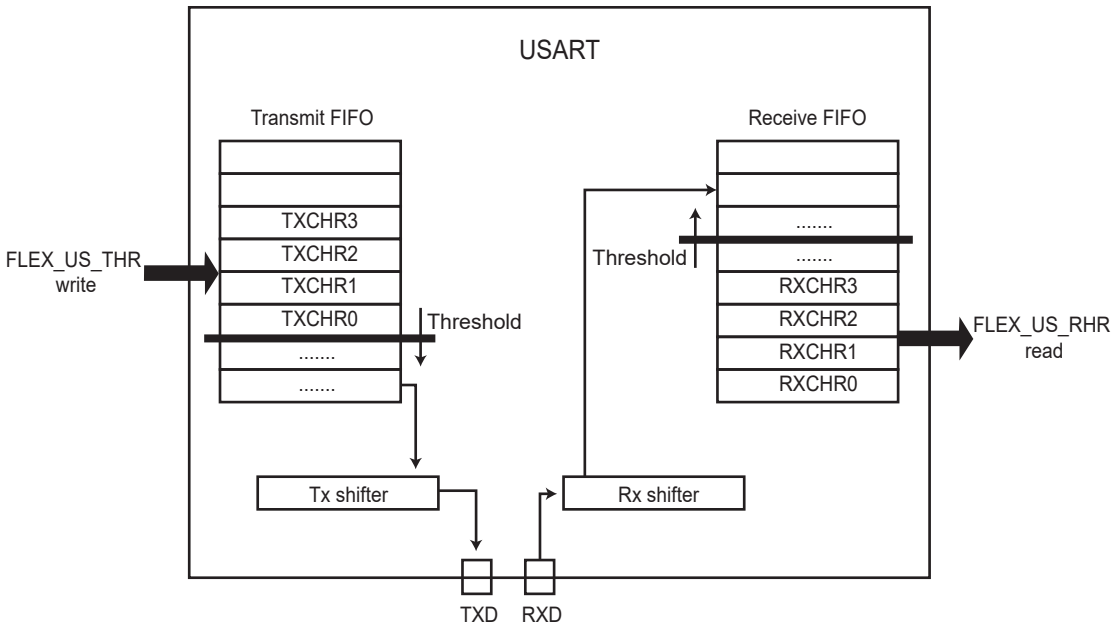


### 44.7.11 FIFOs

The USART includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in FLEX_US_CR. It is recommended to disable both the transmitter and the receiver before enabling or disabling the FIFOs (TXDIS and RXDIS bit in FLEX_US_CR).

Writing a ‘1’ to the FIFOEN bit enables a 32-data Transmit FIFO and a 32-data Receive FIFO.

**Figure 44-60. FIFOs Block Diagram**

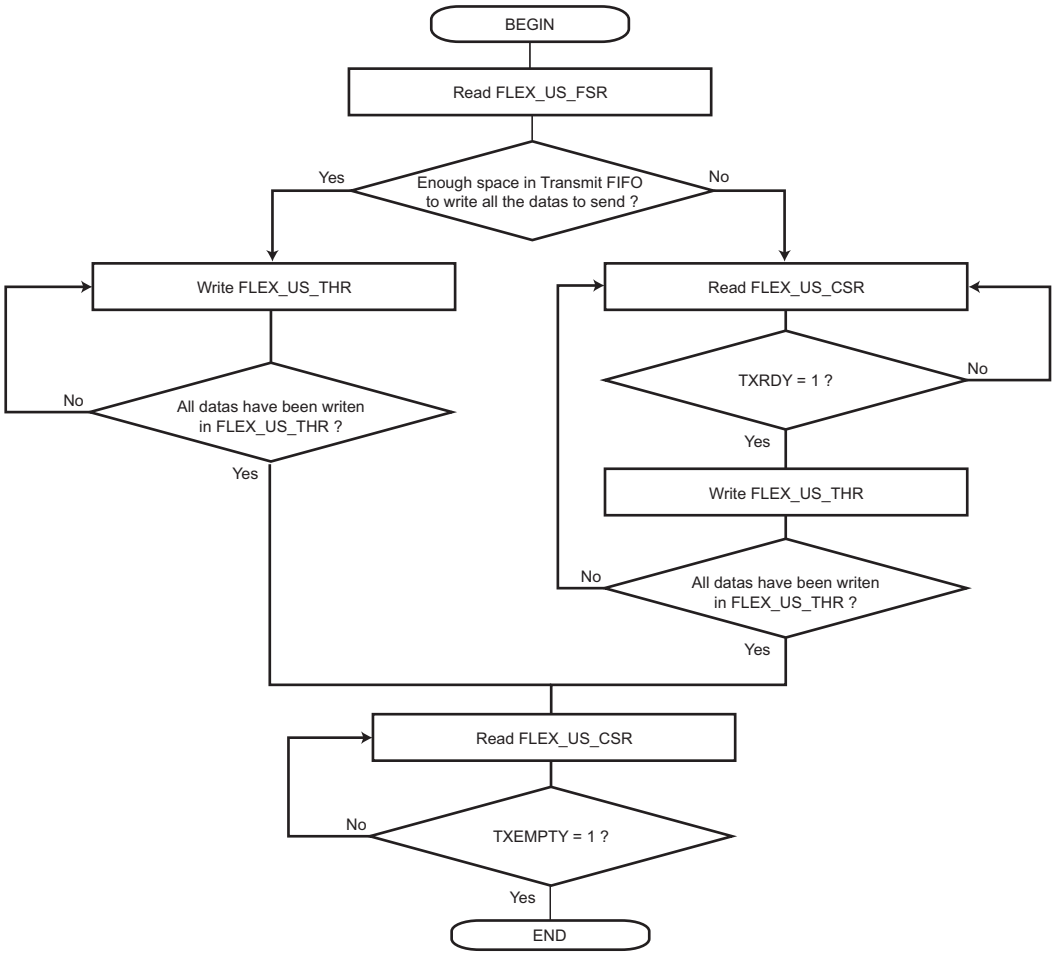



#### 44.7.11.1 Sending Data with FIFO Enabled

With the Transmit FIFO enabled, any write access to FLEX_US_THR loads the written data to the Transmit FIFO. As a consequence it is not mandatory any more to monitor the TXRDY flag state to send multiple data without DMAC.

Knowing the number of data to send, and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in FLEX_US_THR without checking the TXRDY flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the USART FIFO Level Register (FLEX_US_FLR).

**Figure 44-61. Sending Data with FIFO Flowchart**

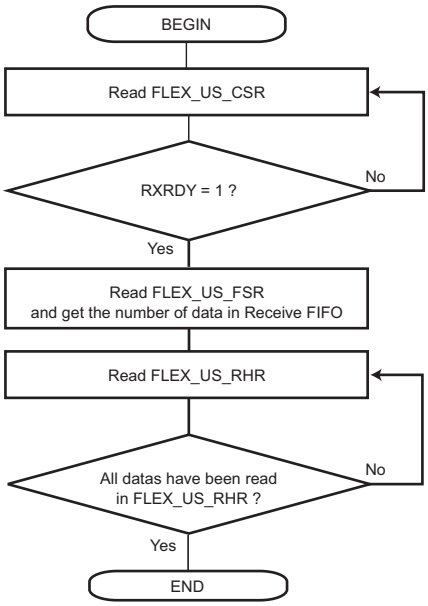



### 44.7.11.2 Receiving Data with FIFO Enabled

With Receive FIFO enabled, any read access on FLEX_US_RHR pulls out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RXRDY flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RXRDY flag set to ‘1’), the exact number of data can be checked with the RXFL field in FLEX_US_FLR and all the data read successively in FLEX_US_RHR without checking the RXRDY flag between each access.

**Figure 44-62. Receiving Data with FIFO Flowchart**




### 44.7.11.3 Clearing/Flushing FIFOs

Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in FLEX_US_CR.

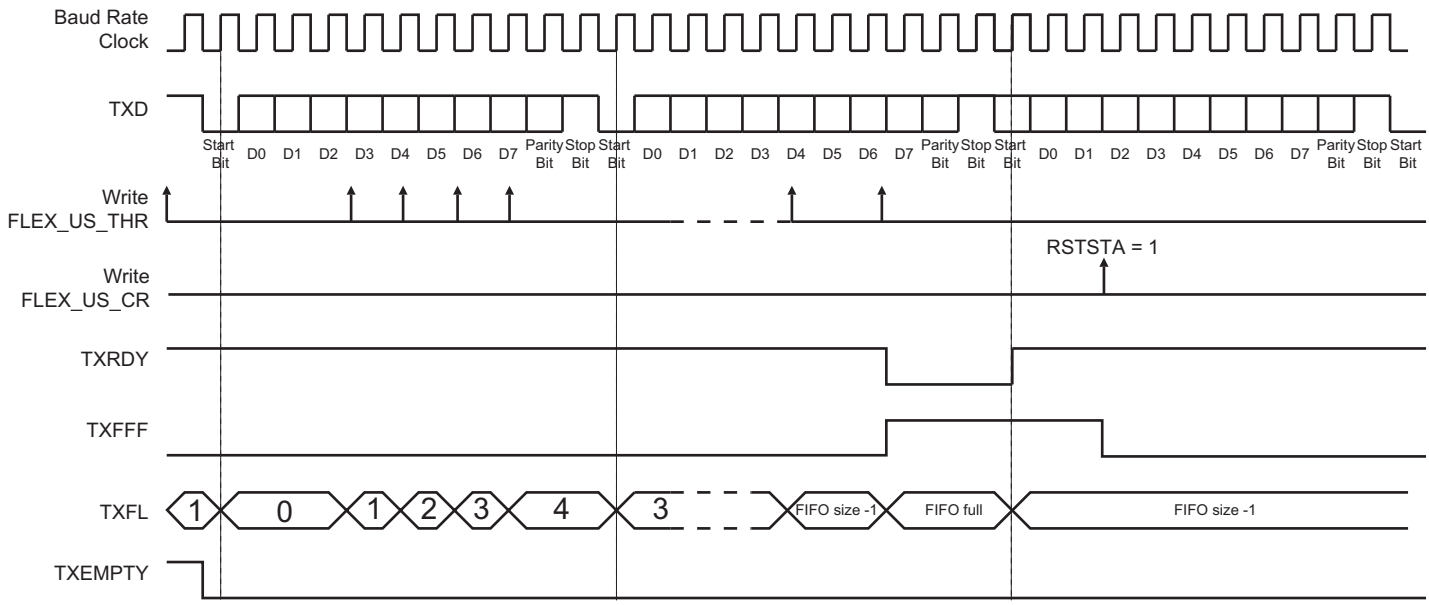
### 44.7.11.4 TXRDY, RXRDY and TXEMPTY Behavior

If FIFOs are enabled, the behavior of the TXRDY, RXRDY and TXEMPTY flags is slightly different.

With FIFO enabled, the TXEMPTY flag remains at ‘0’ state as long as there are characters in the Transmit FIFO or in the Transmit Shift Register. It is at ‘1’ state when there are no character in the Transmit FIFO and no character in the Transmit Shift Register.

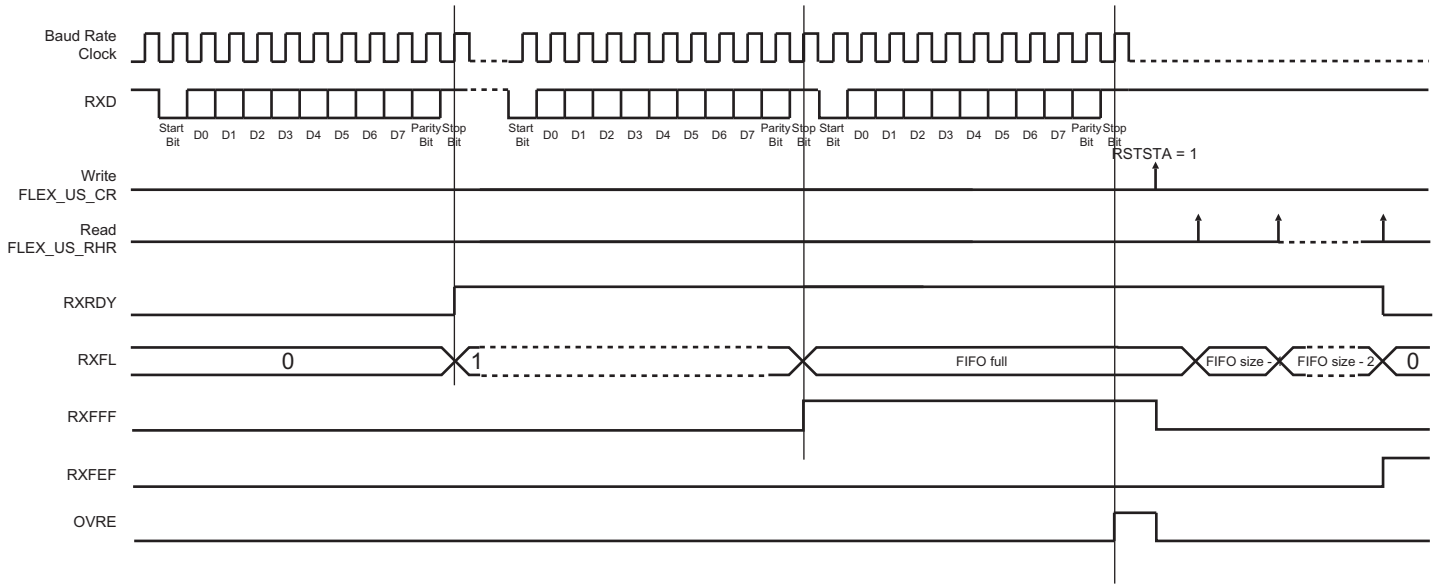
TXRDY indicates if a data can be written in the Transmit FIFO. Then, by default, the TXRDY flag remains at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 44-63. TXRDY in Single Data Mode and TXRDYM = 0x0**



RXRDY indicates if an unread data is present in the Receive FIFO. Then, by default, the RXRDY flag is at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 44-64. RXRDY in Single Data Mode and RXRDYM = 0x0**




TXRDY and RXRDY behavior can be modified using the TXRDYM and RXRDYM fields in the USART FIFO Mode Register (FLEX_US_MR). In Single Data mode, there is no need to modify the TXRDY and RXRDY behavior; however it may be useful in Multiple Data mode.

### 44.7.11.5 Single Data Mode

If the MODE9 bit is set, or if the USART_MODE field is set to either LIN_MASTER or LIN_SLAVE, the FIFOs operate in Single Data mode.

In this mode, only one data can be written/read per write/read access of FLEX_US_THR/FLEX_US_RHR (see Section 44.10.20 “USART Receive Holding Register” and Section 44.10.22 “USART Transmit Holding Register”). On the other hand TXRDYM and RXRDYM fields should be configured with 0x0 value in this mode.

**DMAC**

The TXRDYM and RXRDYM fields must be configured with 0x0 value when working with DMAC transfer in Single Data mode.

The same DMAC procedure applies as with FIFO disabled.

### 44.7.11.6 Multiple Data Mode

When the FIFOs do not operate in Single Data mode, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write/read up to four data in one FLEX_US_THR/FLEX_US_RHR access.

The number of data to write/read is defined by the size of the register access. If the access is a byte-size register access, only one data is written/read, if the access is a halfword size register access, then two data are written/read and, finally, if the access is a word-size register access, four data are written/read.

Written/Read data are always right-aligned, as shown in Section 44.10.21 “USART Receive Holding Register (FIFO Multi Data)” and Section 44.10.23 “USART Transmit Holding Register (FIFO Multi Data)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six FLEX_US_THR byte write accesses.
- Perform three FLEX_US_THR halfword write accesses.

- Perform one FLEX_US_THR word write access and one FLEX_US_THR halfword write access.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six FLEX_US_RHR byte read accesses.
- Perform three FLEX_US_RHR halfword read accesses.
- Perform one FLEX_US_RHR word read access and one FLEX_US_RHR halfword read access.

This mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

#### TXRDY and RXRDY Configuration

In Multiple Data mode, the TXRDYM and RXRDYM fields in FLEX_US_FMR become useful.

As in Multiple Data mode, it is possible to write several data in the same access it might be useful to configure TXRDY flag behavior to indicate if 1, 2 or 4 data can be written in the FIFO depending on the access to perform on FLEX_US_THR.

If, for instance, four data are written each time in FLEX_US_THR, it might be useful to configure the TXRDYM field to value 0x2 so that the TXRDY flag is at ‘1’ only when at least four data can be written in the Transmit FIFO.

In the same way, if four data are read each time in FLEX_US_RHR, it might be useful to configure the RXRDYM field to value 0x2 so that the RXRDY flag is at ‘1’ only when at least four unread data are in the Receive FIFO.

#### DMAC

If DMAC transfer is used, it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

### 44.7.11.7 Transmit FIFO Lock

- LIN Mode:

If a frame is aborted using the Abort LIN Transmission bit (FLEX_US_CR.LINABT), a lock is set on the Transmit FIFO preventing any new frame from being sent until it is cleared. This allows clearing the FIFO if needed, reset DMAC channels, etc., without any risk.

The TXFLOCK bit in the USART FIFO Event Status Register (FLEX_US_FESR) is used to check the state of the Transmit FIFO lock.

The Transmit FIFO lock can be cleared by setting the TXFLCLR bit in FLEX_US_CR.

### 44.7.11.8 FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

If the Transmit FIFO is full and a write access is performed on FLEX_US_THR, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in FLEX_US_FESR.

In Multiple Data mode, if the number of data written in FLEX_US_THR (according to the register access size) is bigger than the Transmit FIFO free space, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in FLEX_US_FESR.

- Receive FIFO:

In Multiple Data mode, if the number of data read in FLEX_US_RHR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it generates a Receive FIFO pointer error and sets the RXFPTEF flag in FLEX_US_FESR.

No pointer error should happen if the FIFO state is checked before writing/reading in FLEX_US_THR/FLEX_US_RHR. The FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a Transmit pointer error occurs, the transmitter must be reset through the RSTTX bit in FLEX_US_CR.

If a Receive pointer error occurs, the receiver must be reset through the RSTRX bit in FLEX_US_CR.

#### 44.7.11.9 FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

The Transmit FIFO threshold can be set using the TXFTHRES field in FLEX_US_FMR. Each time the Transmit FIFO goes from the 'above threshold' to the 'equal or below threshold' state, the TXFTHF flag in FLEX_US_FESR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

The Receive FIFO threshold can be set using the RXFTHRES field in FLEX_US_FMR. Each time the Receive FIFO goes from the 'below threshold' to the 'equal to or above threshold' state, the RXFTHF flag in FLEX_US_FESR is set. This enables the application to know that the Receive FIFO reached the defined threshold and read some data before it becomes full. The Receive FIFO threshold 2 can be set with RXFTHRES2 field in FLEX_US_FMR. Each time the Receive FIFO goes from the 'above threshold 2' to the 'equal or below threshold 2' state, the RXFTHF2 flag in FLEX_US_FESR is set. This enables the application to know that the Receive FIFO reached the defined threshold.

The RXFTHF, RXFTHF and RXTHF2 flags can be configured to generate an interrupt using FLEX_US_FIER and FLEX_US_FIDR.

#### 44.7.11.10 FIFO Flags

FIFOs come with a set of flags which can be configured each to generate an interrupt through FLEX_US_FIER and FLEX_US_FIDR.

FIFO flags state can be read in FLEX_US_FESR. They are cleared writing to '1' the RSTSTA bit in FLEX_US_CR.

### 44.7.12 USART Register Write Protection

The FLEXCOM operating mode (FLEX_MR.OPMODE) must be set to FLEX_MR_OPMODE_USART to enable access to the write protection registers.

To prevent any single software error from corrupting USART behavior, certain registers in the address space can be write-protected by setting the WPEN (Write Protection Enable) bit in the USART Write Protection Mode Register (FLEX_US_WPMR).

If a write access to a write-protected register is detected, the Write Protection Violation Status (WPVS) flag in the USART Write Protection Status Register (FLEX_US_WPSR) is set and the Write Protection Violation Source (WPVSRC) field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading FLEX_US_WPSR.

The following registers can be write-protected when WPEN is set:

- USART Mode Register
- USART Baud Rate Generator Register
- USART Receiver Timeout Register
- USART Transmitter Timeguard Register
- USART FI DI RATIO Register
- USART IrDA FILTER Register
- USART Manchester Configuration Register
- USART Comparison Register

## 44.8 SPI Functional Description

### 44.8.1 Modes of Operation

The SPI operates in Master mode or in Slave mode.

- The SPI operates in Master mode by writing a 1 to the MSTR bit in the SPI Mode Register (FLEX_SPI_MR):
  - The pins NPCS0 to NPCS1 are all configured as outputs
  - The SPCK pin is driven
  - The MISO line is wired on the receiver input
  - The MOSI line is driven as an output by the transmitter.
- The SPI operates in Slave mode if the MSTR bit in FLEX_SPI_MR is written to 0:
  - The MISO line is driven by the transmitter output
  - The MOSI line is wired on the receiver input
  - The SPCK pin is driven by the transmitter to synchronize the receiver.
  - The NPCS0 pin becomes an input, and is used as a slave select signal (NSS)
  - Pin NPCS1 is not are not are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operation. The bit rate generator is activated only in Master mode.

### 44.8.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the SPI Chip Select Register (FLEX_SPI_CSR). The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data are driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Consequently, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are connected and require different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

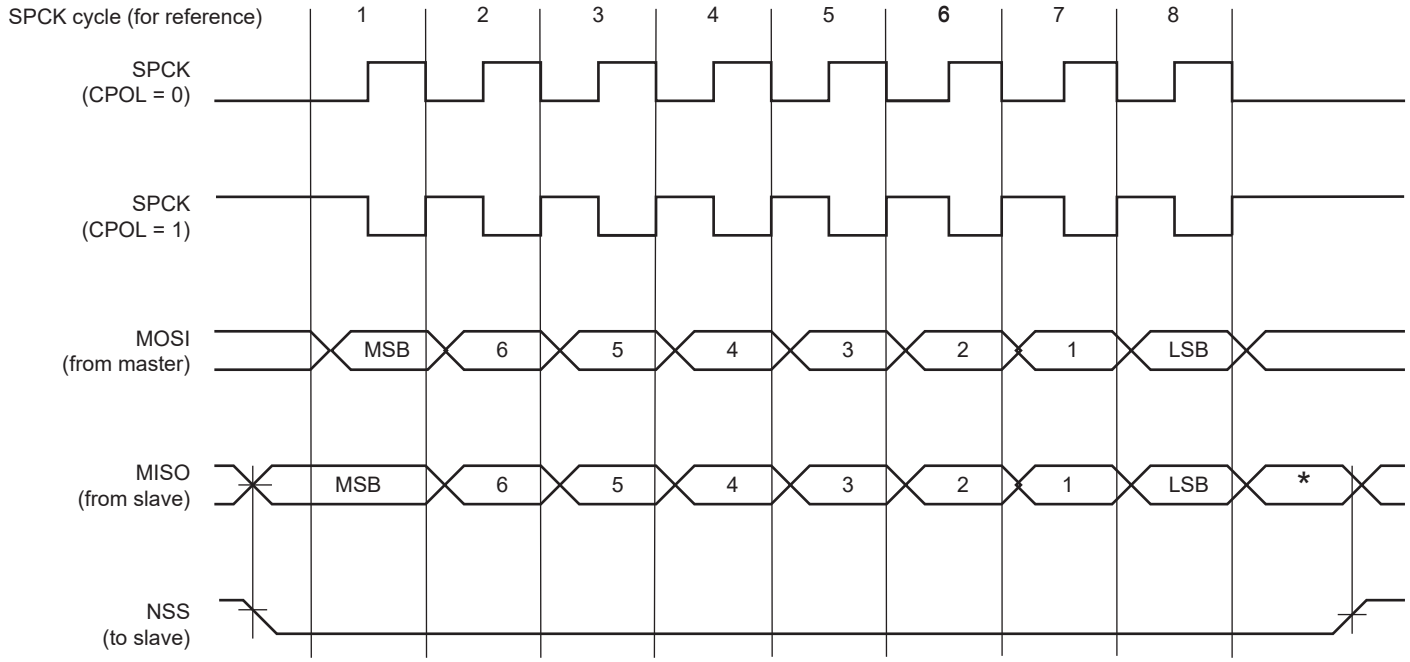
Table 44-17 shows the four modes and corresponding parameter settings.

**Table 44-17. SPI Bus Protocol Mode**

| SPI Mode | CPOL | NCPHA | Shift SPCK Edge | Capture SPCK Edge | SPCK Inactive Level |
|---|---|---|---|---|---|
| 0 | 0 | 1 | Falling | Rising | Low |
| 1 | 0 | 0 | Rising | Falling | Low |
| 2 | 1 | 1 | Rising | Falling | High |
| 3 | 1 | 0 | Falling | Rising | High |

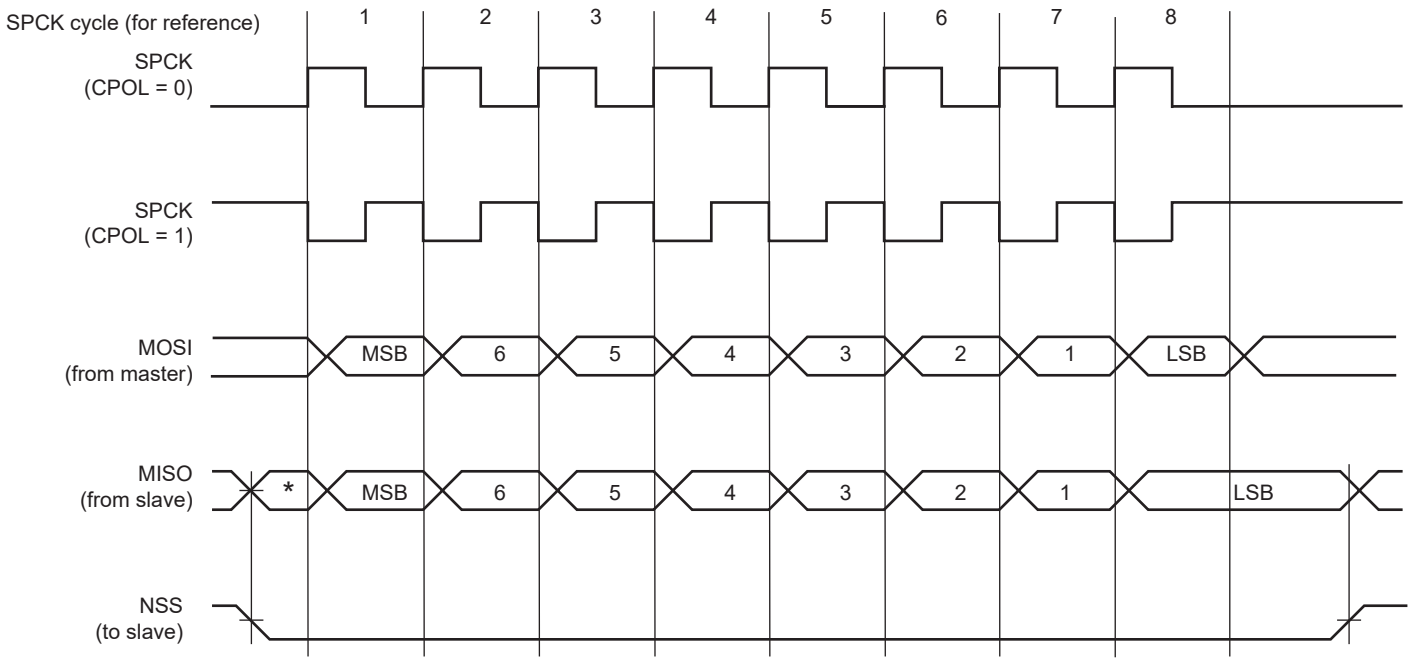
Figure 44-65 and Figure 44-66 show examples of data transfers.

**Figure 44-65. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**




* Not defined.

**Figure 44-66. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**




* Not defined.

### 44.8.3 Master Mode Operations

When configured in Master mode, the SPI operates on the clock generated by the internal programmable bit rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register (FLEX_SPI_TDR) and the Receive Data Register (FLEX_SPI_RDR), and a single shift register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer starts when the processor writes to FLEX_SPI_TDR. The written data are immediately transferred in the Shift register and the transfer on the SPI bus starts. While the data in the Shift register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift register. Data cannot be loaded in FLEX_SPI_RDR without transmitting data. If there is no data to transmit, a dummy data can be used (FLEX_SPI_TDR filled with ones). When the WDRBT bit is set, a new data cannot be transmitted if FLEX_SPI_RDR has not been read. If Receiving mode is not required, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the SPI Status Register (FLEX_SPI_SR) can be discarded.

Before writing the TDR, the PCS field in FLEX_SPI_MR must be set in order to select a slave.

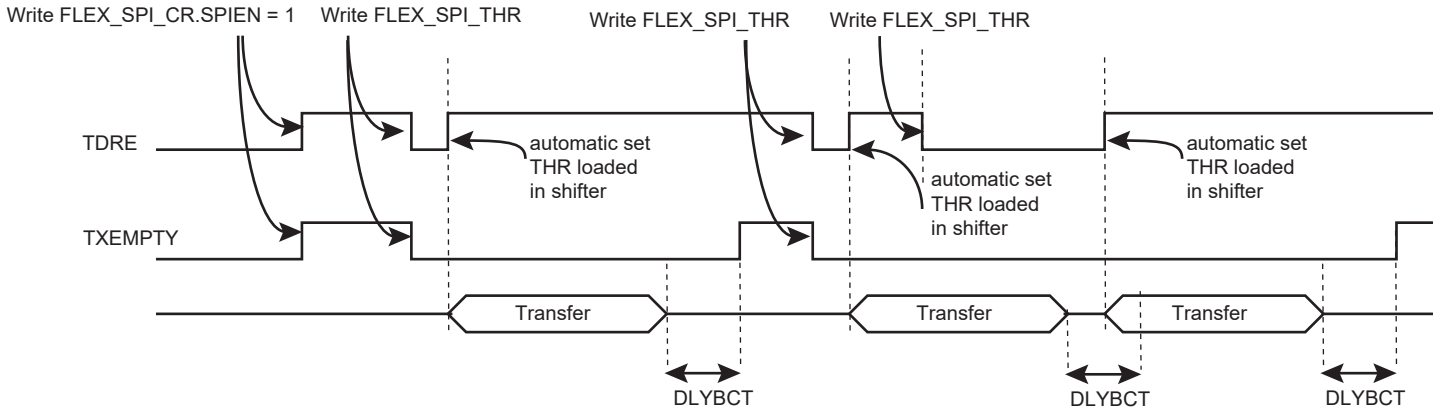
If new data are written in FLEX_SPI_TDR during the transfer, it is kept in FLEX_SPI_TDR until the current transfer is completed. Then, the received data are transferred from the Shift register to FLEX_SPI_RDR, the data in FLEX_SPI_TDR is loaded in the Shift register and a new transfer starts.

As soon as the FLEX_SPI_TDR is written, the Transmit Data Register Empty (TDRE) flag in the FLEX_SPI_SR is cleared. When the data written in the FLEX_SPI_TDR is loaded into the Shift register, the TDRE flag in the FLEX_SPI_SR is set. The TDRE bit is used to trigger the Transmit DMA channel (see Figure 44-67).

The end of transfer is indicated by the TXEMPTY flag in FLEX_US_SR. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of this delay. The peripheral clock can be switched off at this time.

Notes:
1. When the SPI is enabled, the TDRE and TXEMPTY flags are set.
2. The TXEMPTY flag alone cannot be used to detect the end of the buffer DMA transfer.

**Figure 44-67. TDRE and TXEMPTY Flag Behavior**



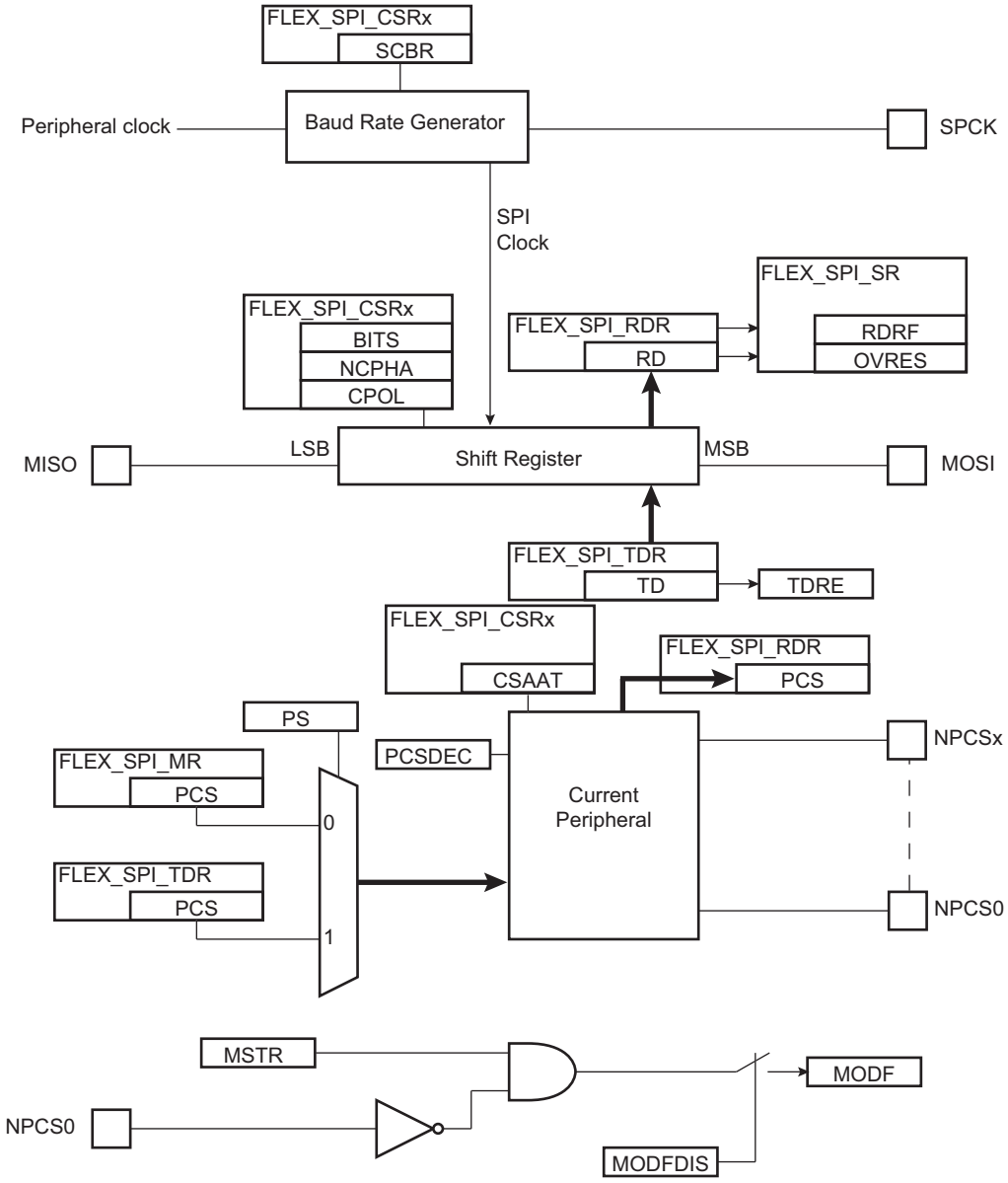

The transfer of received data from the Shift register to FLEX_SPI_RDR is indicated by the Receive Data Register Full (RDRF) bit in FLEX_SPI_SR. When the received data are read, the RDRF bit is cleared.

If FLEX_SPI_RDR has not been read before new data are received, the Overrun Error bit (OVRES) in FLEX_SPI_SR is set. As long as this flag is set, data are loaded in FLEX_SPI_RDR. The user has to read the status register to clear the OVRES bit.

Figure 44-68 shows a block diagram of the SPI when operating in Master mode. Figure 44-69 shows a flow chart describing how transfers are handled.

### 44.8.3.1 Master Mode Block Diagram

**Figure 44-68. Master Mode Block Diagram**



### 44.8.3.2 Master Mode Flow Diagram

**Figure 44-69. Master Mode Flow Diagram**

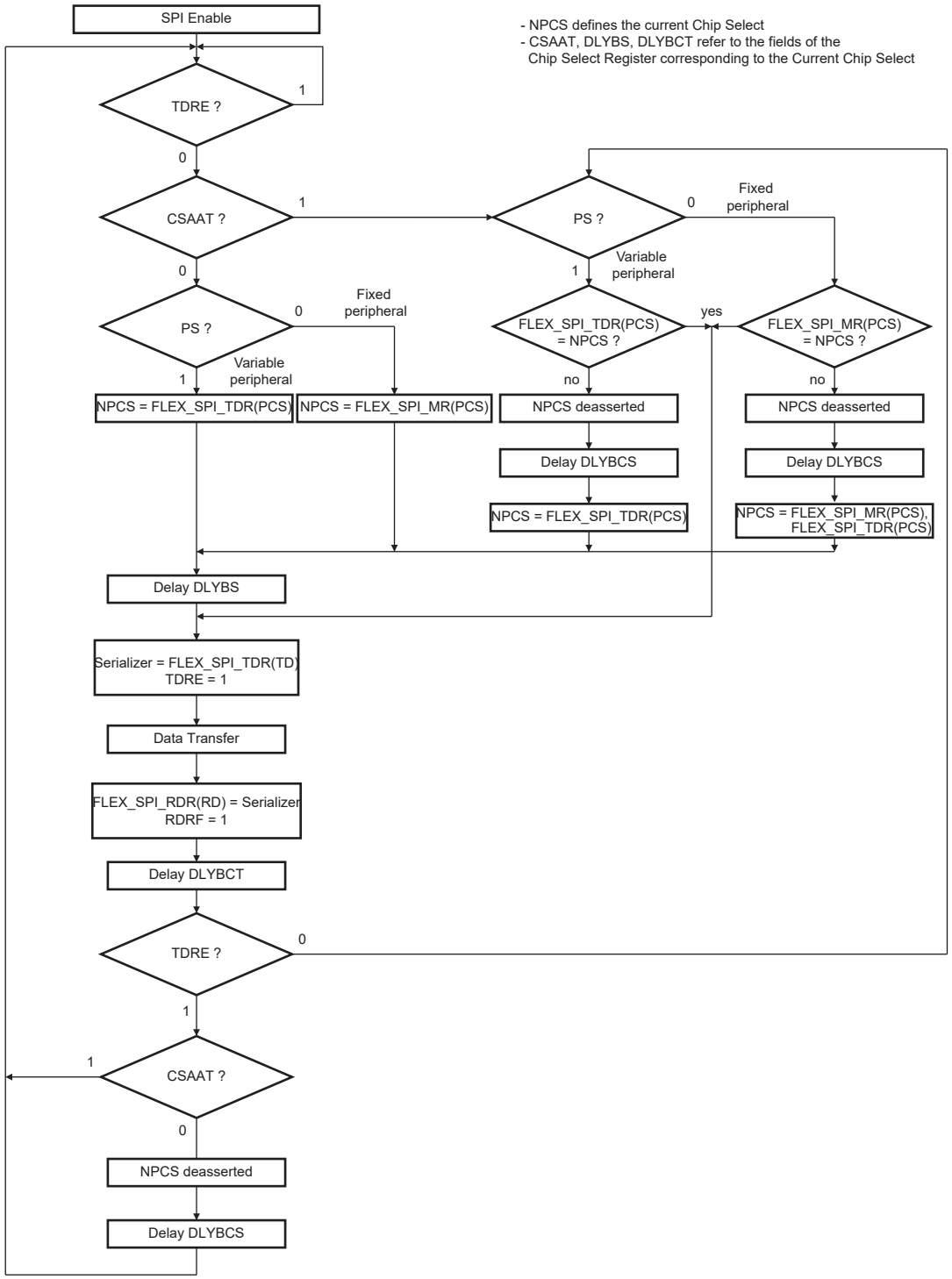
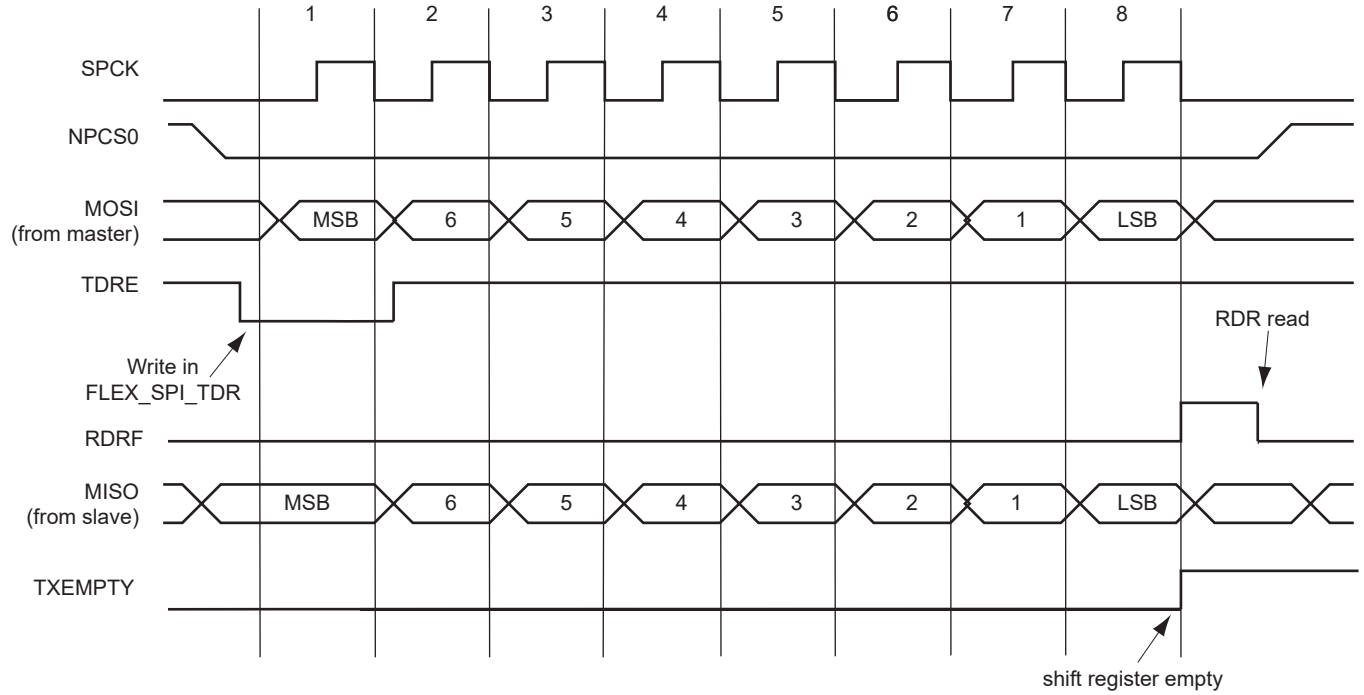



Figure 44-70 shows the behavior of Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags within FLEX_SPI_SR during an 8-bit data transfer in Fixed mode without the DMAC involved.

**Figure 44-70. Status Register Flags Behavior**




### 44.8.3.3 Clock Generation

The SPI bit rate clock is generated by dividing a source clock which can be the peripheral clock or a programmable clock from the GCLK. The divider can be a value between 1 and 255.

If the SCBR field is programmed to 1 and the clock source is GCLK, the operating bit rate is peripheral clock (see the electrical characteristics section for the SPCK maximum frequency). Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it to a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field in FLEX_SPI_CSR. This allows the SPI to automatically adapt the bit rate for each interfaced peripheral without reprogramming.

If GCLK is selected as source clock (BRSRCCLK = 1 in FLEX_SPI_MR), the bit rate is independent of the processor/bus clock. Thus the processor clock can be changed while SPI is enabled. The processor clock frequency changes must be performed only by programming the PRES field in PMC_MCKR (see PMC section). Other methods to modify the processor/bus clock frequency (PLL multiplier, etc.) are forbidden when SPI is enabled.

The peripheral clock frequency must be at least three times higher than GCLK.

### 44.8.3.4 Transfer Delays

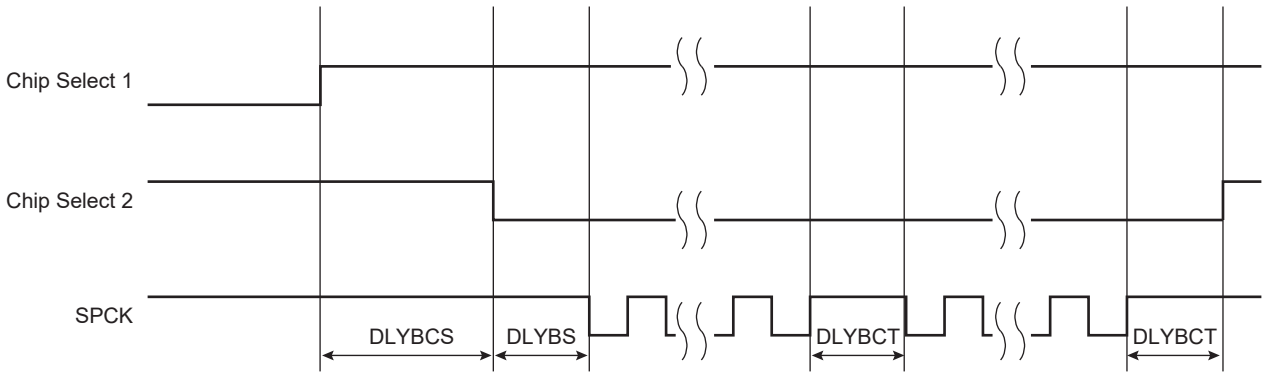
Figure 44-71 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- The delay between the chip selects. It is programmable only once for all chip selects by writing the DLYBCS field in FLEX_SPI_MR. The SPI slave device deactivation delay is managed through DLYBCS. If there is only one SPI slave device connected to the master, the DLYBCS field does not need to be configured. If several slave devices are connected to a master, DLYBCS must be configured depending on the highest deactivation delay. Refer to the SPI slave device electrical characteristics.

- The delay before SPCK, independently programmable for each chip select by writing the DLYBS field. The SPI slave device activation delay is managed through DLYBS. Refer to the SPI slave device electrical characteristics to define DLYBS.
- The delay between consecutive transfers, independently programmable for each chip select by writing the DLYBCT field. The time required by the SPI slave device to process received data is managed through DLYBCT. This time depends on the SPI slave system activity.

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

**Figure 44-71. Programmable Delays**




#### 44.8.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS1 signals. By default, all NPCS signals are high before and after each transfer.

- **Fixed Peripheral Select Mode**: SPI exchanges data with only one peripheral.
  Fixed Peripheral Select mode is enabled by writing the PS bit to zero in FLEX_SPI_MR. In this case, the current peripheral is defined by the PCS field in FLEX_SPI_MR and the PCS field in FLEX_SPI_TDR has no effect.
- **Variable Peripheral Select Mode:** Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in FLEX_SPI_MR.
  Variable Peripheral Select Mode is enabled by setting the PS bit to one in FLEX_SPI_MR. The PCS field in FLEX_SPI_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data. The value to write in FLEX_SPI_TDR has the following format:

  [xxxxxxx(7-bit) + LASTXFER(1-bit)[(1)]+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals the chip select to assert, as defined in Section 44.10.48 “SPI Transmit Data Register” and LASTXFER bit at 0 or 1 depending on the CSAAT bit.

Note: 1. Optional

The CSAAT, LASTXFER and CSNAAT bits are discussed in Section 44.8.3.9 “Peripheral Deselection with DMA”.

If LASTXFER is used, the command must be issued after writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the DMA transfer, it is necessary to wait for the TXEMPTY flag and then write SPIDIS into the SPI Control Register (FLEX_SPI_CR). This does not change the configuration register values). The NPCS is disabled after the last character transfer. Then, another DMA transfer can be started if the SPIEN has previously been written in FLEX_SPI_CR.

#### 44.8.3.6 SPI Direct Access Memory Controller (DMAC)

In both Fixed and Variable modes, the Direct Memory Access Controller (DMAC) can be used to reduce processor overhead.

The fixed peripheral selection allows buffer transfers with a single peripheral. Using the DMAC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, if the peripheral selection is modified, FLEX_SPI_MR must be reprogrammed.

The variable peripheral selection allows buffer transfers with multiple peripherals without reprogramming FLEX_SPI_MR. Data written in FLEX_SPI_TDR is 32 bits wide and defines the real data to be transmitted and the destination peripheral. Using the DMAC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs. However, the SPI still controls the number of bits (8 to 16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in terms of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

#### 44.8.3.7 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 3 slave peripherals by decoding the two chip select lines, NPCS0 to NPCS1 with an external decoder/demultiplexer (refer to Figure 44-72). This can be enabled by setting the PCSDEC bit in FLEX_SPI_MR.

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

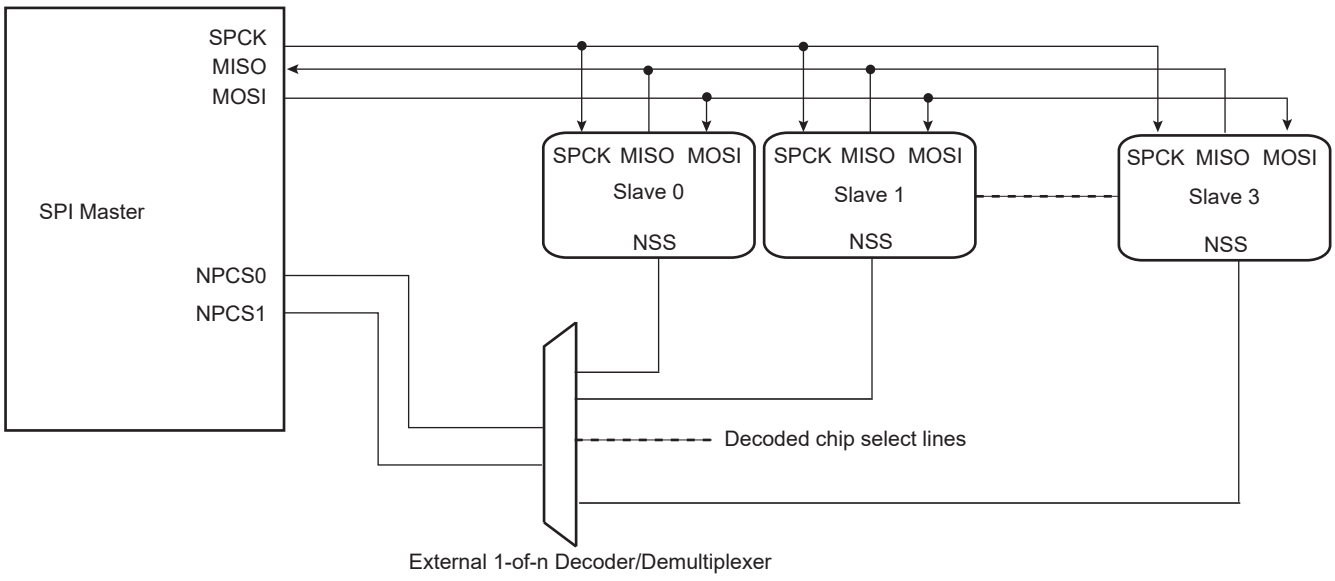
When operating with decoding, the SPI directly outputs the value defined by the PCS field on the NPCS lines of either FLEX_SPI_MR or FLEX_SPI_TDR (depending on PS).

As the SPI sets a default value of 0x3 on the chip select lines (i.e., all chip select lines at 1) when not processing any transfer, only 3 peripherals can be decoded.

The SPI has only two Chip Select registers. As a result, when external decoding is activated, each NPCS chip select defines the characteristics of up to two peripherals. As an example, FLEX_SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 1, corresponding to the PCS values 0x0 to 0x1. Consequently, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 1 and 2. Figure 44-72 shows this type of implementation.

If the CSAAT bit is used, with or without the DMAC, the mode fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since mode fault detection is only on NPCS0.

**Figure 44-72. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation**



#### 44.8.3.8 Peripheral Deselection without DMA

During a transfer of more than one data on a Chip Select without the DMA, FLEX_SPI_TDR is loaded by the processor, the TDRE flag rises as soon as the content of FLEX_SPI_TDR is transferred into the internal Shift register. When this flag is detected high, FLEX_SPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload FLEX_SPI_TDR in time to keep the chip select active (low). A null DLYBCT value (delay between consecutive transfers) in FLEX_SPI_CSR, gives even less time for the processor to reload FLEX_SPI_TDR. With some SPI slave peripherals, if the chip select line must remain active (low) during a full set of transfers, communication errors can occur.

To facilitate interfacing with such devices, the Chip Select registers [CSR0...CSR1] can be programmed with the Chip Select Active After Transfer (CSAAT) bit to 1. This allows the chip select lines to remain in their current state (low = active) until a transfer to another chip select is required. Even if FLEX_SPI_TDR is not reloaded, the chip select remains active. To de-assert the chip select line at the end of the transfer, the Last Transfer (LASTXFER) bit in FLEX_SPI_CR must be set after writing the last data to transmit into FLEX_SPI_TDR.

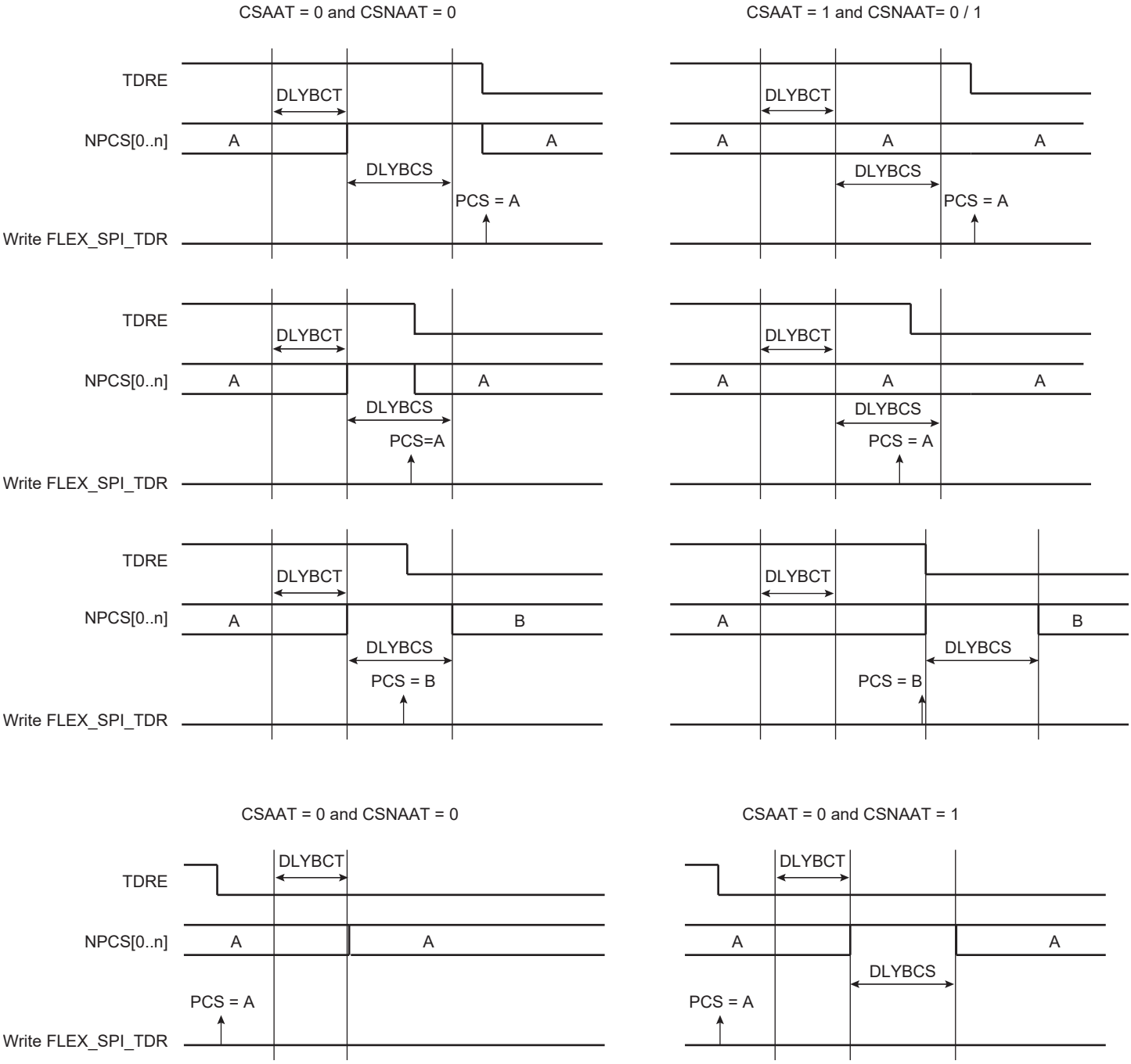
#### 44.8.3.9 Peripheral Deselection with DMA

DMA provides faster reloads of FLEX_SPI_TDR compared to software. However, depending on the system activity, it is not guaranteed that FLEX_SPI_TDR is written with the next data before the end of the current transfer. Consequently, a data can be lost by the de-assertion of the NPCS line for SPI slave peripherals requiring the chip select line to remain active between two transfers. The only way to guarantee a safe transfer in this case is the use of the CSAAT and LASTXFER bits.

When the CSAAT bit is cleared, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the TDRE flag rises as soon as the content of FLEX_SPI_TDR is transferred into the internal shift register. When this flag is detected, FLEX_SPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This can lead to difficulties to interface with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, FLEX_SPI_CSR can be programmed with the Chip Select Not Active After Transfer (CSNAAT) bit to 1. This allows the chip select lines to be de-asserted systematically during a time “DLYBCS” (the value of the CSNAAT bit is processed only if the CSAAT bit is cleared for the same chip select).

Figure 44-73 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

**Figure 44-73. Peripheral Deselection**




### 44.8.3.10 Mode Fault Detection

The SPI has the capability to operate in multi-master environment. Consequently, the NPCS0/NSS line must be monitored. If one of the masters on the SPI bus is currently transmitting, the NPCS0/NSS line is low and the SPI must not transmit a data. A mode fault is detected when the SPI is programmed in Master mode and a low level is driven by an external master on the NPCS0/NSS signal. In multi-master environment, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the MODF bit in FLEX_SPI_SR is set until FLEX_SPI_SR is read and the SPI is automatically disabled until it is re-enabled by writing the SPIEN bit in FLEX_SPI_CR to 1.

By default, the mode fault detection is enabled. The user can disable it by setting the MODFDIS bit in FLEX_SPI_MR.

### 44.8.4 SPI Slave Mode

When operating in Slave mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits until NSS goes active before receiving the serial clock from an external master. When NSS falls, the clock is validated and the data are loaded in FLEX_SPI_RDR according to the configuration value of the BITS field in FLEX_SPI_CSR0. These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits in FLEX_SPI_CSR0. Note that the BITS field, CPOL bit and NCPHA bit of the other Chip Select registers have no effect when the SPI is programmed in Slave mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

Note: For more information on the BITS field, see also the note below the FLEX_SPI_CSRx register bitmap in Section 44.10.54 “SPI Chip Select Register”.

When all bits are processed, the received data are transferred in FLEX_SPI_RDR and the RDRF bit rises. If FLEX_SPI_RDR has not been read before new data are received, the Overrun Error bit (OVRES) in FLEX_SPI_SR is set. As long as this flag is set, data are loaded in FLEX_SPI_RDR. The user must read FLEX_SPI_SR to clear the OVRES bit.

When a transfer starts, the data shifted out is the data present in the Shift register. If no data has been written in FLEX_SPI_TDR, the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift register resets to 0.

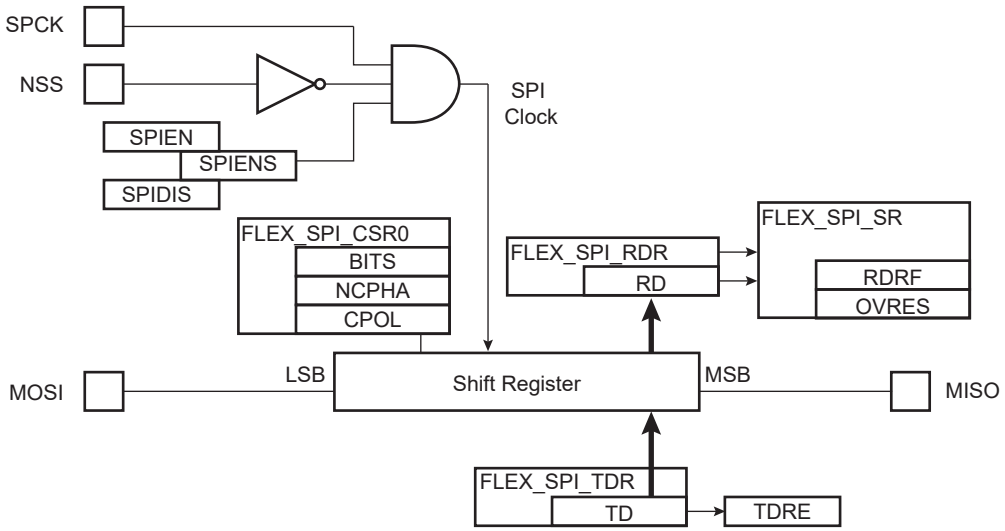
When a first data is written in FLEX_SPI_TDR, it is transferred immediately in the Shift register and the TDRE flag rises. If new data is written, it remains in FLEX_SPI_TDR until a transfer occurs, i.e., NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in FLEX_SPI_TDR is transferred in the Shift register and the TDRE flag rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift register from FLEX_SPI_TDR. If no character is ready to be transmitted, i.e., no character has been written in FLEX_SPI_TDR since the last load from FLEX_SPI_TDR to the Shift register, FLEX_SPI_TDR is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in FLEX_SPI_SR.

If NSS rises between two characters, it must be kept high for two MCK clock periods or more and the next SPCK capture edge must not occur less than four MCK periods after NSS rise.

Figure 44-74 shows a block diagram of the SPI when operating in Slave mode.

**Figure 44-74. Slave Mode Functional Block Diagram**



### 44.8.5 SPI Comparison Function on Received Character

The comparison is only relevant for SPI Slave mode (MSTR = 0 in FLEX_US_MR).

The effect of a comparison match changes if the system is in Wait or Active mode.

In Wait mode, if asynchronous partial wakeup is enabled, a system wakeup is performed (see Section 44.8.6 “SPI Asynchronous and Partial Wakeup (SleepWalking)”).

In Active mode, the CMP flag in FLEX_SPI_SR is raised. It is set when the received character matches the conditions programmed in the SPI Comparison Register (FLEX_SPI_CMPR). The CMP flag is set as soon as FLEX_SPI_RDR is loaded with the new received character. The CMP flag is cleared by reading FLEX_SPI_SR.

FLEX_SPI_CMPR (see Section 44.10.57) can be programmed to provide different comparison methods. These are listed below:

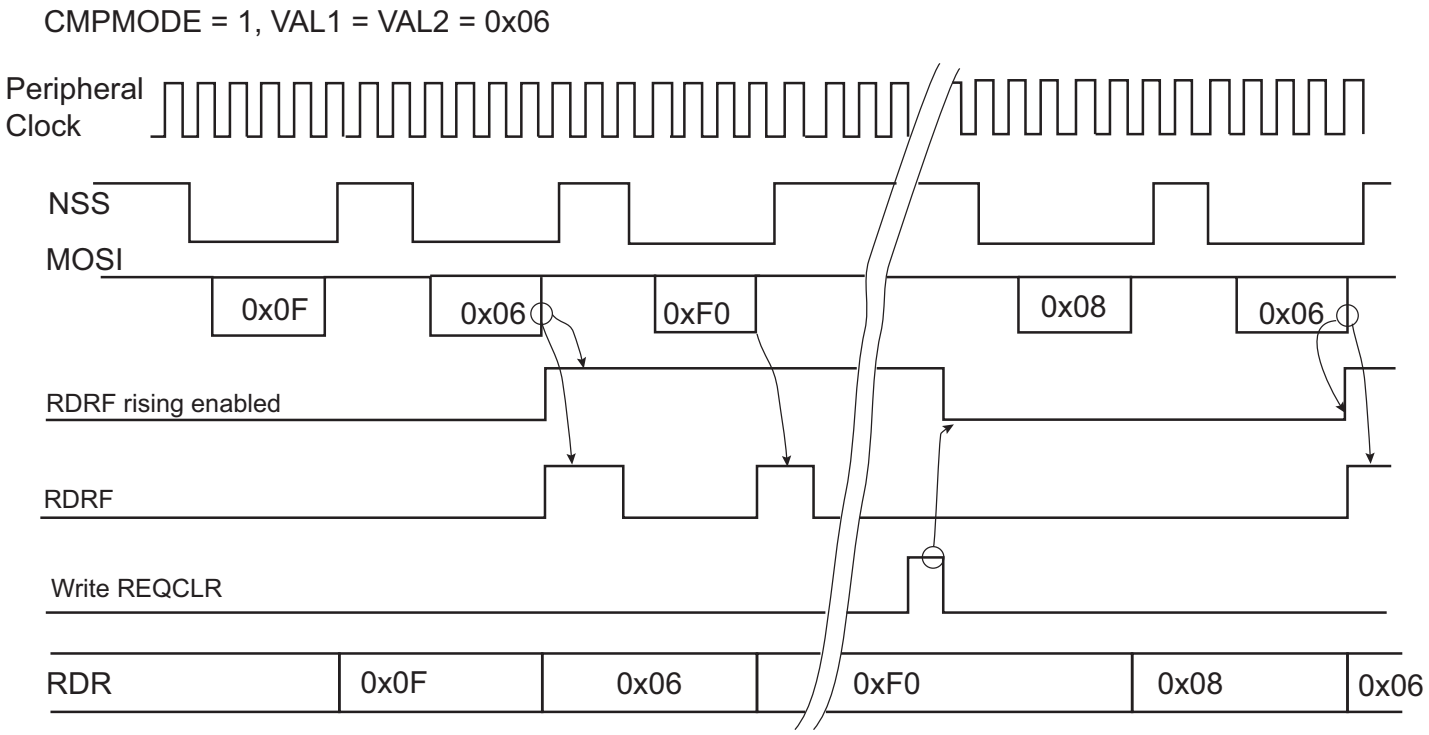
- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to 1 if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the flag CMP is set to 1 if any received character equals VAL1 or VAL2.

When the CMPMODE bit is cleared in FLEX_SPI_CMPR, all received data is loaded in FLEX_SPI_RDR and the CMP flag provides the status of the comparison result.

By setting the CMPMODE bit, the comparison result triggers the start of FLEX_SPI_RDR loading (see Figure 44-75). The trigger condition exists as soon as the received character value matches the conditions defined by VAL1 and VAL2 in FLEX_SPI_CMPR. The comparison trigger event is restarted by writing a 1 to the REQCLR bit in FLEX_SPI_CR.

The value programmed in VAL1 and VAL2 fields must not exceed the maximum value of the received character (see BITS field in FLEX_SPI_CSR0).

**Figure 44-75. Receive Data Register Management**



### 44.8.6 SPI Asynchronous and Partial Wakeup (SleepWalking)

This operating mode is a means of data pre-processing that qualifies an incoming event, thus allowing the SPI to decide whether or not to wake up the system. Asynchronous and partial wakeup is mainly used when the system is in Wait mode (see the PMC section for further details). It can also be enabled when the system is fully running.

Asynchronous and partial wakeup can be used only when SPI is configured in Slave mode (MSTR is cleared in FLEX_SPI_MR).

The maximum SPI clock (SPCK) frequency that can be provided by the SPI master is bounded by the peripheral clock frequency. The SPCK frequency must be lower than or equal to the peripheral clock. The NSS line must be de-asserted by the SPI master between two characters. The NSS de-assertion duration time must be greater than or equal to six peripheral clock periods. The time between the assertion of NSS line (falling edge) and the first edge of the SPI clock must be higher than 15 µs.

The FLEX_SPI_RDR register must be read before enabling the asynchronous and partial wakeup.

When asynchronous and partial wakeup is enabled for the SPI (see the PMC section), the PMC decodes a clock request from the SPI. The request is generated as soon as there is a falling edge on the NSS line as this may indicate the beginning of a frame. If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the SPI.

The SPI processes the received frame and compares the received character with VAL1 and VAL2 in FLEX_SPI_CMPR (Section 44.10.57).

The SPI instructs the PMC to disable the peripheral clock if the received character value does not meet the conditions defined by VAL1 and VAL2 fields in FLEX_SPI_CMPR (see Figure 44-77).

If the received character value meets the conditions, the SPI instructs the PMC to exit the system from Wait mode (see Figure 44-77).

The VAL1 and VAL2 fields can be programmed to provide different comparison methods and thus matching conditions.

- If VAL1 equals VAL2, then the comparison is performed on a single value and the wakeup is triggered if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 wakes up the system.
- If VAL1 is strictly higher than VAL2, the wakeup is triggered if any received character equals VAL1 or VAL2.
- If VAL1 = 0 and VAL2 = 65535, the wakeup is triggered as soon as a character is received.

If the processor and peripherals are running, the SPI can be configured in Asynchronous and Partial Wakeup mode by enabling the PMC_SLPWK_ER (see PMC section). When activity is detected on the receive line, the SPI requests the clock from the PMC and the comparison is performed. If there is a comparison match, the SPI continues to request the clock. If there is no match, the clock is switched off for the SPI only, until a new activity is detected.

The CMPMODE configuration has no effect when Asynchronous and Partial Wakeup mode is enabled for the SPI (see PMC_SLPWK_ER in PMC section).

When the system is in Active mode and the SPI enters Asynchronous and Partial Wakeup mode, the flag RDRF must be programmed as the unique source of the SPI interrupt.

When the system exits Wait mode as the result of a matching condition, the RDRF flag is used to determine if the SPI is the source for the exit from Wait mode.

**Figure 44-76. Asynchronous Wakeup Use Case Example**

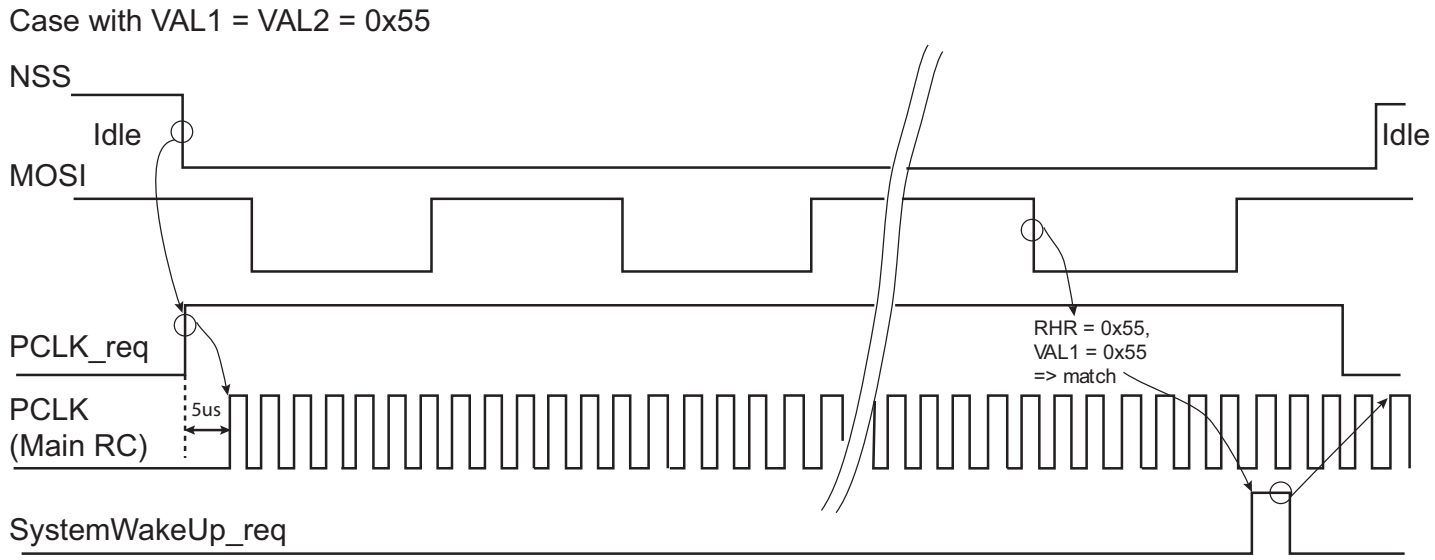
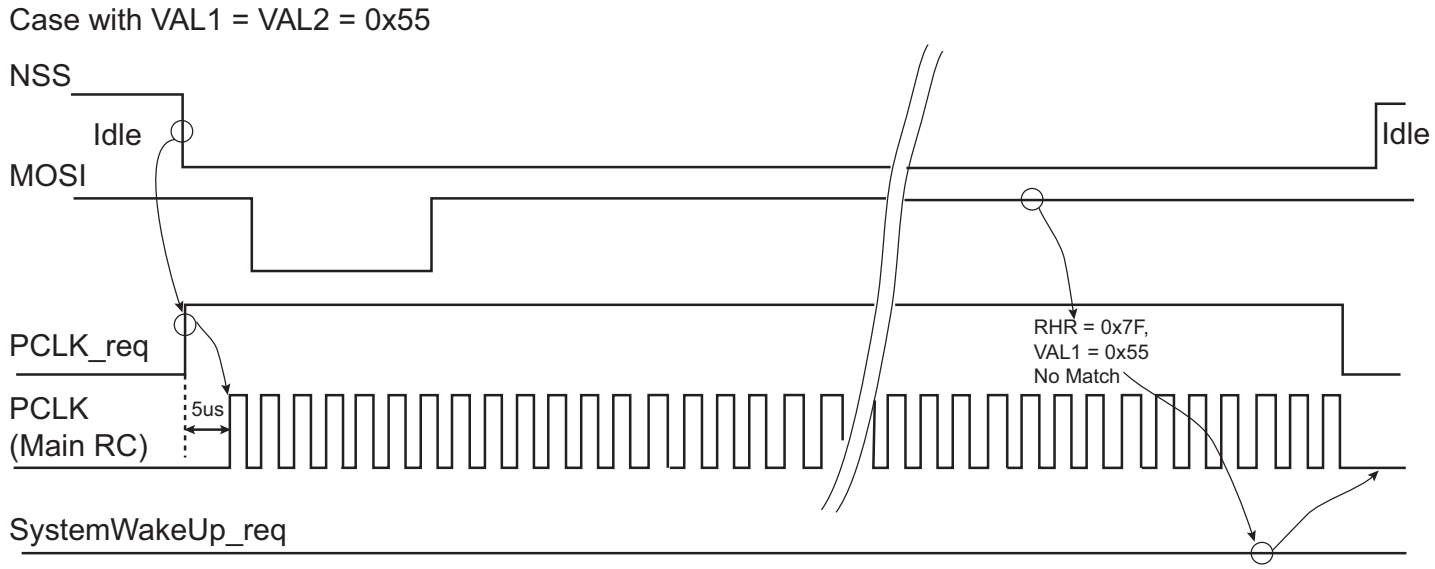



**Figure 44-77. Asynchronous Event Generating Only Partial Wakeup**

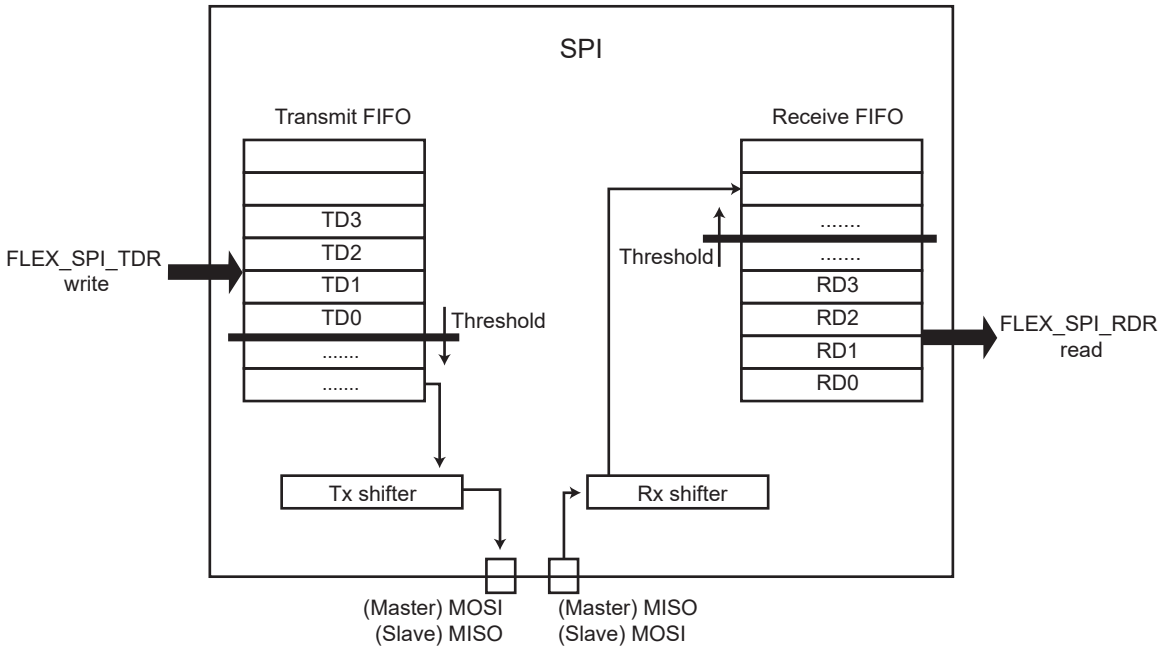



### 44.8.7 FIFOs

The SPI includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in FLEX_SPI_CR. It is recommended to disable the SPI module before enabling or disabling the SPI FIFOs (TXDIS and RXDIS bit in FLEX_SPI_CR).

Writing the FIFOEN bit to ‘1’ enables a 32-data Transmit FIFO and a 32-data Receive FIFO.

**Figure 44-78. FIFOs Block Diagram**

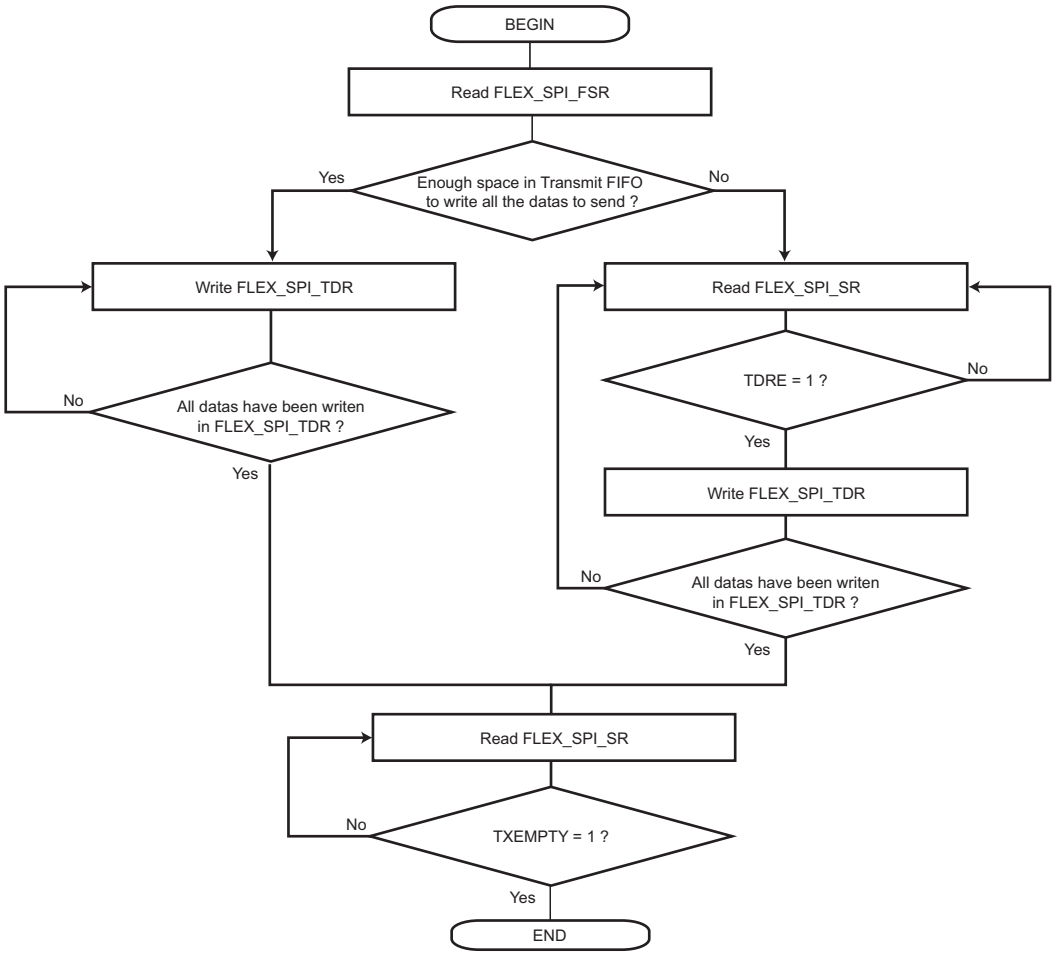



#### 44.8.7.1 Sending Data with FIFO Enabled

With the Transmit FIFO enabled, any write access to FLEX_SPI_TDR brings the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TDRE flag state to send multiple data without DMAC.

Knowing the number of data to send, and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in FLEX_SPI_TDR without checking the TDRE flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the SPI FIFO Level Register (FLEX_SPI_FLR).

**Figure 44-79. Sending Data with FIFO Flowchart**

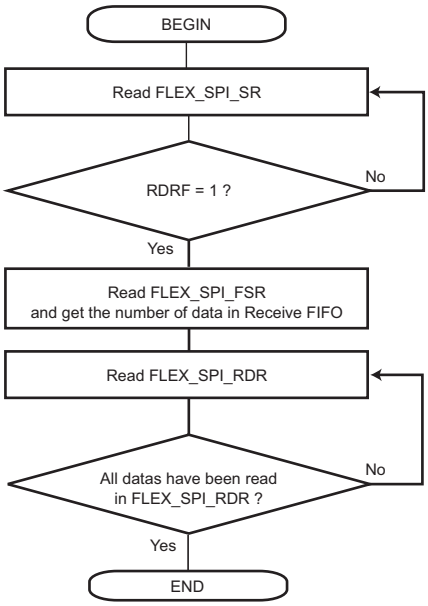



### 44.8.7.2 Receiving Data with FIFO Enabled

With Receive FIFO enabled, any read access on FLEX_SPI_RDR pulls out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RDRF flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RDRF flag set to ‘1’), the exact number of data can be checked with the RXFL field in FLEX_SPI_FLR and all the data read successively in FLEX_SPI_RDR without checking the RDRF flag between each access.

**Figure 44-80. Receiving Data with FIFO Flowchart**




#### 44.8.7.3 Clearing/Flushing FIFOs

Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in FLEX_SPI_CR.

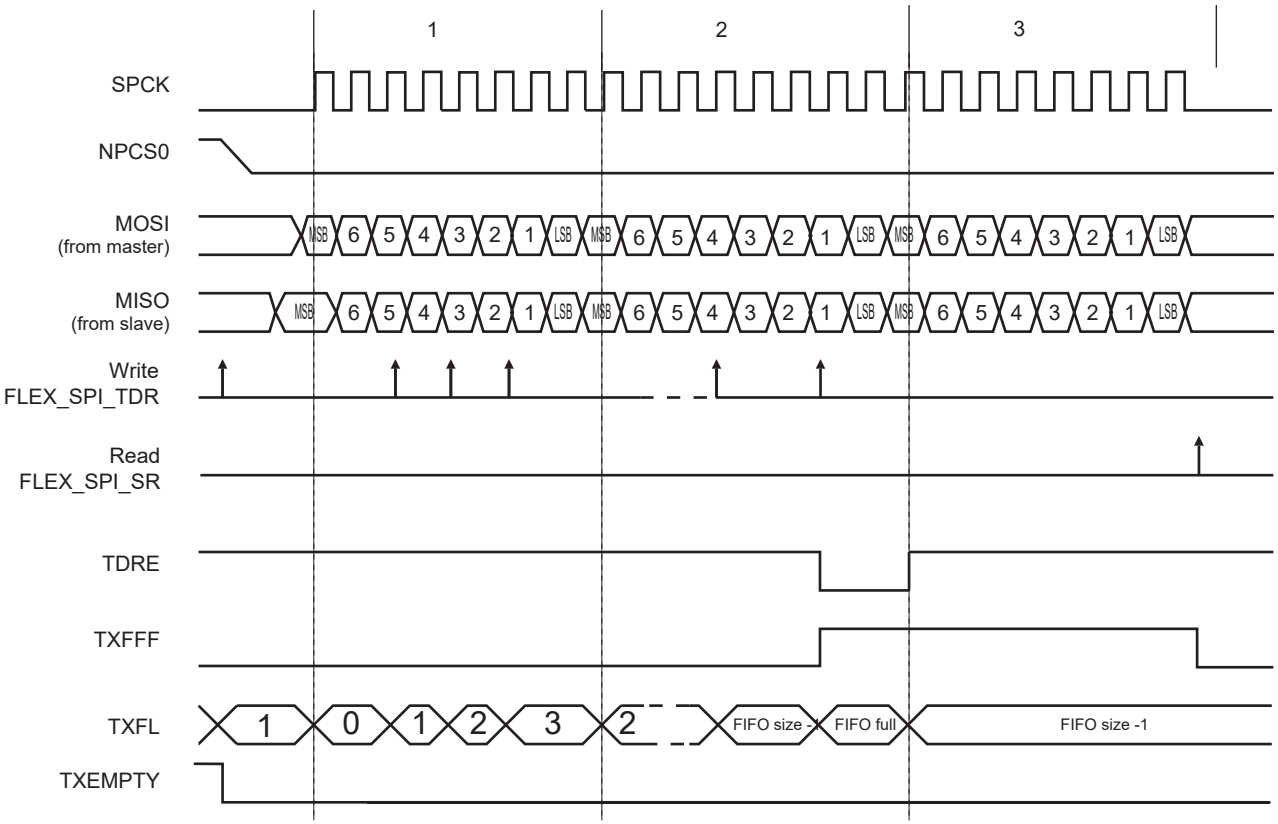
#### 44.8.7.4 TDRE, RDRF and TXEMPTY behavior

If FIFOs are enabled, the behavior of the TDRE, RDRF and TXEMPTY flags is slightly different.

With FIFO enabled, the TXEMPTY flag remains at ‘0’ state as long as there are characters in the Transmit FIFO or in the Transmit Shift Register. It is at ‘1’ state when there are no character in the Transmit FIFO and no character in the Transmit Shift Register.

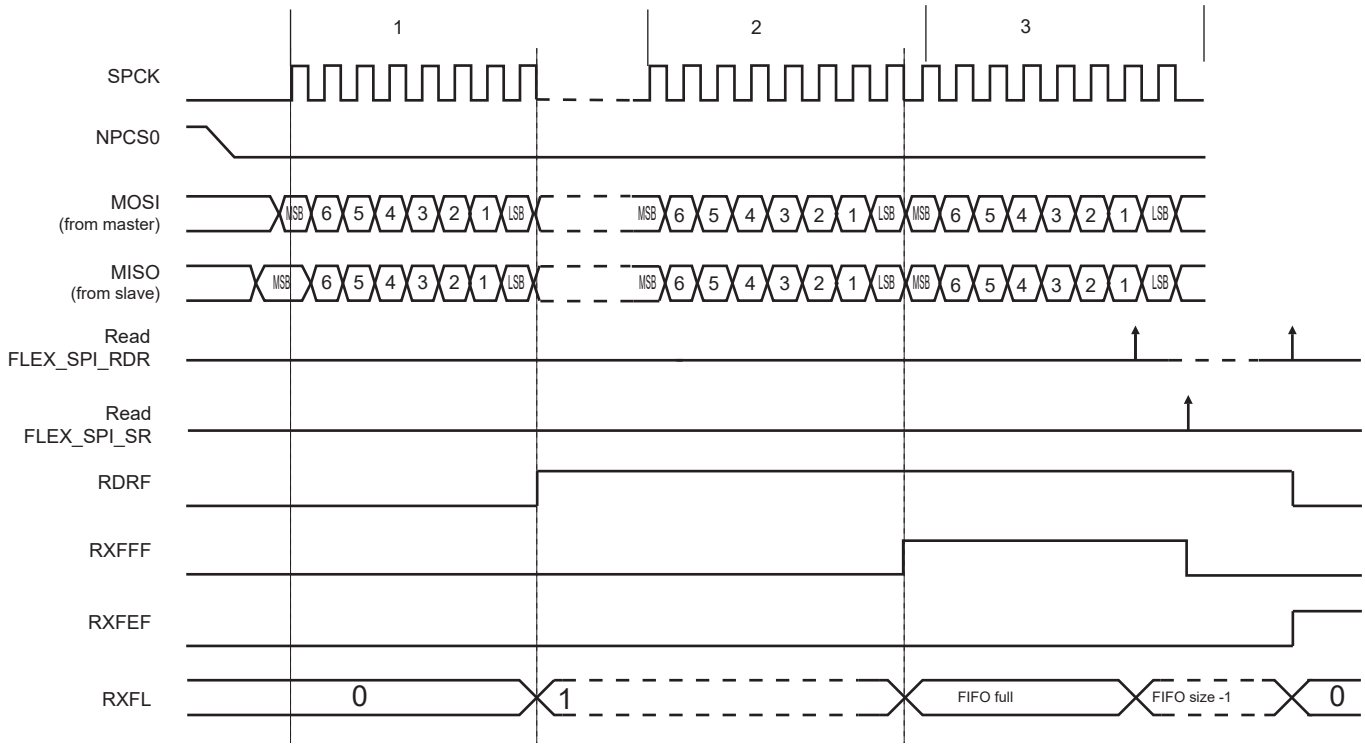
TDRE indicates if a data can be written in the Transmit FIFO. By default, the TDRE flag then stays at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 44-81. TDRE in Single Data Mode and TXRDYM = 0x0**




RDRF indicates if an unread data is present in the Receive FIFO. By default, the RDRF flag is then at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 44-82. RDRF in Single Data Mode and RXRDYM = 0x0**




TDRE and RDRF behavior can be modified using the TXRDYM and RXRDYM fields in the SPI FIFO Mode Register (FLEX_SPI_FMR). In Single Data mode, there is no need to modify the TDRE and RDRF behavior; however, it may be useful in Multiple Data mode.

### 44.8.7.5 Single Data Mode

If Variable Peripheral Select mode is used (PS bit set to ‘1’ in FLEX_SPI_MR), the Transmit FIFO operates in Single Data mode.

If Master mode is set (MSTR bit set to ‘1’ in FLEX_SPI_MR), the Receive FIFO operates in Single Data mode.

In this mode, only one data can be written/read per write/read access of FLEX_SPI_TDR/FLEX_SPI_RDR (see Section 44.10.45 “SPI Receive Data Register”). On the other hand TXRDYM and RXRDYM fields should be configured with 0x0 value in this mode.

**DMAC**

TXRDYM and RXRDYM fields must be configured with 0x0 value when working with DMAC transfer in Single Data mode.

The same DMAC procedure applies as with FIFO disabled.

### 44.8.7.6 Multiple Data Mode

When the FIFOs do not operate in Single Data mode, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write up to two data in one FLEX_SPI_TDR write access. It is possible, in this mode, to read up to four data in one FLEX_SPI_RDR access if the BITS field is configured to 0 (8-bit data size) and up to two data if the BITS field value is other than 0 (more than 8-bit data size).

The number of data to write/read is defined by the size of the register access. If the access is a byte-size register access, only one data is written/read. If the access is a halfword size register access, then up to two data are read/only one data is written and finally, if the access is a word-size register access, then up to four data are read/up to two data are written.

Written/Read data are always right-aligned, as shown in Section 44.10.46 “SPI Receive Data Register (FIFO Multiple Data, 8-bit)”, Section 44.10.47 “SPI Receive Data Register (FIFO Multiple Data, 16-bit)” and Section 44.10.49 “SPI Transmit Data Register (FIFO Multiple Data, 8- to 16-bit)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six FLEX_SPI_TDR-byte write accesses.
- Perform three FLEX_SPI_TDR-halfword write accesses.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six FLEX_SPI_RDR-byte read accesses.
- Perform three FLEX_SPI_RDR-halfword read accesses.
- Perform one FLEX_SPI_RDR-word read access and one FLEX_SPI_RDR-halfword read access.

This mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

#### TDRE and RDRF Configuration

In Multiple Data mode the TXRDYM and RXRDYM fields in FLEX_SPI_FMR become useful.

As in Multiple Data mode, it is possible to write several data in the same access it might be useful to configure TDRE flag behavior to indicate if one or two data can be written in the FIFO depending on the access to perform on FLEX_SPI_TDR.

If, for instance, two data are written each time in FLEX_SPI_TDR, it might be useful to configure the TXRDYM field to 0x1 value so that the TDRE flag is at ‘1’ only when at least two data can be written in the Transmit FIFO.

In the same way, if four data are read each time in FLEX_SPI_RDR, it might be useful to configure the RXRDYM field to 0x2 value so that the RDRF flag is at ‘1’ only when at least four unread data are in the Receive FIFO.

#### DMAC

If DMAC transfer is used, it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

### 44.8.7.7 FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

If the Transmit FIFO is full and a write access is performed on FLEX_SPI_TDR, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in FLEX_SPI_SR.

In Multiple Data mode, if the number of data written in FLEX_SPI_TDR (according to the register access size) is bigger than the Transmit FIFO free space, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in FLEX_SPI_SR.

- Receive FIFO:

In Multiple Data mode, if the number of data read in FLEX_SPI_RDR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it generates a Receive FIFO pointer error and sets the RXFPTEF flag in FLEX_SPI_SR.

No pointer error should happen if the FIFO state is checked before writing/reading in FLEX_SPI_TDR/FLEX_SPI_RDR. The FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a pointer error occurs, a software reset must be performed through the SWRST bit in FLEX_SPI_CR (configuration will be lost).

#### 44.8.7.8 FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

The Transmit FIFO threshold can be set using the TXFTHRES field in FLEX_SPI_FMR. Each time the Transmit FIFO goes from the 'above threshold' to the 'equal or below threshold' state, the TXFTHF flag in FLEX_SPI_SR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

The Receive FIFO threshold can be set using the RXFTHRES field in FLEX_SPI_FMR. Each time the Receive FIFO goes from the 'below threshold' to the 'equal to or above threshold' state, the RXFTHF flag in FLEX_SPI_SR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using FLEX_SPI_IER and FLEX_SPI_IDR.

#### 44.8.7.9 FIFO Flags

FIFOs come with a set of flags which can be configured each to generate interrupt through FLEX_SPI_IER and FLEX_SPI_IDR.

FIFO flags state can be read in FLEX_SPI_SR. They are cleared on FLEX_SPI_SR read.

### 44.8.8 SPI Register Write Protection

The FLEXCOM operating mode (FLEX_MR.OPMODE) must be set to FLEX_MR_OPMODE_SPI to enable access to the write protection registers.

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected by setting the WPEN (Write Protection Enable) bit in the SPI Write Protection Mode Register (FLEX_SPI_WPMR).

If a write access to a write-protected register is detected, the Write Protection Violation Status (WPVS) flag in the SPI Write Protection Status Register (FLEX_SPI_WPSR) is set and the Write Protection Violation Source (WPVSRC) field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading FLEX_SPI_WPSR.

The following register(s) can be write-protected when WPEN is set:

- SPI Mode Register
- SPI Chip Select Register
- SPI Comparison Register

## 44.9 TWI Functional Description

### 44.9.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data are transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see Figure 44-84).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 44-83).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

**Figure 44-83. START and STOP Conditions**

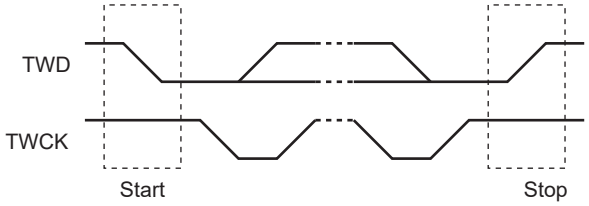
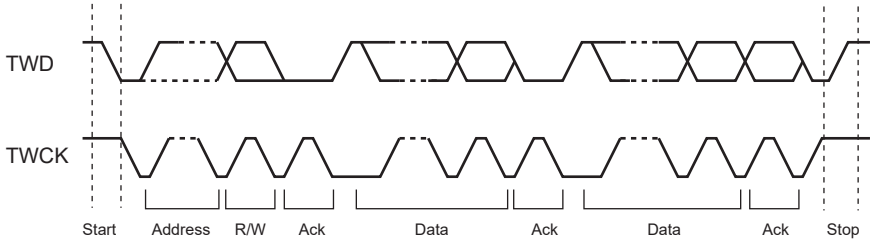



**Figure 44-84. Transfer Format**




### 44.9.2 Modes of Operation

The TWI has different modes of operation:

- Master Transmitter mode (Standard and Fast modes only)
- Master Receiver mode (Standard and Fast modes only)
- Multi-master Transmitter mode (Standard and Fast modes only)
- Multi-master Receiver mode (Standard and Fast modes only)
- Slave Transmitter mode (Standard, Fast and High-speed modes)
- Slave Receiver mode (Standard, Fast and High-speed modes)

These modes are described in the following sections.

### 44.9.3 Master Mode

#### 44.9.3.1 Definition

The master is the device that starts a transfer, generates a clock and stops it. This operating mode is not available if High-speed mode is selected.

#### 44.9.3.2 Programming Master Mode

The following fields must be programmed before entering Master mode:

1. DADR (+ IADRSZ + IADR if a 10-bit device is addressed): The device address is used to access slave devices in Read or Write mode.
2. CWGR + CKDIV + CHDIV + CLDIV: Clock waveform.
3. SVDIS: Disables Slave mode.
4. MSEN: Enables Master mode.

Note: If the TWI is already in Master mode, the device address (DADR) can be configured without disabling the Master mode.

#### 44.9.3.3 Transfer Speed/Bit Rate

The TWI speed is defined in FLEX_TWI_CWGR. The TWI bit rate can be based either on the peripheral clock if the BRSRCCLK bit value is 0 or on a programmable clock source provided by the GCLK if the BRSRCCLK bit value is 1.

If BRSRCCLK = 1, the bit rate is independent of the processor/peripheral clock and thus processor/peripheral clock frequency can be changed without affecting the TWI transfer rate.

The GCLK frequency must be at least three times lower than the peripheral clock frequency.

#### 44.9.3.4 Master Transmitter Mode

This operating mode is not available if High-speed mode is selected.

After the master initiates a START condition when writing into the Transmit Holding register FLEX_TWI_THR, it sends a 7-bit slave address, configured in the Master Mode Register (DADR in FLEX_TWI_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in FLEX_TWI_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (ninth pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. If the slave does not acknowledge the byte, then the Not Acknowledge flag (NACK) is set in the TWI Status Register (FLEX_TWI_SR) of the master and a STOP condition is sent. The NACK flag must be cleared by reading the TWI Status Register (FLEX_TWI_SR) before the next write into the TWI Transmit Holding Register (FLEX_TWI_THR). As with the other status bits, an interrupt can be generated if enabled in the interrupt enable Register (FLEX_TWI_IER). If the slave acknowledges the byte, the data written in FLEX_TWI_THR is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in FLEX_TWI_THR.

TXRDY is used as transmit ready for the DMA transmit channel.

Note: To clear the TXRDY flag in Master mode, write bit MSDIS to 1, then write bit MSEN to 1 in FLEX_TWI_CR.

While no new data is written in FLEX_TWI_THR, the serial clock line is tied low. When new data is written in FLEX_TWI_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of the TWI Control Register (FLEX_TWI_CR).

After a master write transfer, the Serial Clock line is stretched (tied low) while no new data is written in FLEX_TWI_THR or until a STOP command is performed.

See Figure 44-85, Figure 44-86, and Figure 44-87.

**Figure 44-85. Master Write with One Data Byte**

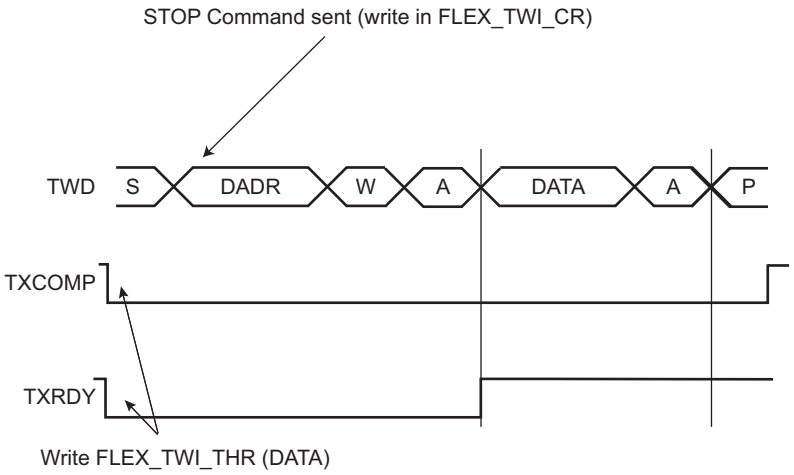


**Figure 44-86. Master Write with Multiple Data Bytes**

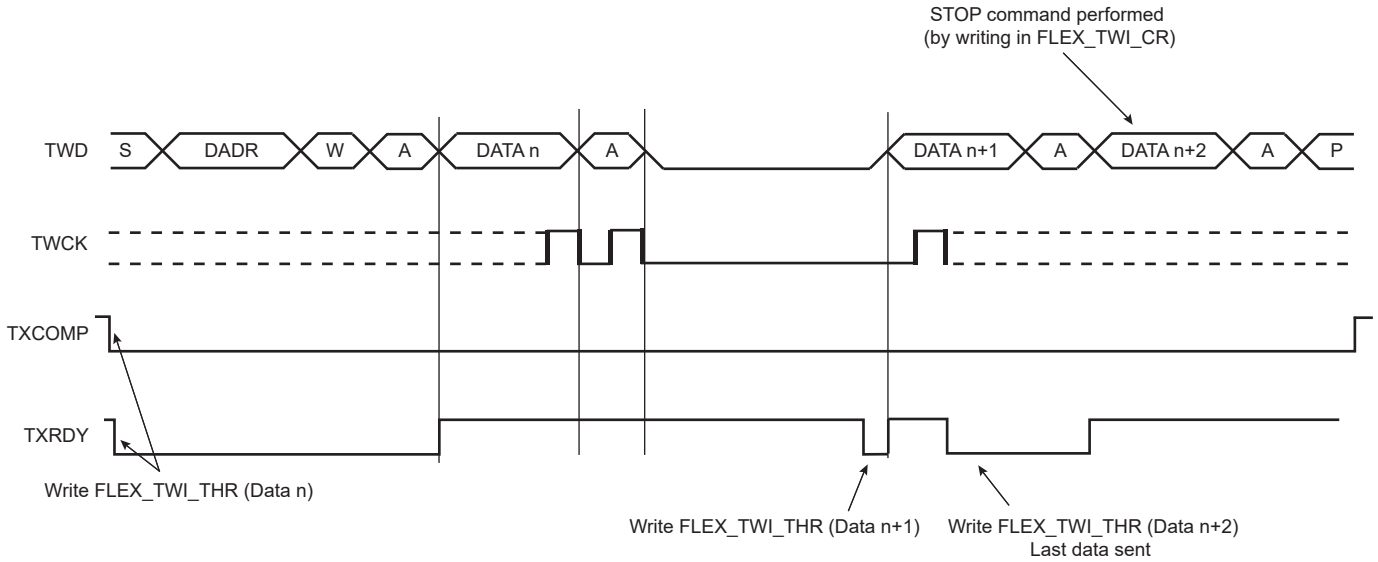
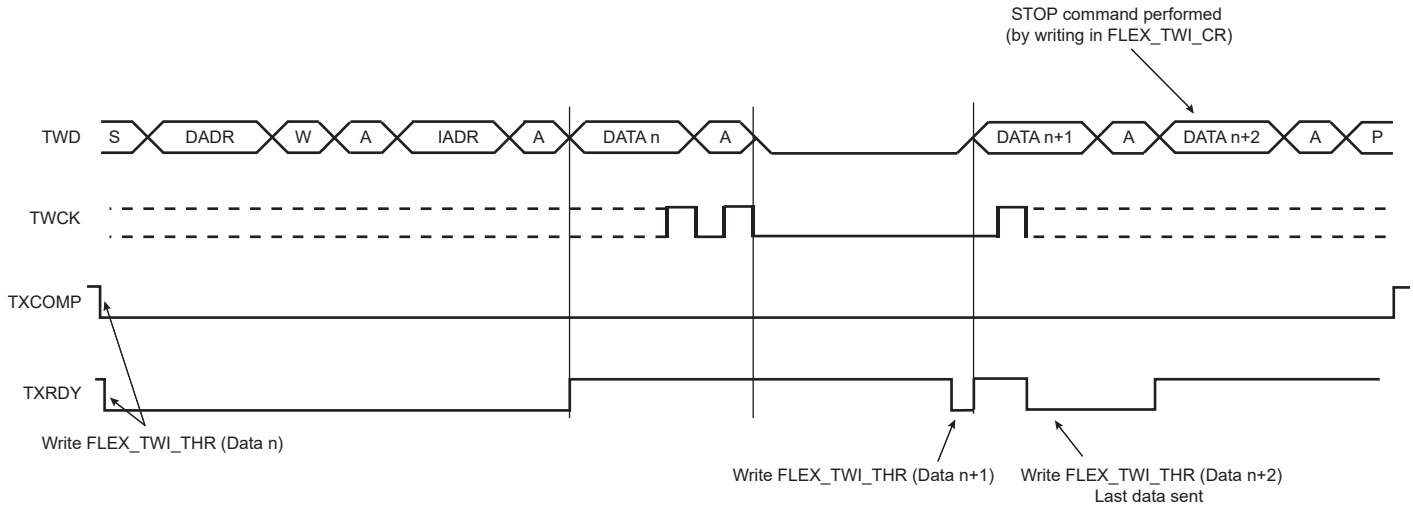



**Figure 44-87. Master Write with One Byte Internal Address and Multiple Data Bytes**




### 44.9.3.5 Master Receiver Mode

Master Receiver mode is not available if High-speed mode is selected.

The read sequence begins by setting the START bit. After the start condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction, 1 in this case (MREAD = 1 in FLEX_TWI_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the NACK bit in FLEX_TWI_SR if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data (see Figure 44-88). When the RXRDY bit is set in FLEX_TWI_SR, a character has been received in the Receive Holding Register (FLEX_TWI_RHR). The RXRDY bit is reset when reading FLEX_TWI_RHR.

When a single data byte read is performed, with or without internal address **(**IADR), the START and STOP bits must be set at the same time. See Figure 44-88. When a multiple data byte read is performed, with or without

internal address **(**IADR), the STOP bit must be set after the next-to-last data received (same condition applies for START bit to generate a repeated start). See Figure 44-89. For internal address usage, see Section 44.9.3.6 “Internal Address”.

If FLEX_TWI_RHR is full (RXRDY high) and the master is receiving data, the serial clock line will be tied low before receiving the last bit of the data and until FLEX_TWI_RHR is read. Once FLEX_TWI_RHR is read, the master will stop stretching the serial clock line and end the data reception. See Figure 44-90.

**Warning:** When receiving multiple bytes in Master Read mode, if the next-to-last access is not read (the RXRDY flag remains high), the last access will not be completed until FLEX_TWI_RHR is read. The last access stops on the next-to-last bit (clock stretching). When FLEX_TWI_RHR is read there is only half a bit period to send the STOP bit (or START bit) command, else another read access might occur (spurious access).

A possible workaround is to set the STOP bit (or START bit) before reading FLEX_TWI_RHR on the next-to-last access (within IT handler).

**Figure 44-88. Master Read with One Data Byte**

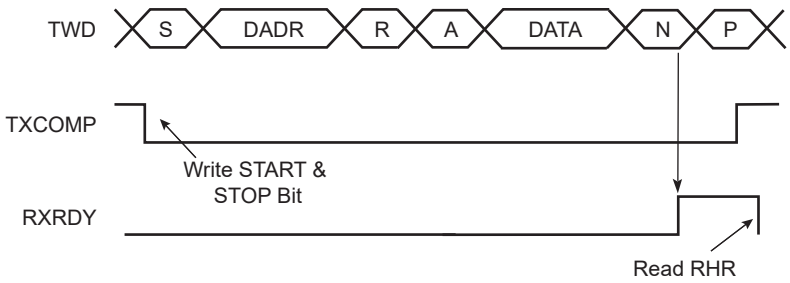



**Figure 44-89. Master Read with Multiple Data Bytes**

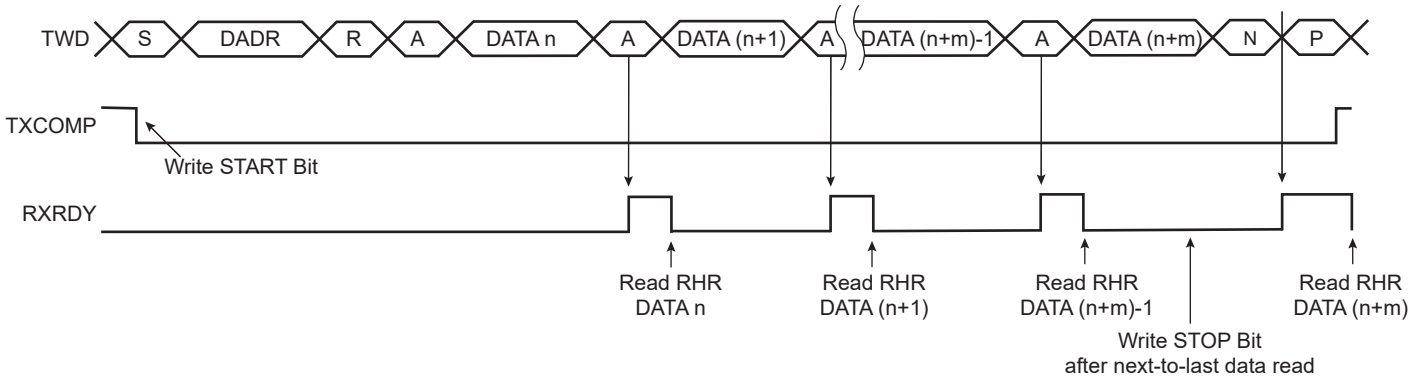
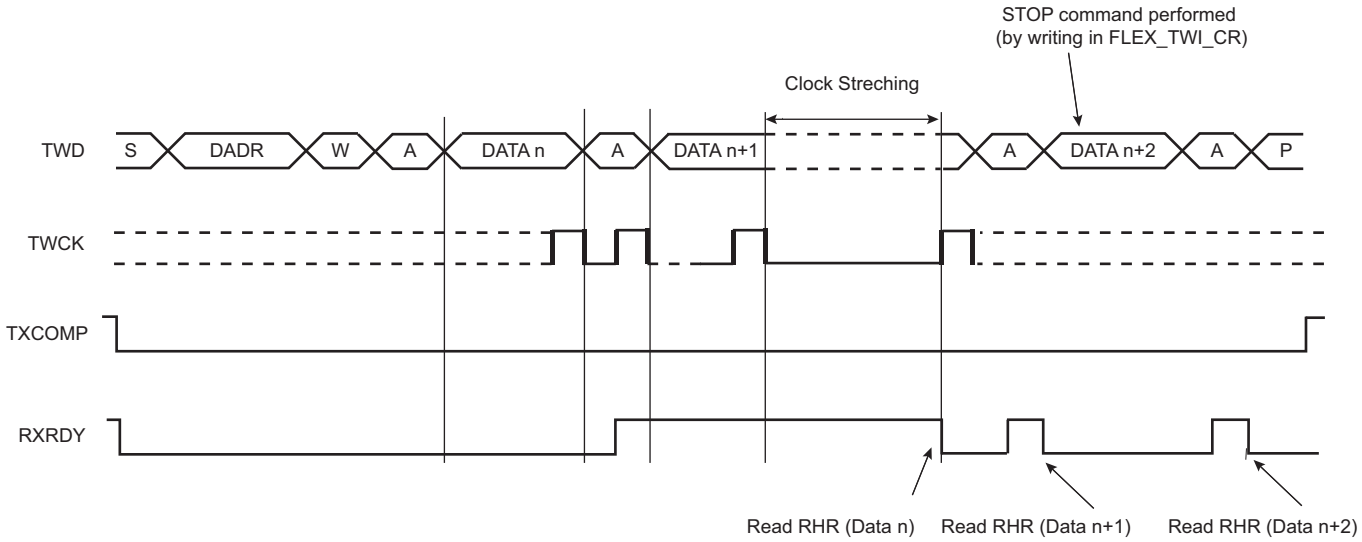


**Figure 44-90. Master Read Clock Stretching with Multiple Data Bytes**




RXRDY is used as receive ready trigger event for the DMA receive channel.

#### 44.9.3.6 Internal Address

The TWI interface can perform transfers with 7-bit slave address devices and with 10-bit slave address devices.

**7-bit Slave Addressing**

When addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, e.g., within a memory page location in a serial memory. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second start condition (after sending the IADR) is sometimes called “repeated start” (Sr) in $I^2C$ fully-compatible devices. See Figure 44-92.

See Figure 44-91 and Figure 44-93 for the master write operation with internal address.

The three internal address bytes are configurable through the Master Mode Register (FLEX_TWI_MMR).

If the slave device supports only a 7-bit address, i.e., no internal address, IADRSZ must be configured to 0.

The abbreviations listed below are used in Figure 44-91 and Figure 44-92:

| | |
|---|---|
| S | Start |
| Sr | Repeated Start |
| P | Stop |
| W | Write |
| R | Read |
| A | Acknowledge |
| N | Not Acknowledge |
| DADR | Device Address |
| IADR | Internal Address |

**Figure 44-91. Master Write with One, Two or Three Bytes Internal Address and One Data Byte**

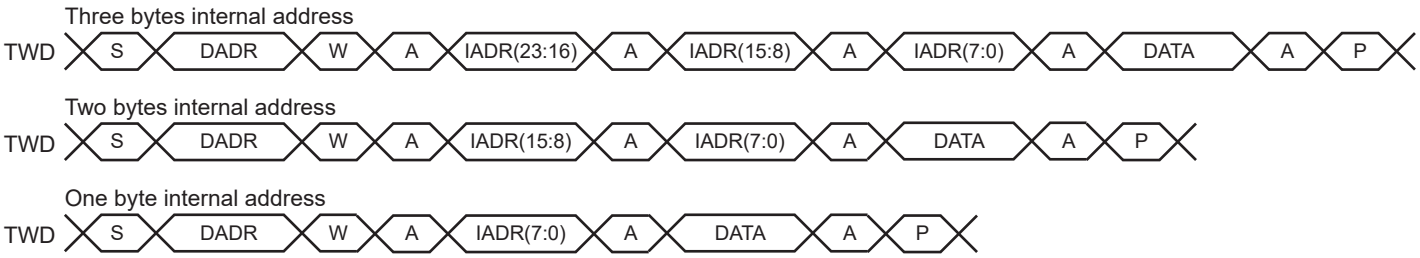
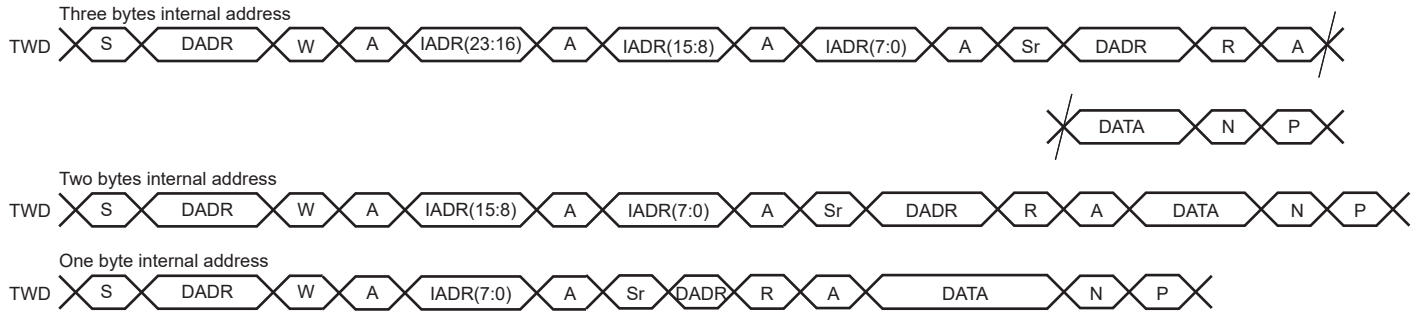



**Figure 44-92. Master Read with One, Two or Three Bytes Internal Address and One Data Byte**




### 10-bit Slave Addressing

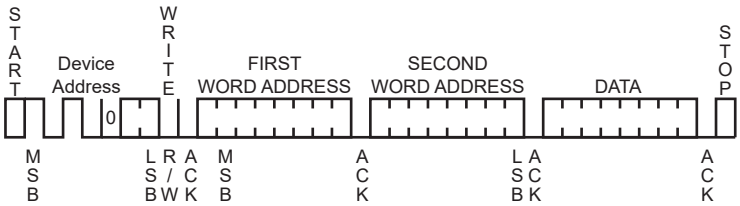
For a slave address higher than seven bits, the user must configure the address size (IADRSZ) and set the other slave address bits in the Internal Address Register (FLEX_TWI_IADR). The two remaining internal address bytes, IADR[15:8] and IADR[23:16], can be used the same way as in 7-bit slave addressing.

**Example:** Address a 10-bit device (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program FLEX_TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 44-93 shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

**Figure 44-93. Internal Address Usage**




### 44.9.3.7 Repeated Start

In addition to Internal Address mode, repeated start (Sr) can be generated manually by writing the START bit at the end of a transfer instead of the STOP bit. In such case the parameters of the next transfer (direction, SADR, etc.) will need to be set before writing the START bit at the end of the previous transfer.

See Section 44.9.3.13 for detailed flowcharts.

Note that generating a repeated start after a single data read is not supported.

#### 44.9.3.8 Bus Clear Command

The TWI interface can perform a Bus Clear Command:

1. Configure the Master mode (DADR, CKDIV, etc).
2. Start the transfer by setting the CLEAR bit in FLEX_TWI_CR.

Note: If an alternative command is used (ACMEN bit = 1), the DATAL field must be cleared.

#### 44.9.3.9 SMBus Mode

SMBus mode is enabled when the SMEN bit is written to one in FLEX_TWI_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

1. Only 7-bit addressing can be used.
2. The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into FLEX_TWI_SMBTR.
3. Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).
4. A set of addresses has been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring FLEX_TWI_CR appropriately.

**Packet Error Checking**

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in FLEX_TWI_CR to one enables automatic PEC handling in the current transfer. Transfers with and without PEC can freely be intermixed in the same system, since some slaves may not support PEC. The PEC LFSR is always updated on every bit transmitted or received, so that PEC handling on combined transfers will be correct.

In Master Transmitter mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave will compare it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave will return an ACK to the master. If the PEC values differ, data was corrupted, and the slave will return a NACK value. Some slaves may not be able to check the received PEC in time to return a NACK if an error occurred. In this case, the slave should always return an ACK after the PEC byte, and some other mechanism must be implemented to verify that the transmission was received correctly.

In Master Receiver mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master will compare it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the PECERR bit in FLEX_TWI_SR is set. In Master Receiver mode, the PEC byte is always followed by a NACK transmitted by the master, since it is the last byte in the transfer.

In combined transfers, the PECRQ bit should only be set in the last of the combined transfers. If Alternative Command mode is enabled, only the NPEC bit should be set.

Consider the following transfer:

S, ADR+W, COMMAND_BYTE, ACK, SR, ADR+R, DATA_BYTE, ACK, PEC_BYTE, NACK, P

See Section 44.9.3.13 for detailed flowcharts.

*Timeouts*

The TLOWS and TLOWM fields in FLEX_TWI_SMBTR configure the SMBus timeout values. If a timeout occurs, the master will transmit a STOP condition and leave the bus. The TOUT bit is also set in FLEX_TWI_SR.

#### 44.9.3.10 SMBus Quick Command (Master Mode Only)

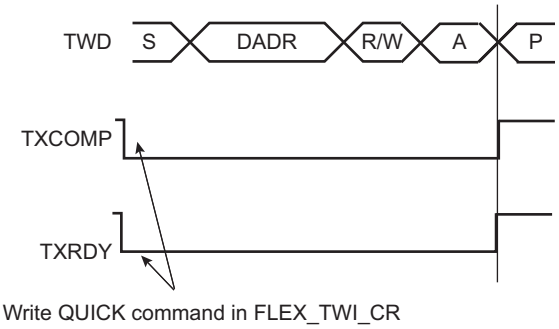
The TWI interface can perform a quick command:

1. Configure the Master mode (DADR, CKDIV, etc).
2. Write the MREAD bit in FLEX_TWI_MMR at the value of the one-bit command to be sent.

3. Start the transfer by setting the QUICK bit in FLEX_TWI_CR.

Note: If an alternative command is used (ACMEN bit = 1) DATAL field must be cleared.

**Figure 44-94. SMBus Quick Command**




#### 44.9.3.11 Alternative Command

Another way to configure the transfer is to enable the Alternative Command mode with the ACMEN bit of the TWI Control Register.

In this mode, the transfer is configured through the TWI Alternative Command Register. It is possible to define a simple read or write transfer or a combined transfer with a repeated start.

In order to set a simple transfer, the DATAL field and the DIR field of the TWI Alternative Command Register must be filled accordingly and the NDATAL field must be cleared. To begin the transfer, either set the START bit in the TWI Control Register in case of a read transfer, or write the TWI Transmit Holding Register in case of a write transfer.

For a combined transfer linked by a repeated start, the NDATAL field must be filled with the length of the second transfer and NDIR with the corresponding direction.

The PEC and NPEC bits are used to set a PEC field. In the case of a single transfer with PEC, the PEC bit must be set. In the case of a combined transfer, the NPEC bit must be set.

Note: If the Alternative Command mode is used, IADRSZ in TWIHS_MMR must be set to 0.

See Section 44.9.3.13 for detailed flowcharts.

#### 44.9.3.12 Handling Errors in Alternative Command

In case of NACK generated by a slave device or SMBus timeout error, the TWI stops immediately the frame but the DMA transfer may still be active. To prevent a new frame to be restarted with remaining DMA data (transmit), the TWI prevents any start of frame until the LOCK flag is cleared in FLEX_TWI_SR.

The LOCK bit in FLEX_TWI_SR indicates the state of the TWI (locked or not locked).

When the TWI is locked, no transfer will begin until the LOCK is cleared using the LOCKCLR bit in FLEX_TWI_CR and error flags cleared reading FLEX_TWI_SR.

In case of error, FLEX_TWI_THR may have been loaded with a new data. The THRCLR bit in FLEX_TWI_CR can be used to flush FLEX_TWI_THR. If the THRCLR bit is set, the TXRDY and TXCOMP flags are set.

#### 44.9.3.13 Read/Write Flowcharts

The following flowcharts shown in Figure 44-96, Figure 44-97, Figure 44-102, Figure 44-103 and Figure 44-104 give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (FLEX_TWI_IER) be configured first.

**Figure 44-95. TWI Write Operation with Single Data Byte without Internal Address**

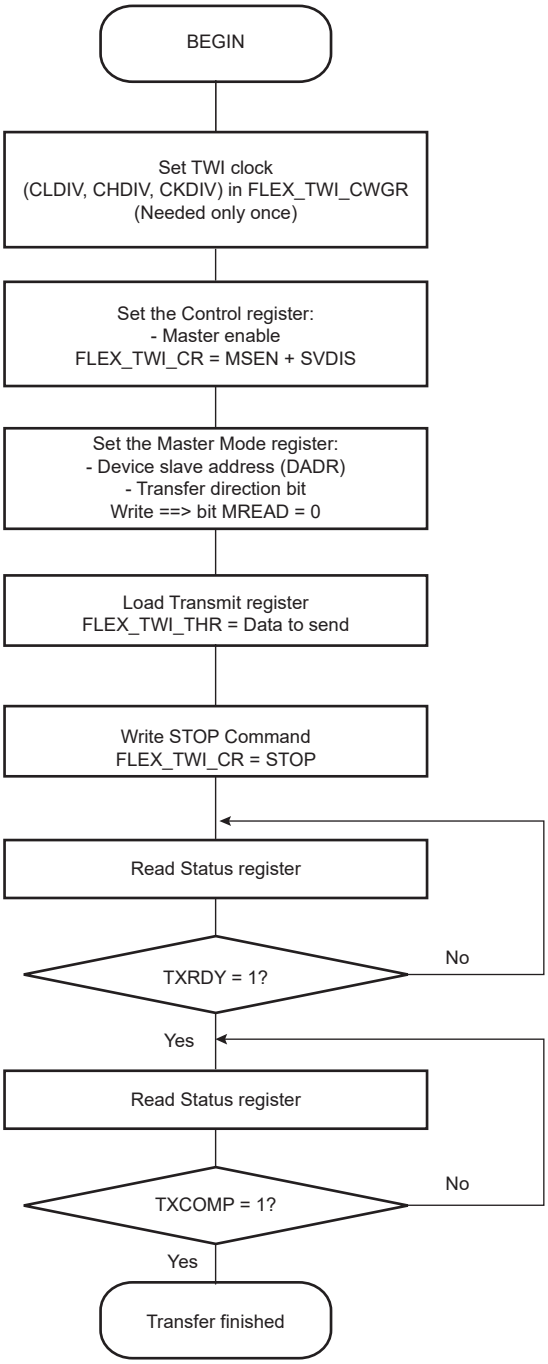


**Figure 44-96. TWI Write Operation with Single Data Byte and Internal Address**

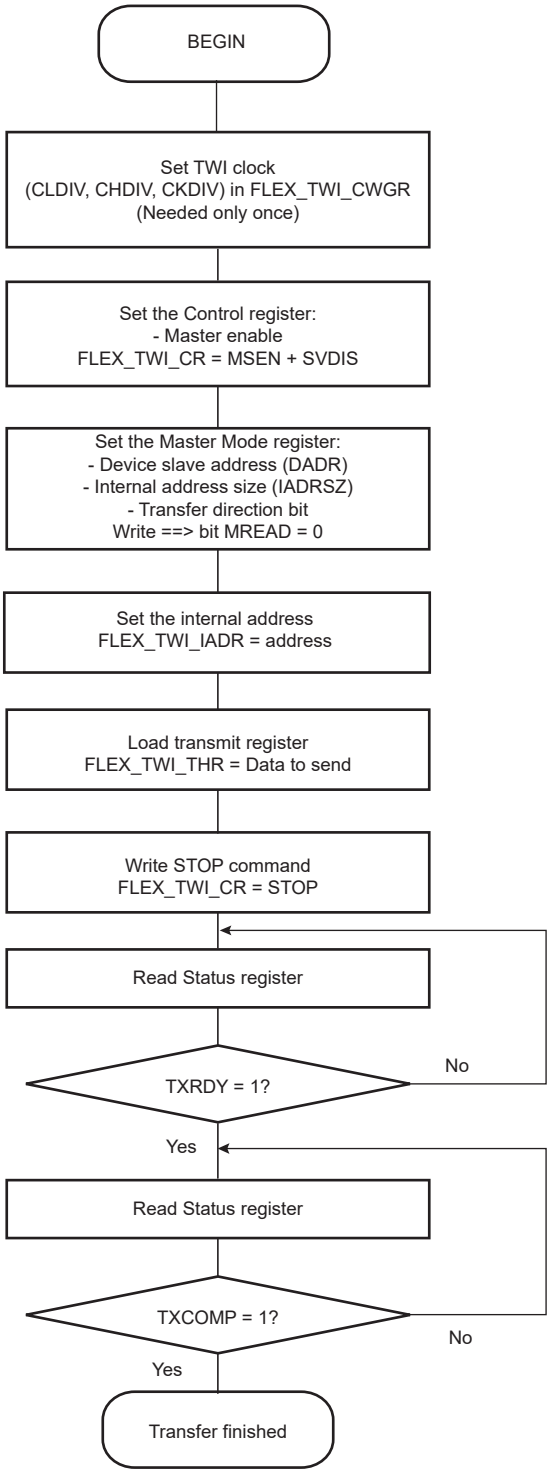


**Figure 44-97. TWI Write Operation with Multiple Data Bytes with or without Internal Address**

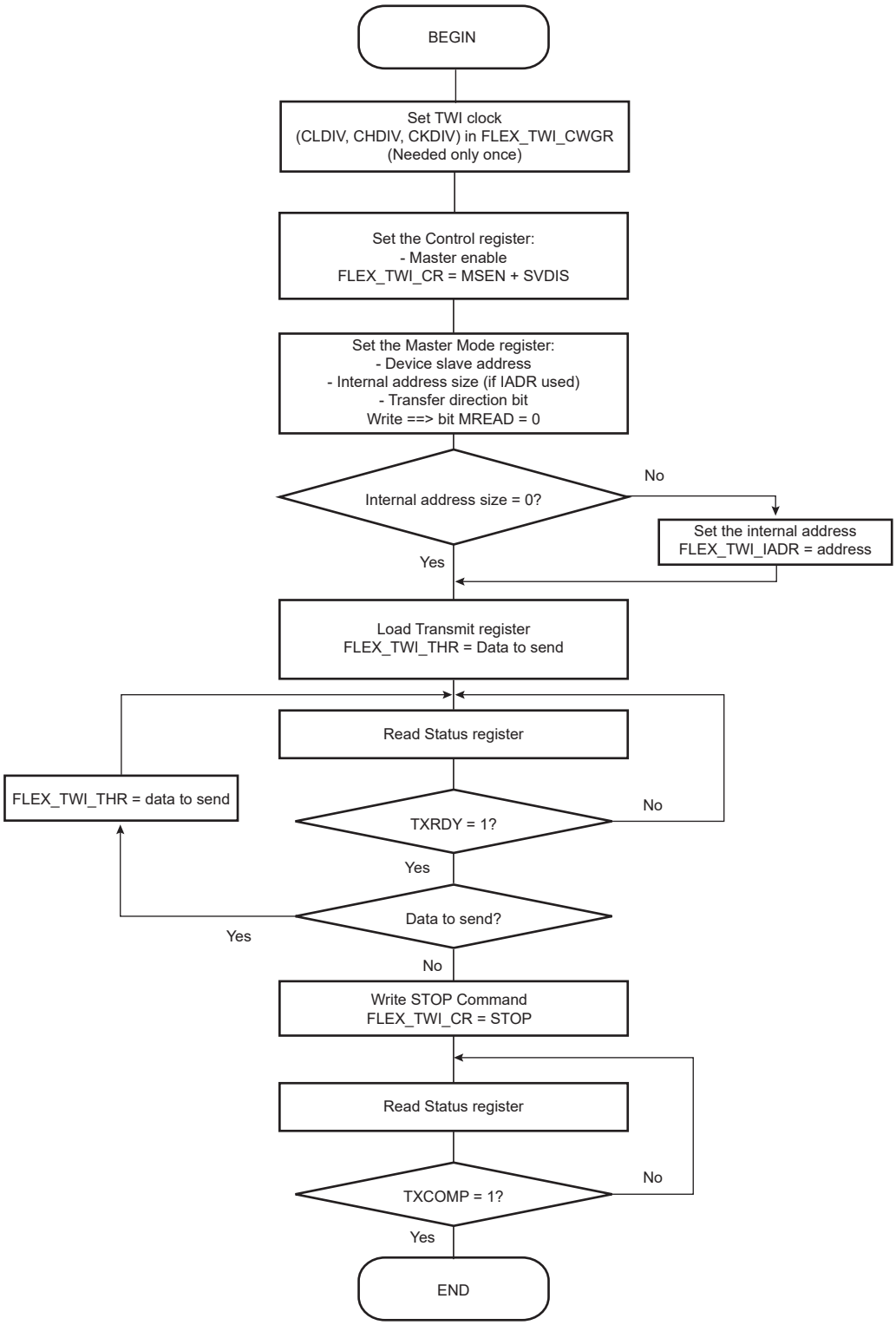


**Figure 44-98. SMBus Write Operation with Multiple Data Bytes with or without Internal Address and PEC Sending**

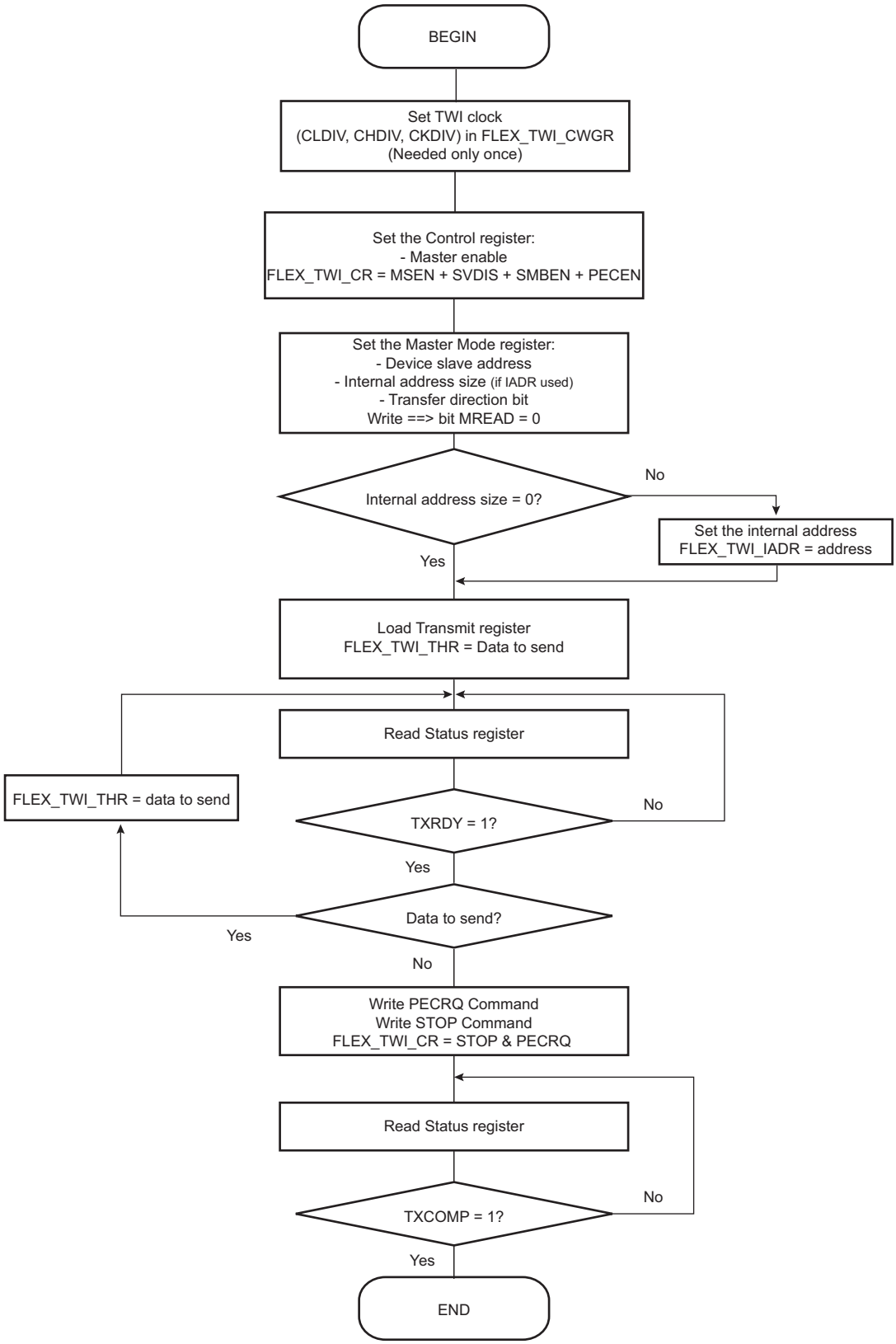


**Figure 44-99. SMBus Write Operation with Multiple Data Bytes with PEC and Alternative Command Mode**

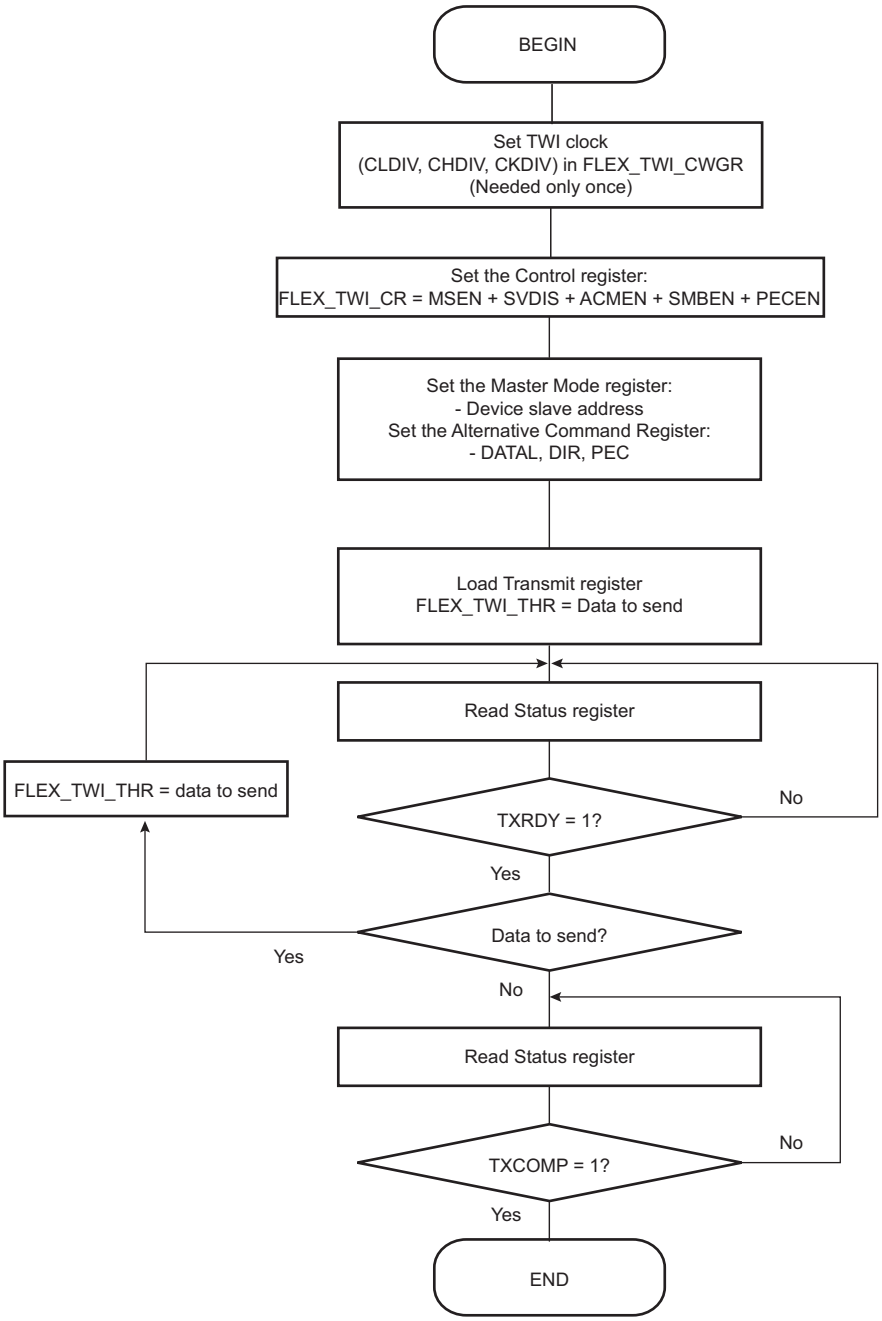


**Figure 44-100. TWI Write Operation with Multiple Data Bytes and Read Operation with Multiple Data Bytes (Sr)**

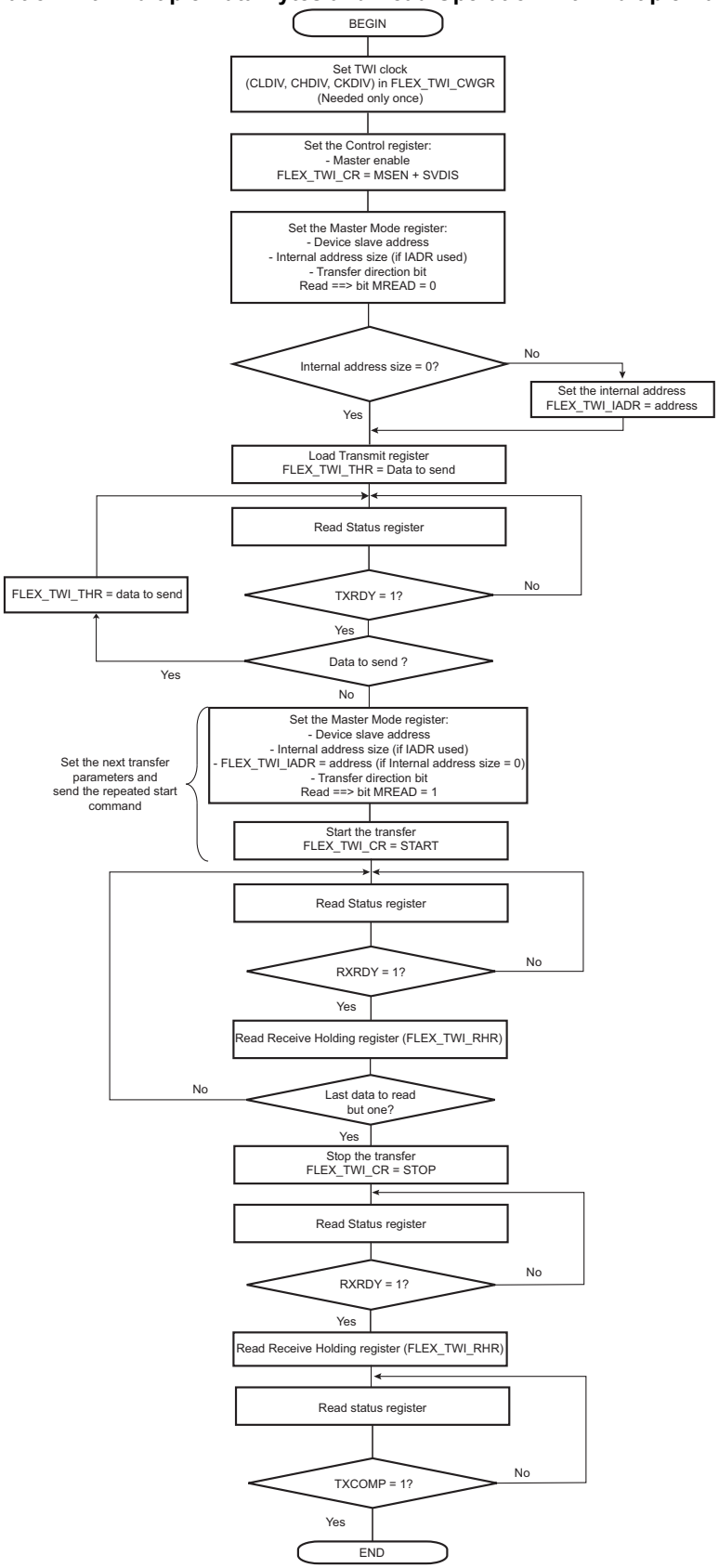


**Figure 44-101. TWI Write Operation with Multiple Data Bytes + Read Operation and Alternative Command Mode + PEC**

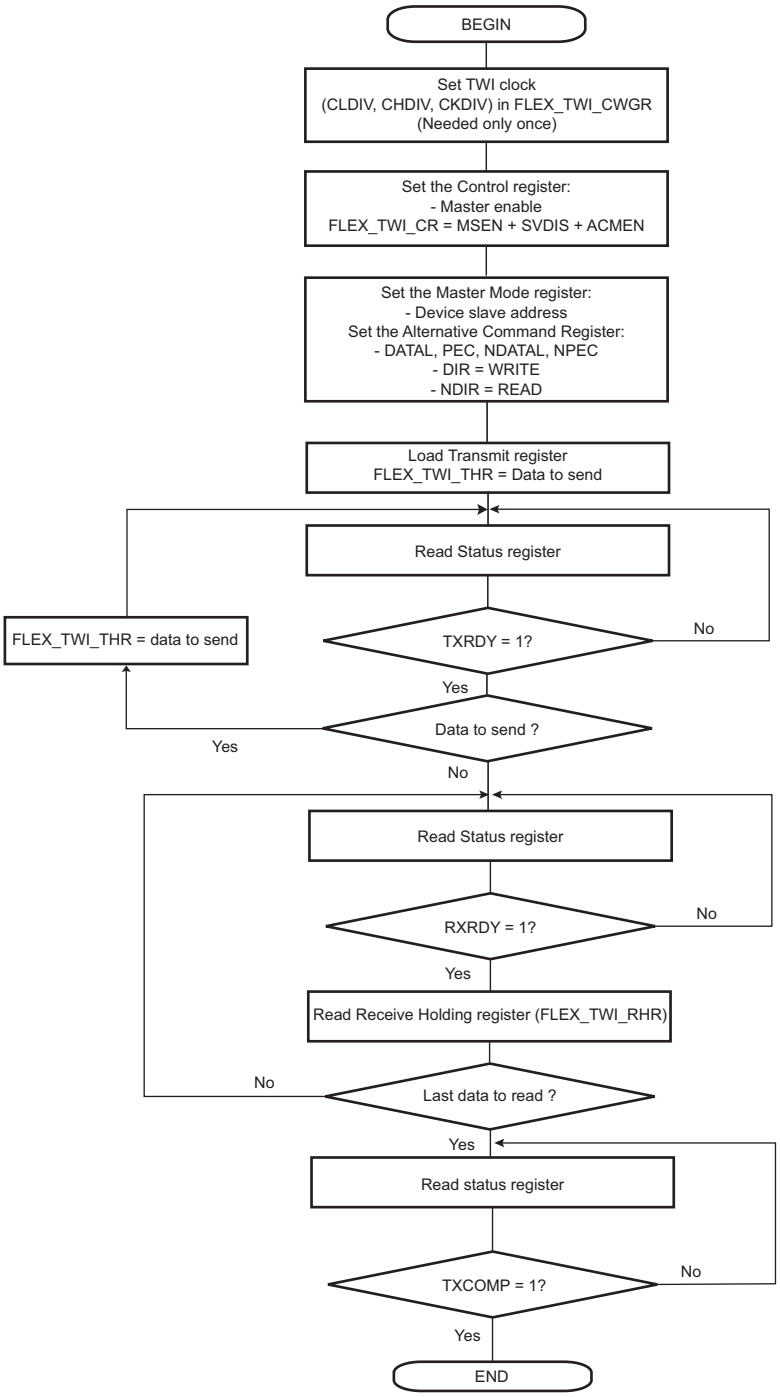


**Figure 44-102. TWI Read Operation with Single Data Byte without Internal Address**

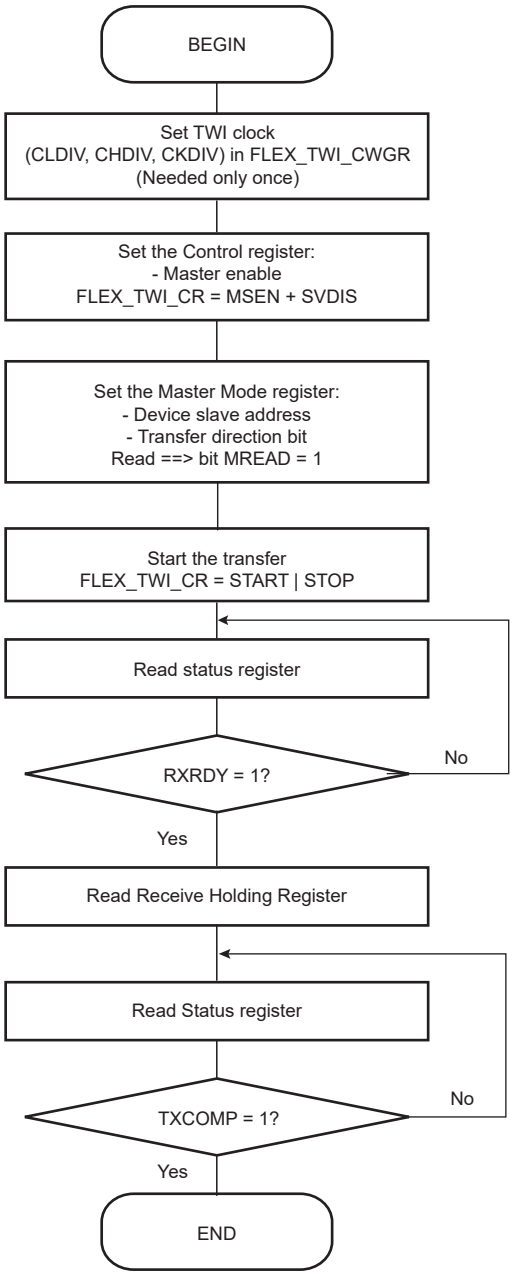


**Figure 44-103. TWI Read Operation with Single Data Byte and Internal Address**

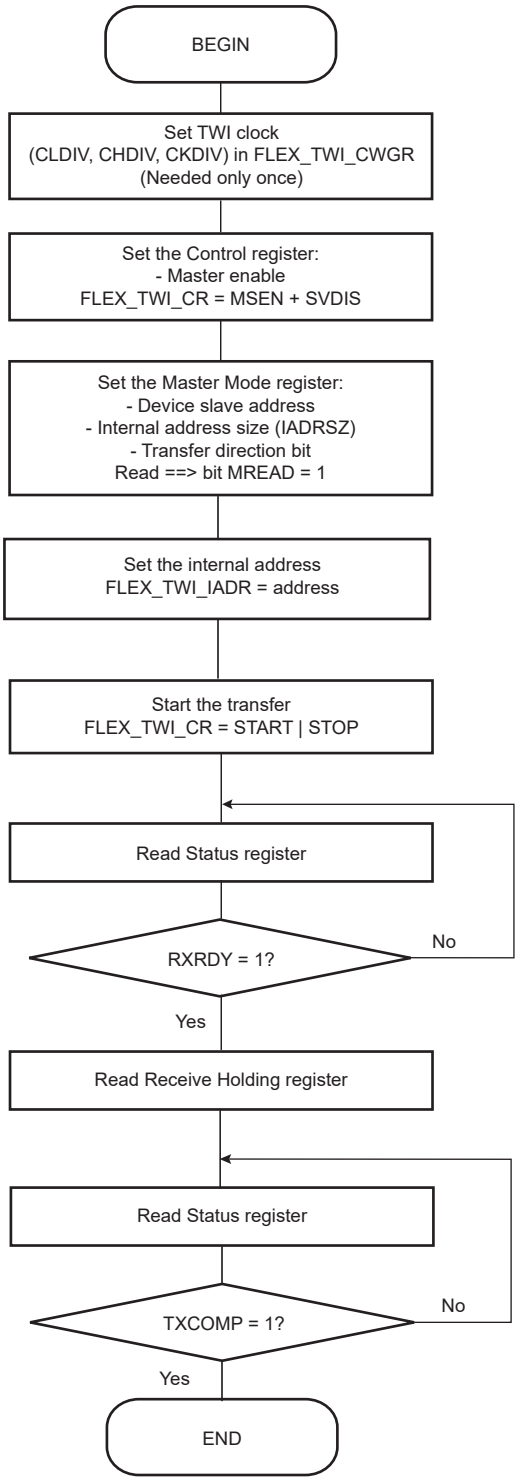


**Figure 44-104. TWI Read Operation with Multiple Data Bytes with or without Internal Address**

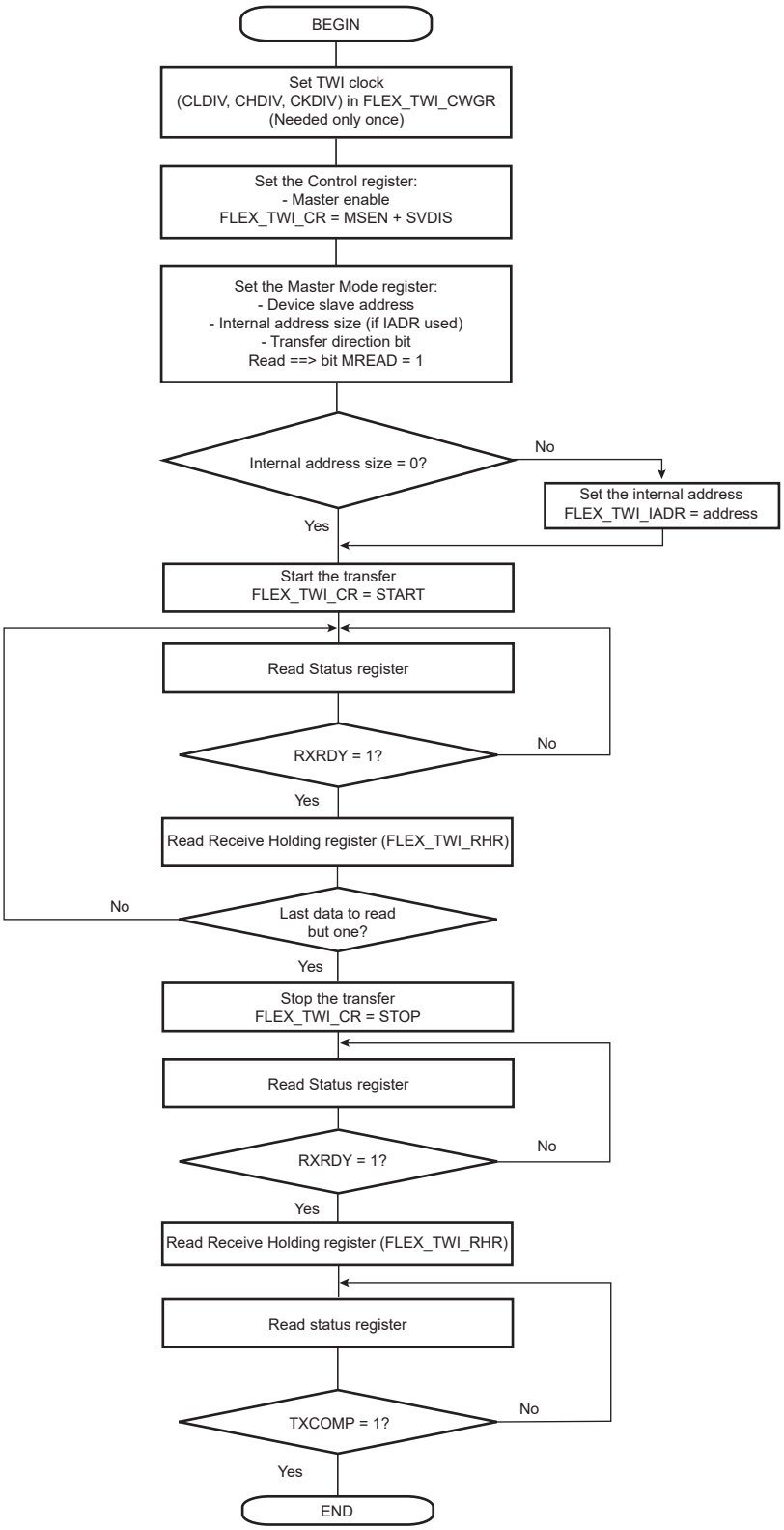


**Figure 44-105. TWI Read Operation with Multiple Data Bytes with or without Internal Address with PEC**

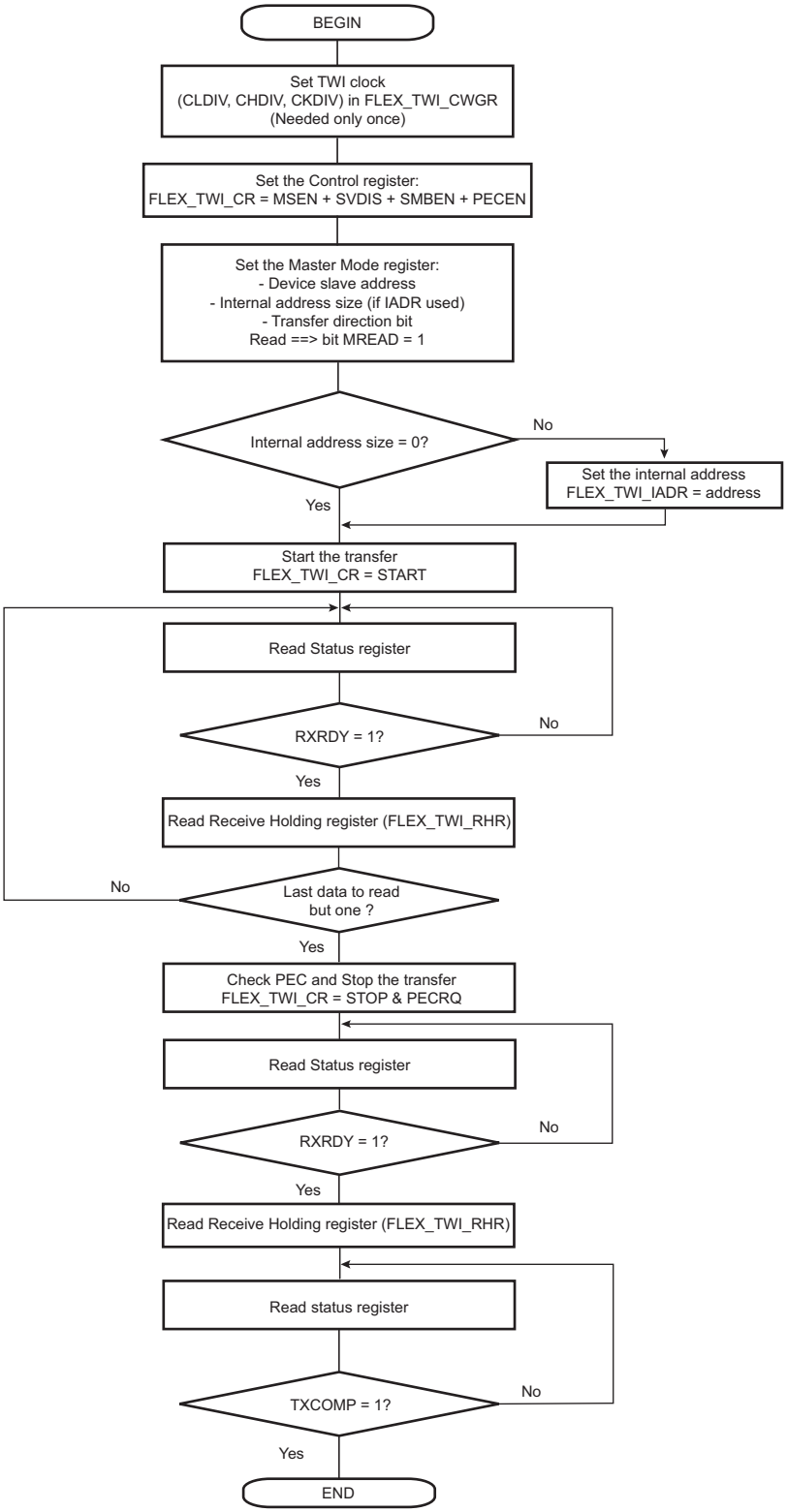


**Figure 44-106. TWI Read Operation with Multiple Data Bytes with Alternative Command Mode with PEC**

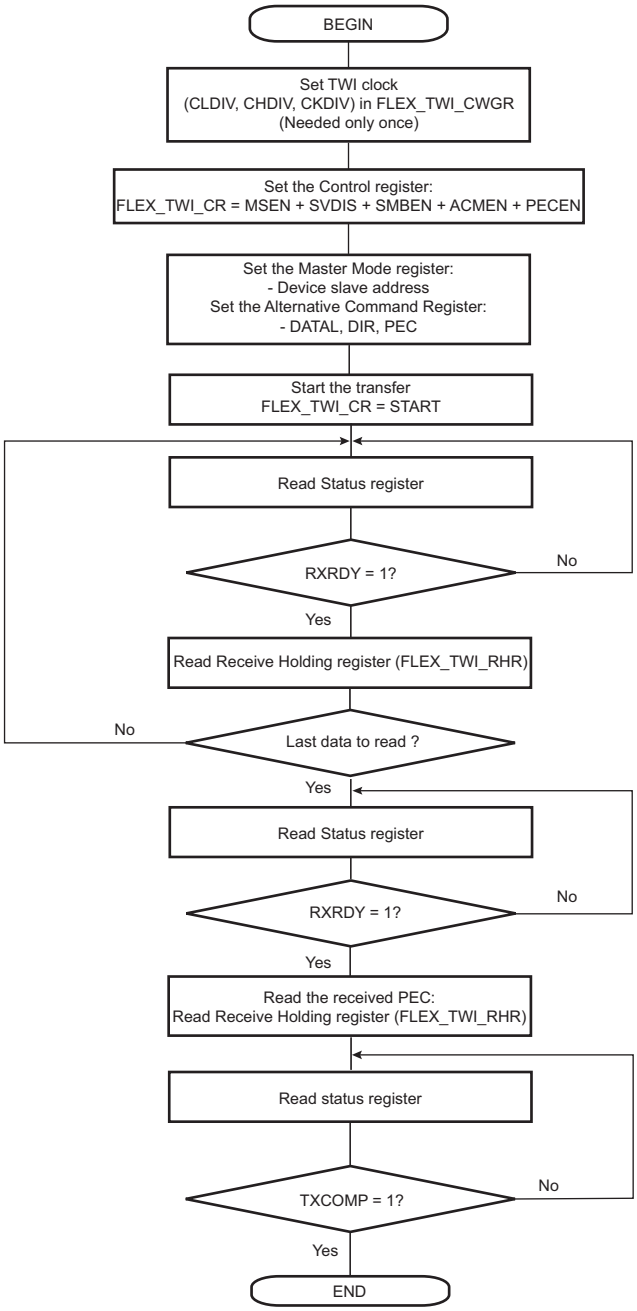


**Figure 44-107. TWI Read Operation with Multiple Data Bytes + Write Operation with Multiple Data Bytes (Sr)**

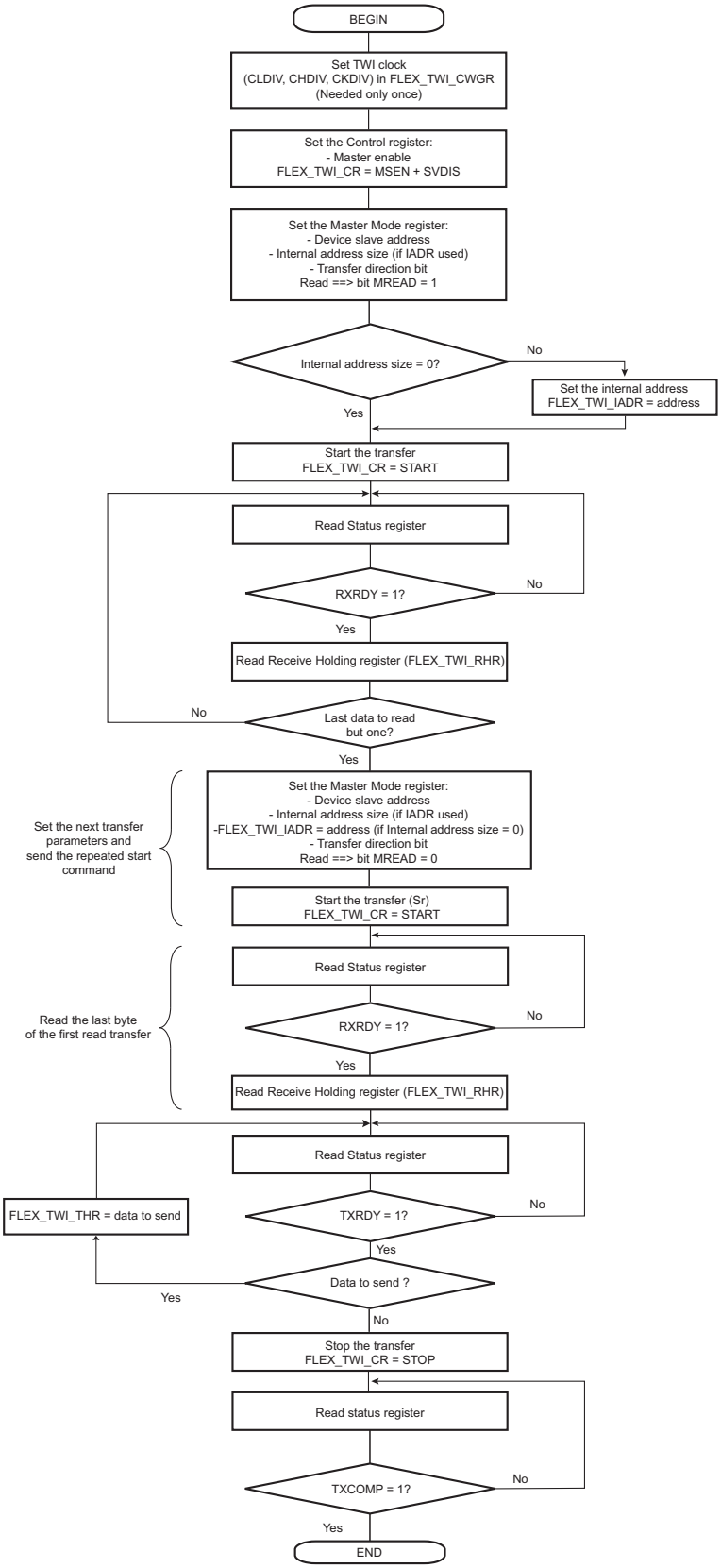


**Figure 44-108. TWI Read Operation with Multiple Data Bytes + Write with Alternative Command Mode with PEC**

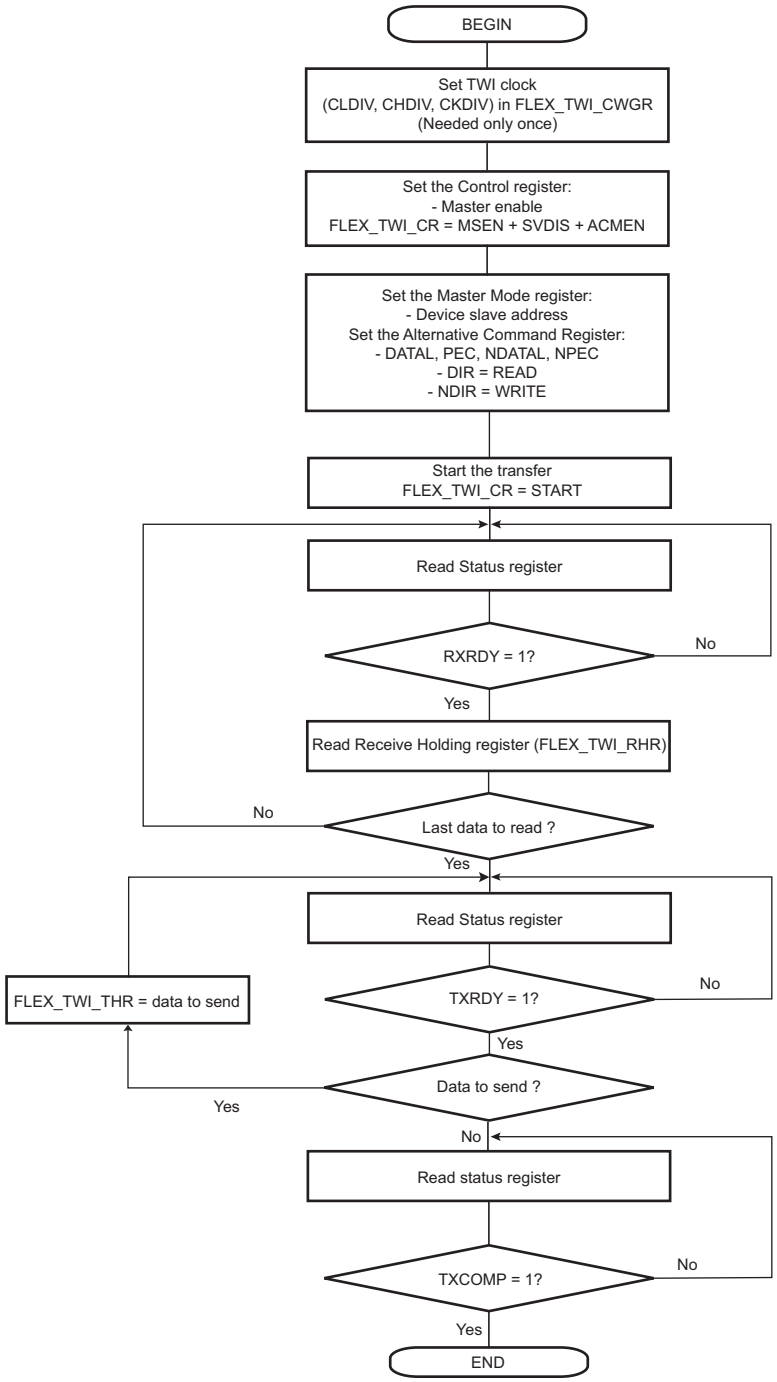
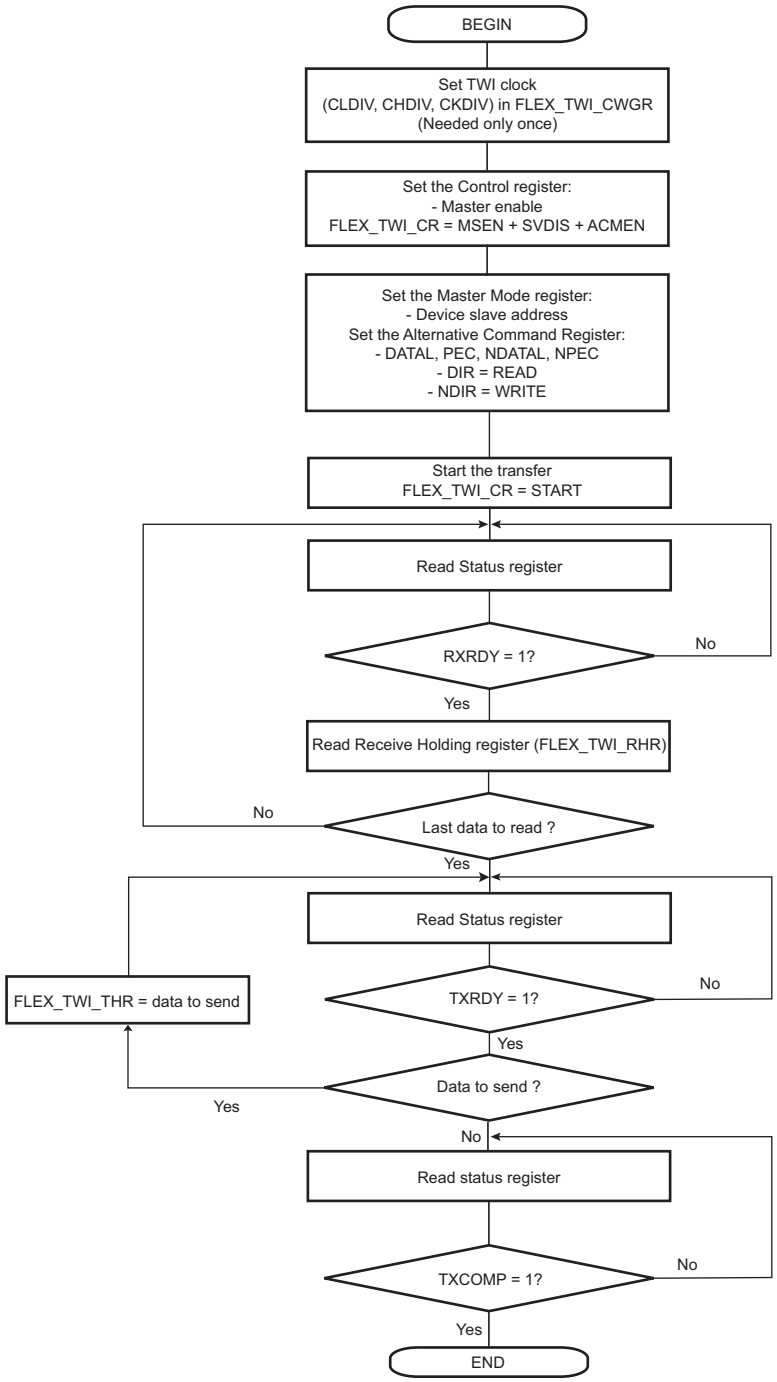


**Figure 44-109. TWI Read Operation with Multiple Data Bytes + Write with Alternative Command Mode with PEC**

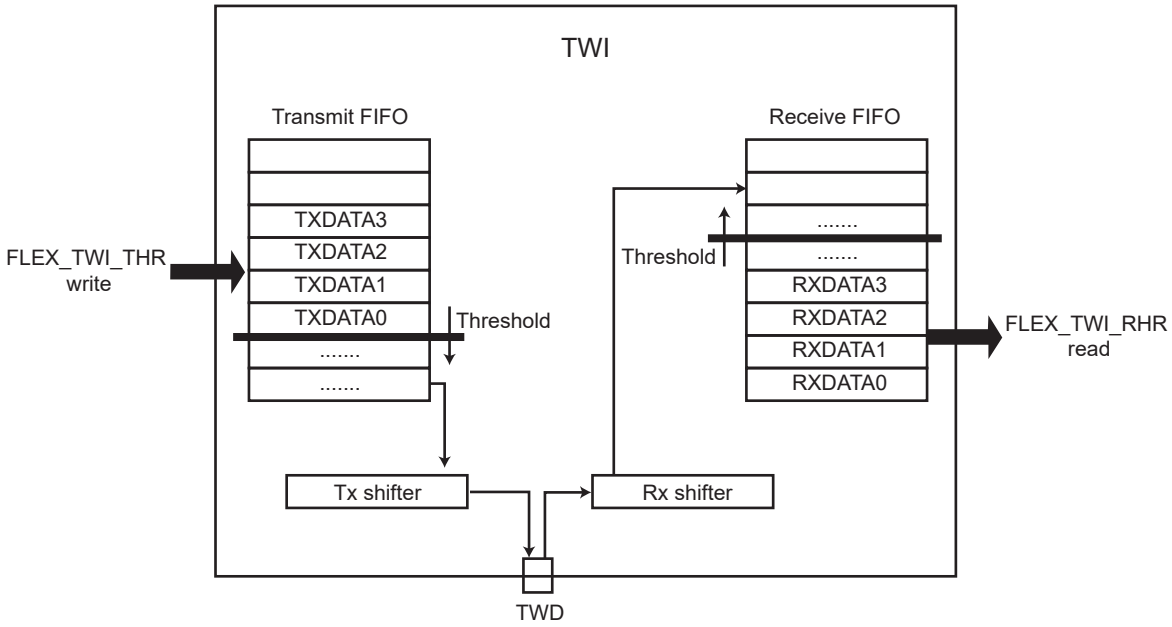



### 44.9.3.14 FIFOs

The TWI includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in FLEX_TWI_CR. It is recommended to disable both Master and Slave modes before enabling or disabling the FIFO (MSDIS and SVDIS bit in FLEX_TWI_CR).

Writing the FIFOEN bit to ‘1’ will enable a 16-data Transmit FIFO and a 16-data Receive FIFO.

**Figure 44-110. FIFOs Block Diagram**

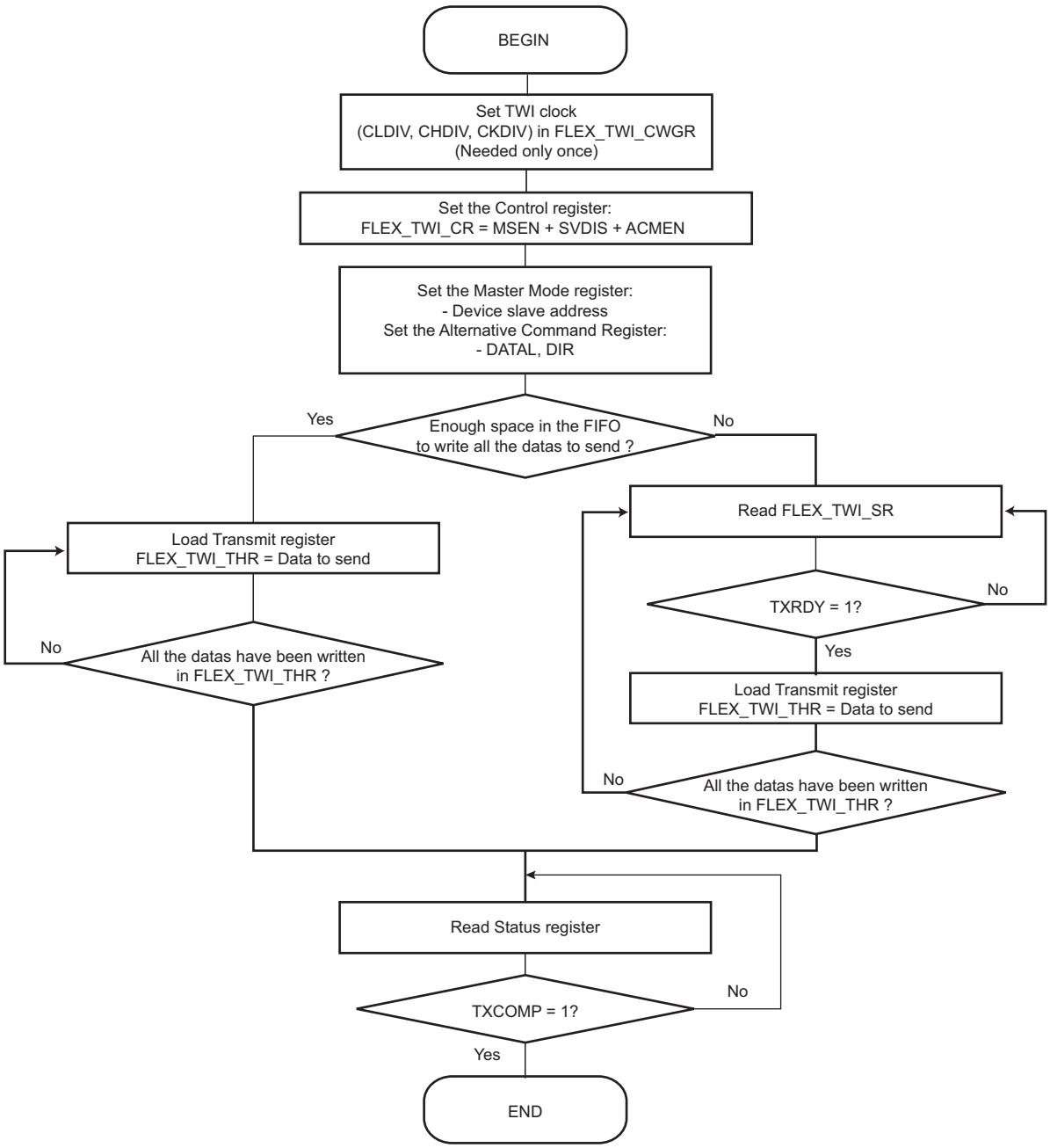


### Sending Data with FIFO Enabled

With the Transmit FIFO enabled, any write access to the TWI Transmit Holding Register (FLEX_TWI_THR) brings the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TXRDY flag state to send multiple data without DMAC.

Knowing the number of data to send and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in FLEX_TWI_THR without checking the TXRDY flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the TWI FIFO Level Register (FLEX_TWI_FLR).

**Figure 44-111. Sending Data with FIFO Flowchart**

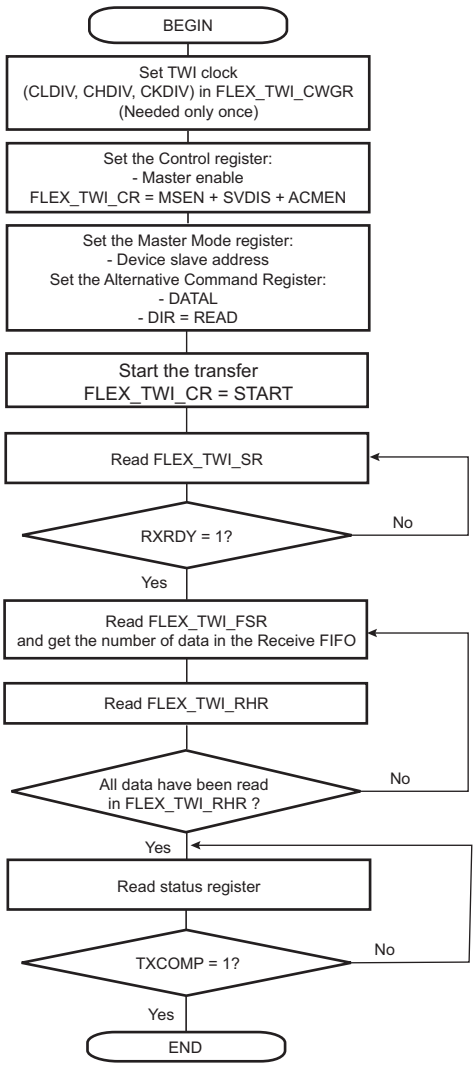


### Receiving Data with FIFO Enabled

With Receive FIFO enabled, any read access on FLEX_TWI_RHR will pull out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RXRDY flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RXRDY flag set to ‘1’), the exact number of data can be checked with the RXFL field in FLEX_TWI_FLR and all the data read successively in FLEX_TWI_RHR without checking the RXRDY flag between each access.

**Figure 44-112. Receiving Data with FIFO Flowchart**




### Clearing/Flushing FIFOs

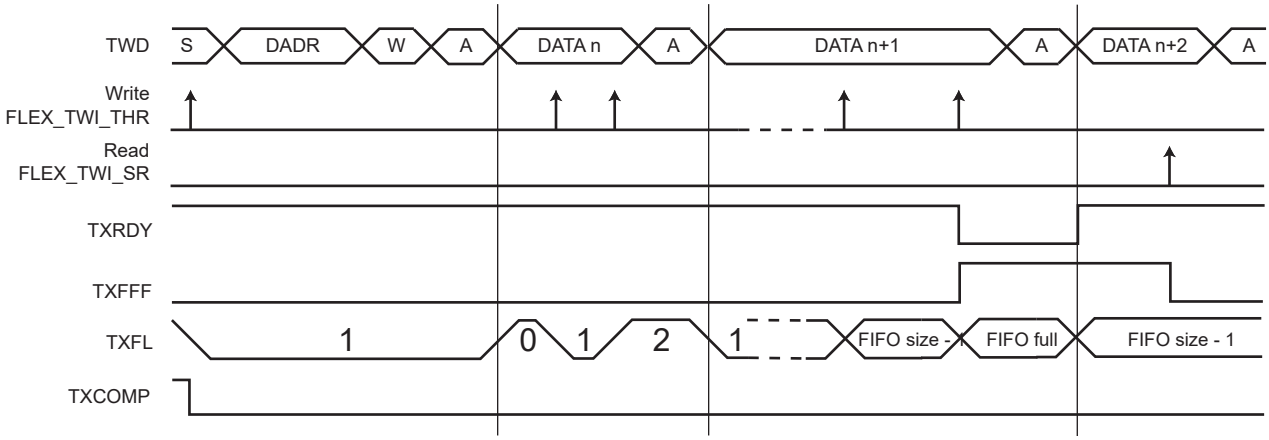
Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in FLEX_TWI_CR.

### TXRDY and RXRDY Behavior

If FIFOs are enabled, the behavior of the TXRDY and RXRDY flags will be slightly different.

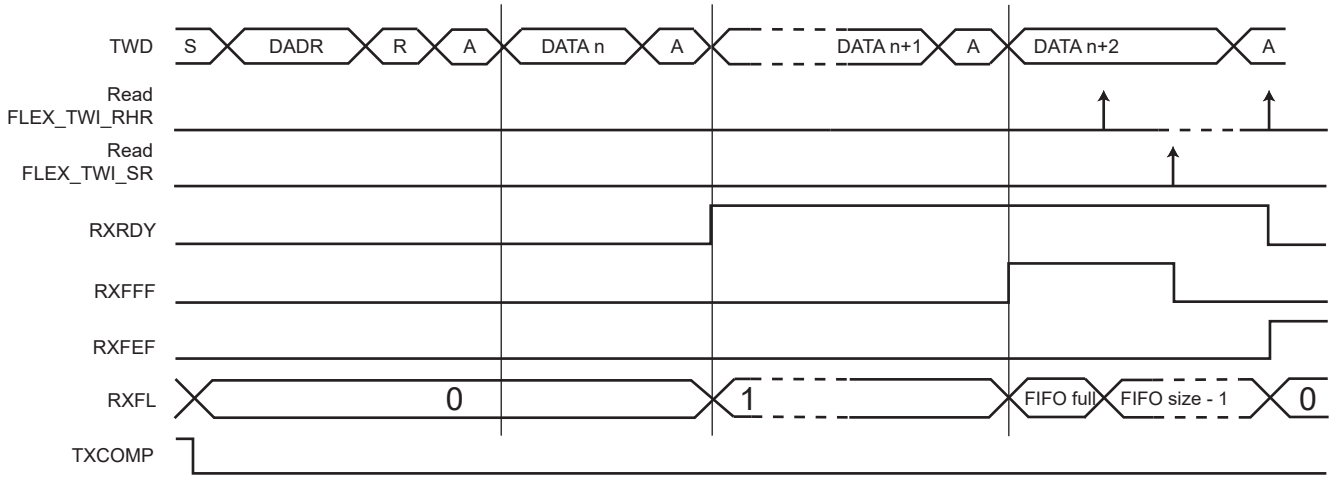
TXRDY will indicate if a data can be written in the Transmit FIFO. By default, the TXRDY flag will then stay at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 44-113. TXRDY in Single Data Mode and TXRDYM = 0x0**




RXRDY will indicate if an unread data is present in the Receive FIFO. By default, the RXRDY flag will then be at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 44-114. RXRDY in Single Data Mode and RXRDYM = 0x0**




TXRDY and RXRDY behavior can be modified using the TXRDYM and RXRDYM fields in the TWI FIFO Mode Register (FLEX_TWI_FMR). In Single Data mode, there is no need to modify the TXRDY and RXRDY behavior; however, it may be useful in Multiple Data Mode.

### Master Multiple Data Mode

When the FIFOs are enabled, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write/read up to four data in one FLEX_TWI_THR/FLEX_TWI_RHR register access.

The number of data to write/read is defined by the size of the register access. If the access is a byte size register access, only one data will be written/read; if the access is a halfword-size register access, then two data will be written/read; finally, if the access is a word-size register access, then four data will be written/read.

Written/Read data are always right-aligned, as shown in Section 44.10.70 “TWI Receive Holding Register (FIFO Enabled)” and Section 44.10.72 “TWI Transmit Holding Register (FIFO Enabled)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six FLEX_TWI_THR-byte write accesses.
- Perform three FLEX_TWI_THR-halfword write accesses.

- Perform one FLEX_TWI_THR-word write access and one FLEX_TWI_THR halfword write access.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six FLEX_TWI_RHR-byte read accesses.
- Perform three FLEX_TWI_RHR-halfword read accesses.
- Perform one FLEX_TWI_RHR-word read access and one FLEX_TWI_RHR-halfword read access.

This mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

• TXRDY and RXRDY Configuration

In Multiple Data mode, the TXRDYM and RXRDYM fields in FLEX_TWI_FMR become useful.

As in Multiple Data mode, it is possible to write several data in the same access it might be useful to configure TXRDY flag behavior to indicate if 1, 2 or 4 data can be written in the FIFO depending on the access to perform on FLEX_TWI_THR.

If, for instance, four data are written each time in FLEX_TWI_THR it might be useful to configure TXRDYM field to 0x2 value so that TXRDY flag will be at ‘1’ only when at least four data can be written in the Transmit FIFO.

In the same way, if four data are read each time in FLEX_TWI_RHR it might be useful to configure RXRDYM field to 0x2 value so that RXRDY flag will be at ‘1’ only when at least four unread data are in the Receive FIFO.

• DMAC

If DMAC transfer is used it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

### Transmit FIFO Lock

If a frame is terminated early due to a not-acknowledge error (NACK flag), SMBus timeout error (TOUT flag) or master code acknowledge error (MACK flag), a lock is set on the Transmit FIFO preventing any new frame from being sent until it is cleared. This allows clearing the FIFO if needed, reset DMAC channels, etc., without any risk.

The LOCK bit in FLEX_TWI_SR is used to check the state of the Transmit FIFO lock.

The Transmit FIFO lock can be cleared setting the TXFLCLR bit to ‘1’ in FLEX_TWI_CR.

### FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

If the Transmit FIFO is full and a write access is performed on FLEX_TWI_THR, it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in FLEX_TWI_FSR.

In Multiple Data mode, if the number of data written in FLEX_TWI_THR (according to the register access size) is bigger than the Transmit FIFO free space, it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in FLEX_TWI_FSR.

- Receive FIFO:

In Multiple Data mode, if the number of data read in FLEX_TWI_RHR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it will generate a Receive FIFO pointer error and set the RXFPTEF flag in FLEX_TWI_FSR.

Pointer error should not happen if FIFO state is checked before writing/reading in FLEX_TWI_THR/FLEX_TWI_RHR. FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a Transmit or Receive pointer error occurs, a software reset must be performed using the SWRST bit in FLEX_TWI_CR. Note that issuing a software while transmitting might leave a slave in an unknown state holding the TWD line. In such case, a Bus Clear Command will allow to make the slave release the TWD line (the first frame sent afterwards might not be received properly by the slave).

#### FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO: The Transmit FIFO threshold can be set using the TXFTHRES field in FLEX_TWI_FMR. Each time the Transmit FIFO goes from the ‘above threshold’ to the ‘equal or below threshold’ state, the TXFTHF flag in FLEX_TWI_FSR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.
- Receive FIFO: The Receive FIFO threshold can be set using the RXFTHRES field in FLEX_TWI_FMR. Each time the Receive FIFO goes from the ‘below threshold’ to the ‘equal to or above threshold’ state, the RXFTHF flag in FLEX_TWI_FSR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using FLEX_TWI_FIER and FLEX_TWI_FIDR.

#### FIFO Flags

FIFOs come with a set of flags which can be configured each to generate an interrupt through FLEX_TWI_FIER and FLEX_TWI_FIDR.

FIFO flags state can be read in FLEX_TWI_FSR. They are cleared on FLEX_TWI_FSR read.

### 44.9.4 Multi-Master Mode

#### 44.9.4.1 Definition

In Multi-Master mode, more than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a STOP. When the STOP is detected, the master that has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 44-116.

#### 44.9.4.2 Different Multi-Master Modes

Two Multi-Master modes may be distinguished:

- TWI as Master Only—TWI is considered as a master only and will never be addressed.
- TWI as Master or Slave—TWI may be either a master or a slave and may be addressed.

Note: Arbitration in supported in both Multi-Master modes.

#### TWI as Master Only

In this mode, the TWI is considered as a master only (MSEN is always at one) and must be driven like a master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the user must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 44-115).

Note: The state of the bus (busy or free) is not indicated in the user interface.

### TWI as Master or Slave

The automatic reversal from master to slave is not supported in case of a lost arbitration.

Then, in the case where TWI may be either a master or a slave, the user must manage the pseudo Multi-Master mode described in the steps below:

1. Program the TWI in Slave mode (SADR + MSDIS + SVEN) and perform a slave access (if TWI is addressed).
2. If the TWI has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program the Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, the TWI scans the bus in order to detect if it is busy or free. When the bus is considered as free, the TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is = 1), the user must program the TWI in Slave mode in case the master that won the arbitration needs to access the TWI.
7. If the TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: In case the arbitration is lost and the TWI is addressed, the TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST = 1. Then the master must repeat SADR.

**Figure 44-115. User Sends Data While the Bus is Busy**

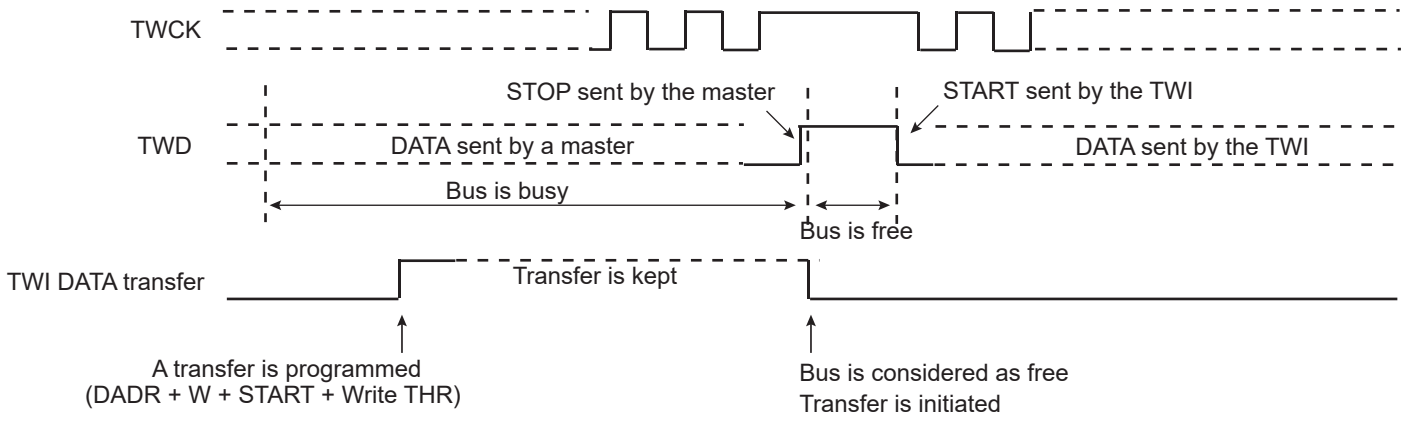
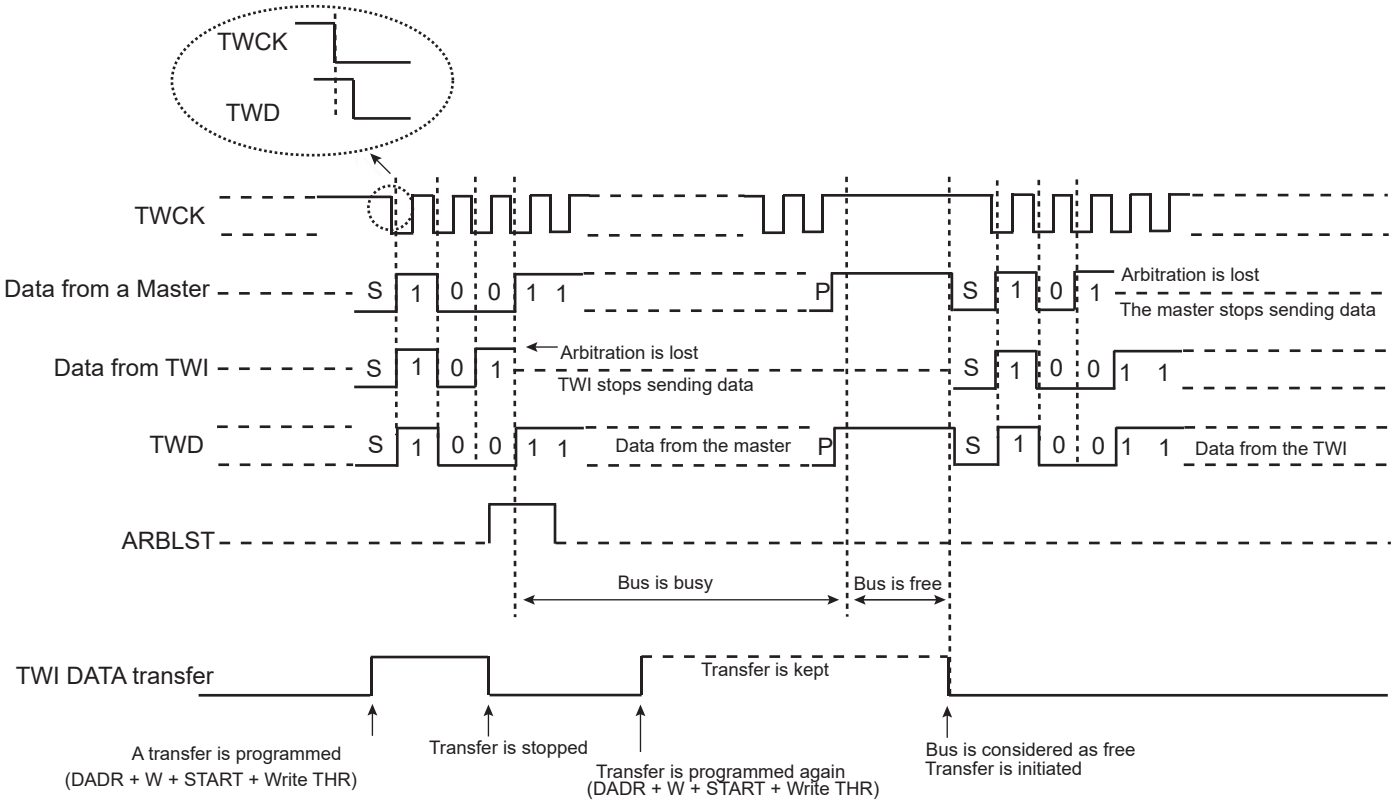
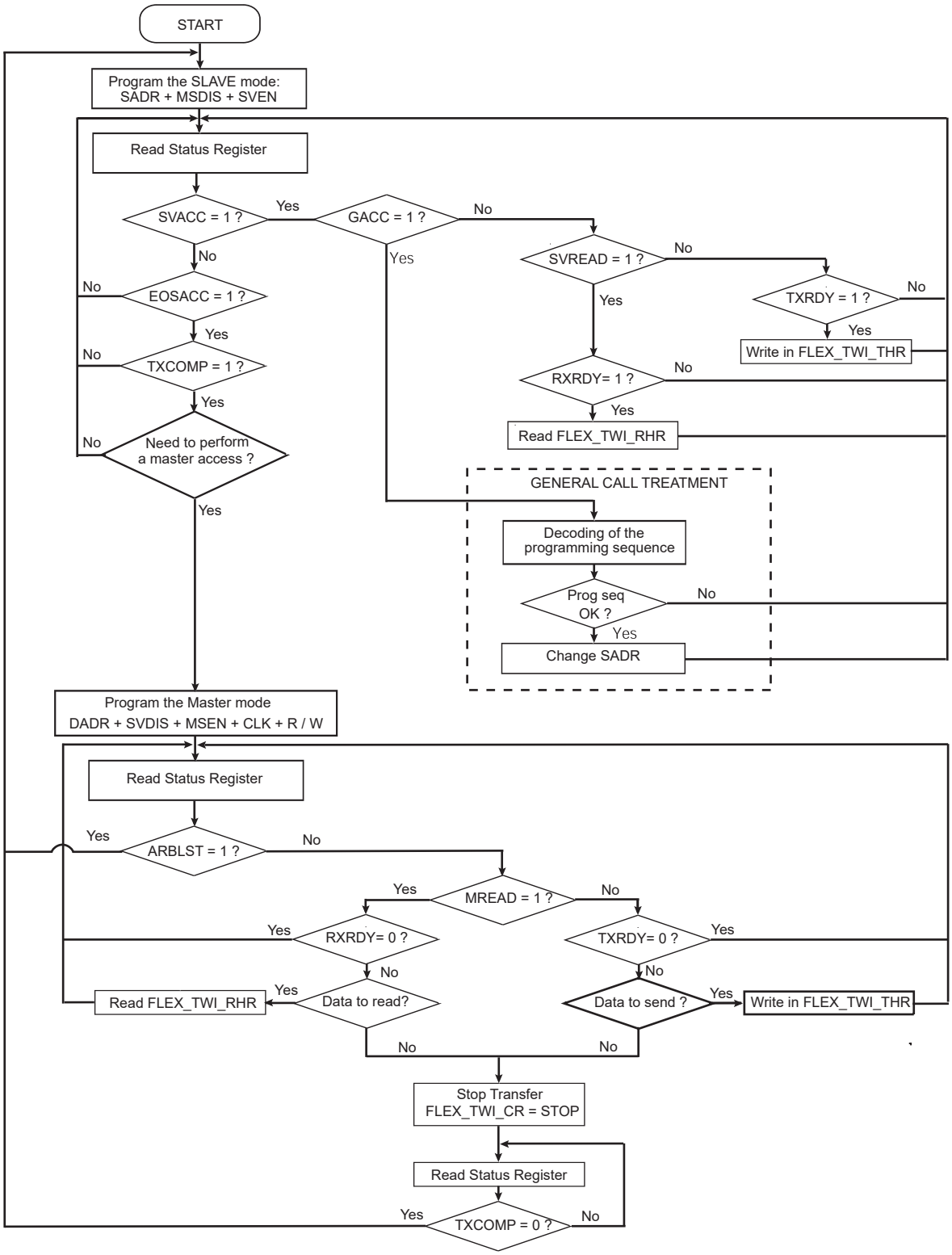


**Figure 44-116. Arbitration Cases**




The flowchart shown in Figure 44-117 gives an example of read and write operations in Multi-Master mode.

**Figure 44-117. Multi-Master Flowchart**



### 44.9.5 Slave Modes

#### 44.9.5.1 Definition

Slave mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED_START and STOP conditions are always provided by the master).

#### 44.9.5.2 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. FLEX_TWI_SMR.SADR: The slave device address is used in order to be accessed by master devices in Read or Write mode.
2. (Optional) FLEX_TWI_SMR.MASK can be set to mask some SADR address bits and thus allow multiple address matching.
3. FLEX_TWI_CR.MSDIS: Disables the Master mode.
4. FLEX_TWI_CR.SVEN: Enables the Slave mode.

As the device receives the clock, values written in FLEX_TWI_CWGR are not processed.

#### 44.9.5.3 Receiving Data

After a START or repeated START condition is detected, and if the address sent by the master matches the slave address programmed in the SADR (Slave Address) field, the SVACC (Slave Access) flag is set and SVREAD (Slave Read) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, EOSACC (End Of Slave Access) flag is set.

**Read Sequence**

In the case of a read sequence (SVREAD is high), the TWI transfers data written in FLEX_TWI_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in FLEX_TWI_THR, the TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the internal shifter is empty and the sent data acknowledged or not. If the data is not acknowledged, the NACK flag is set.

Note that a STOP or a repeated START always follows a NACK.

See Figure 44-118.

Note: To clear the TXRDY flag in Slave mode, in the TW_CR, write bit SVDIS to 1, then write bit SVEN to 1.

**Write Sequence**

In the case of a write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in FLEX_TWI_RHR (TWI Receive Holding Register). RXRDY is reset when reading FLEX_TWI_RHR.

TWI continues receiving data until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the write sequence TXCOMP flag is set and SVACC reset.

See Figure 44-119.

**Clock Stretching Sequence**

If FLEX_TWI_THR or FLEX_TWI_RHR is not written/read in time, the TWI performs a clock stretching.

Clock stretching information is given by the SCLWS (Clock Wait State) bit.

See Figure 44-121 and Figure 44-122.

Note: Clock stretching can be disabled by configuring the SCLWSDIS bit in FLEX_TWI_SMR. In that case, UNRE and OVRE flags will indicate underrun (when FLEX_TWI_THR is not filled on time) or overrun (when FLEX_TWI_RHR is not read on time).

#### General Call

In the case where a GENERAL CALL is performed, the GACC (General Call Access) flag is set.

After GACC is set, it is up to the user to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See Figure 44-120.

### 44.9.5.4 Data Transfer

#### Read Operation

The Read mode is defined as a data requirement from the master.

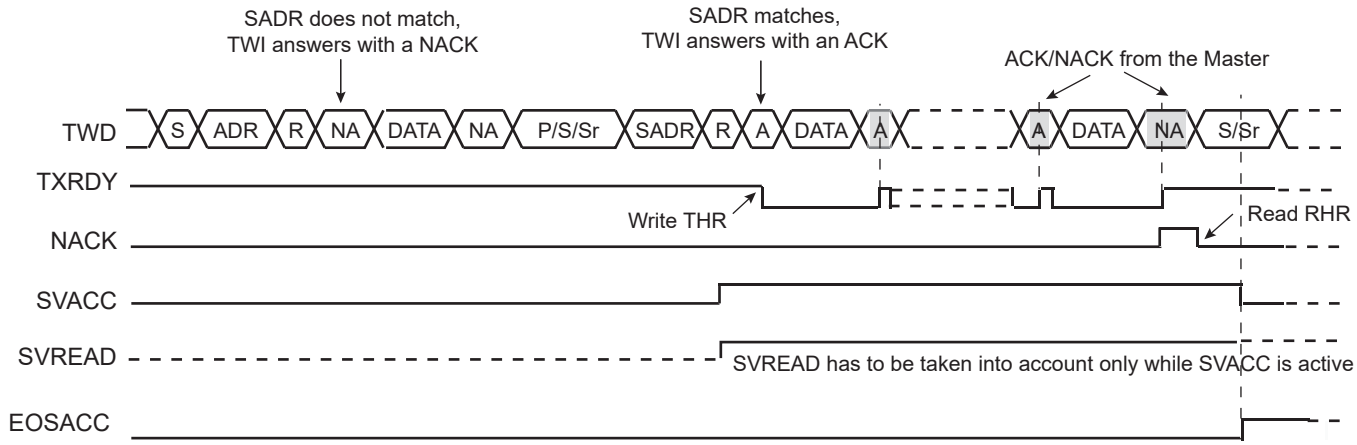
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in FLEX_TWI_THR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 44-118 describes the read operation.

**Figure 44-118. Read Access Ordered by a Master**




Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. TXRDY is reset when data has been transmitted from FLEX_TWI_THR to the internal shifter and set when this data has been acknowledged or non acknowledged.

#### Write Operation

The Write mode is defined as a data transmission from the master.

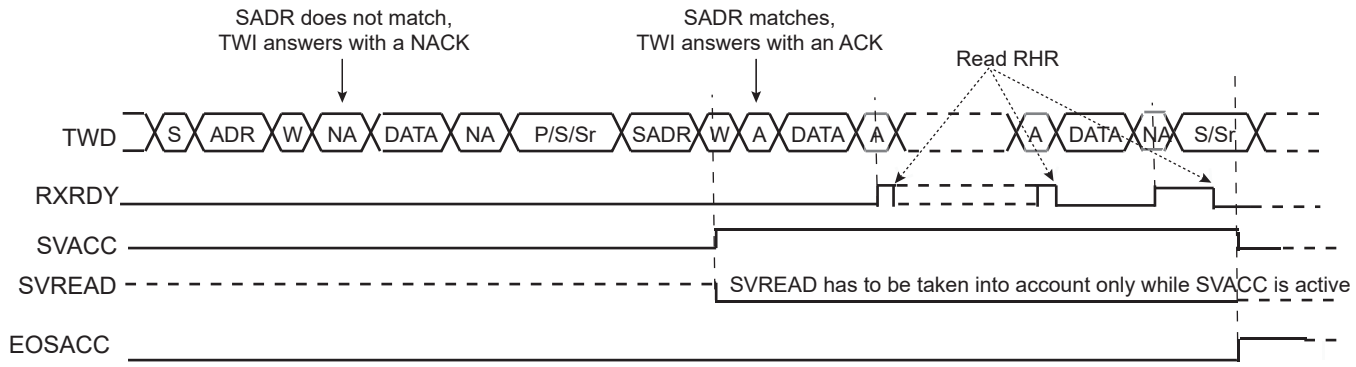
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in FLEX_TWI_RHR.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 44-119 describes the write operation.

**Figure 44-119. Write Access Ordered by a Master**




Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the internal shifter to FLEX_TWI_RHR, and reset when this data is read.

### General Call

The general call is performed in order to change the address of the slave.

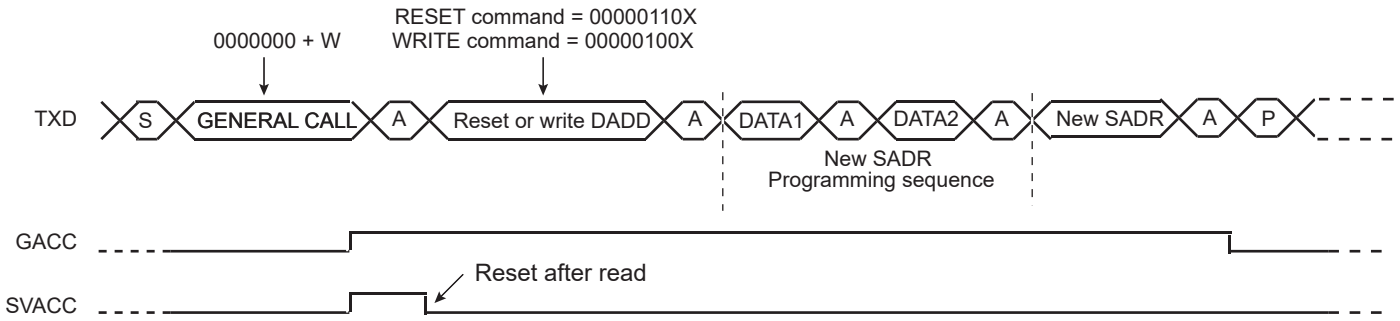
If a GENERAL CALL is detected, GACC is set.

After the detection of general call, it is up to the user to decode the commands which follow.

In case of a WRITE command, the user has to decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 44-120 describes the general call access.

**Figure 44-120. Master Performs a General Call**




Note: This method allows to create a user-specific programming sequence by choosing the number of programming bytes. The programming sequence has to be provided to the master.

### Clock Stretching

In both Read and Write modes, it may happen that FLEX_TWI_THR/FLEX_TWI_RHR buffer is not filled/emptied before the transmission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

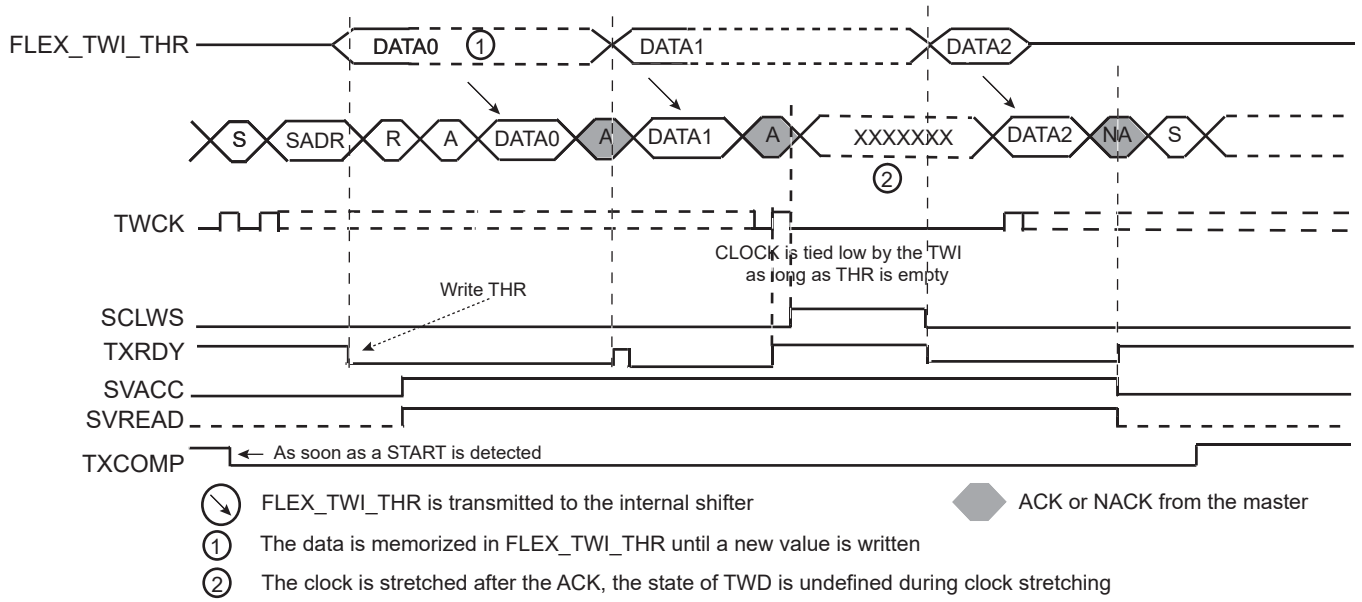
Note: Clock stretching can be disabled by setting the SCLWSDIS bit in FLEX_TWI_SMR. In that case, the UNRE and OVRE flags will indicate underrun (when FLEX_TWI_THR is not filled on time) or overrun (when FLEX_TWI_RHR is not read on time).

#### Clock Stretching in Read Mode

The clock is tied low if the internal shifter is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the internal shifter is loaded.

Figure 44-121 describes the clock stretching in Read mode.

**Figure 44-121. Clock Stretching in Read Mode**



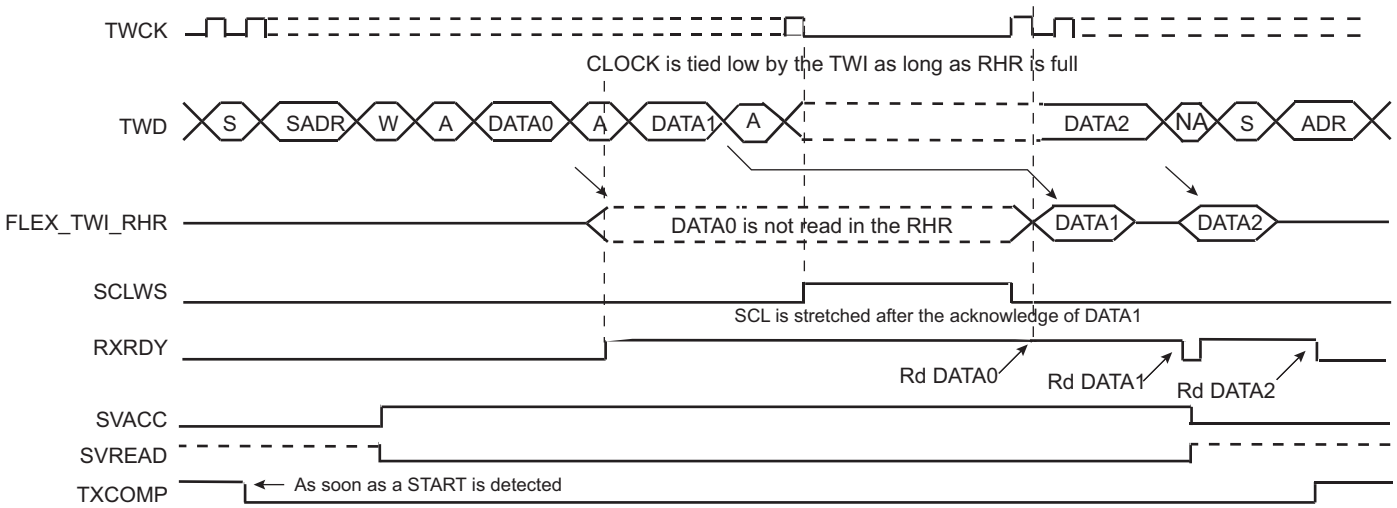

Notes:
1. TXRDY is reset when data has been written in FLEX_TWI_THR to the internal shifter, and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock stretching mechanism is started.

**Clock Stretching in Write Mode**

The clock is tied low if the internal shifter and FLEX_TWI_RHR are full. If a STOP or REPEATED_START condition was not detected, it is tied low until FLEX_TWI_RHR is read.

Figure 44-122 describes the clock stretching in Write mode.

**Figure 44-122. Clock Stretching in Write Mode**




Notes:
1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
2. SCLWS is automatically set when the clock stretching mechanism is started and automatically reset when the mechanism is finished.

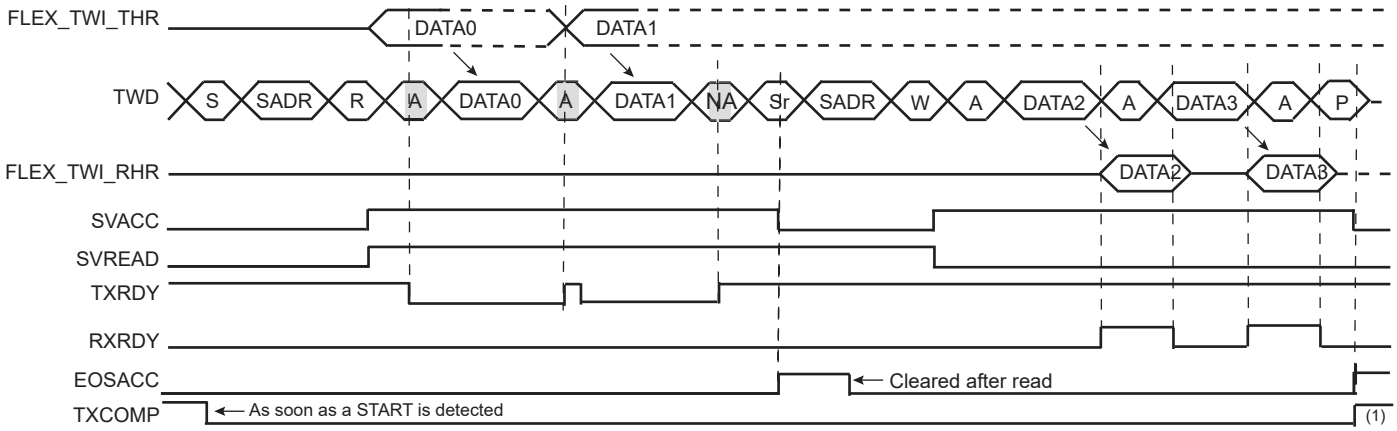
### Reversal after a Repeated Start

#### Reversal of Read to Write

The master initiates the communication by a read command and finishes it by a write command.

Figure 44-123 describes the repeated start and the reversal from Read mode to Write mode.

**Figure 44-123. Repeated Start and Reversal from Read Mode to Write Mode**

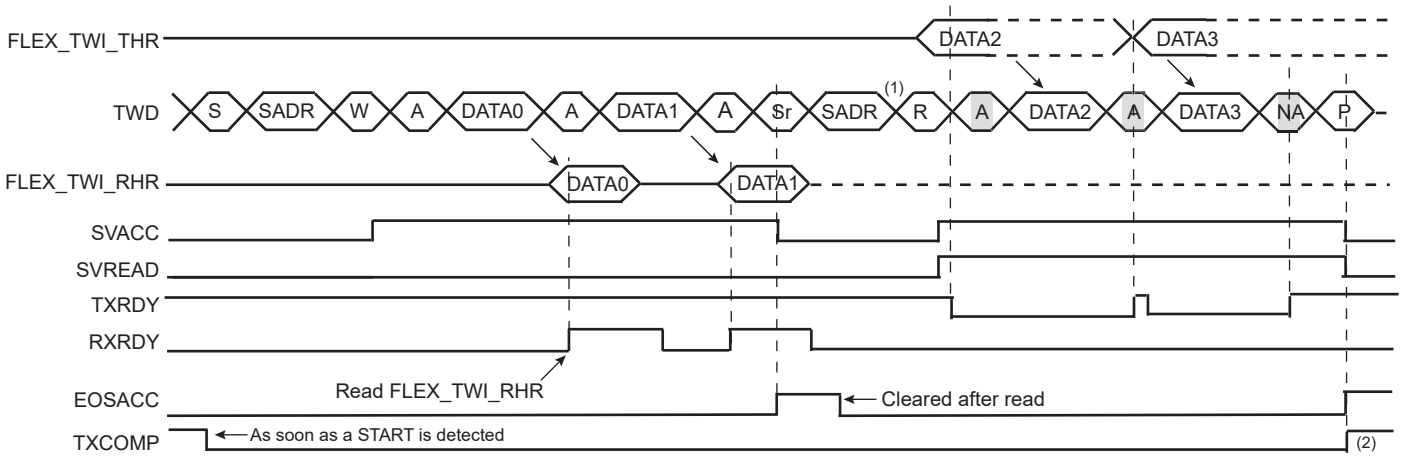



Note:
1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

#### Reversal of Write to Read

The master initiates the communication by a write command and finishes it by a read command. Figure 44-124 describes the repeated start and the reversal from Write mode to Read mode.

**Figure 44-124. Repeated Start and Reversal from Write Mode to Read Mode**




Notes:
1. In this case, if FLEX_TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.
2. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

### SMBus Mode

SMBus mode is enabled when SMEN bit is written to one in FLEX_TWI_CR. SMBus mode operation is similar to I²C operation with the following exceptions:

1. Only 7-bit addressing can be used.
2. The SMBus standard describes a set of timeout values to ensure progress and throughput on the bus. These timeout values must be programmed into FLEX_TWI_SMBTR.
3. Transmissions can optionally include a CRC byte, called Packet Error Check (PEC).

4. A set of addresses have been reserved for protocol handling, such as alert response address (ARA) and host header (HH) address. Address matching on these addresses can be enabled by configuring FLEX_TWI_CR appropriately.

**Packet Error Checking**

Each SMBus transfer can optionally end with a CRC byte, called the PEC byte. Writing the PECEN bit in FLEX_TWI_CR to one will send/check the FLEX_TWI_ACR.PEC field in the current transfer. The PEC generator is always updated on every bit transmitted or received, so that PEC handling on following linked transfers will be correct.

In Slave Receiver mode, the master calculates a PEC value and transmits it to the slave after all data bytes have been transmitted. Upon reception of this PEC byte, the slave will compare it to the PEC value it has computed itself. If the values match, the data was received correctly, and the slave will return an ACK to the master. If the PEC values differ, data was corrupted, and the slave will return a NACK value. The PECERR bit in FLEX_TWI_SR is set automatically if a PEC error occurred.

In Slave Transmitter mode, the slave calculates a PEC value and transmits it to the master after all data bytes have been transmitted. Upon reception of this PEC byte, the master will compare it to the PEC value it has computed itself. If the values match, the data was received correctly. If the PEC values differ, data was corrupted, and the master must take appropriate action.

See Section 44.9.5.8 for detailed flowcharts.

**Timeouts**

The TWI SMBus Timing Register (FLEX_TWI_SMBTR) configures the SMBus timeout values. If a timeout occurs, the slave will leave the bus. The TOUT bit is also set in FLEX_TWI_SR.

### 44.9.5.5 High-Speed Slave Mode

High-speed mode is enabled when the HSEN bit is written to one in FLEX_TWI_CR. Furthermore, the analog pad filter must be enabled, the PADFEN bit must be written to one in FLEX_TWI_FILTR and the FILT bit must be cleared. TWI High-speed mode operation is similar to TWI operation with the following exceptions:

1. A master code is received first at normal speed before entering High-speed mode period.
2. When TWI High-speed mode is active, clock stretching is only allowed after acknowledge (ACK), not-acknowledge (NACK), START (S) or repeated START (Sr) (asa consequence, OVF may happen).

TWI High-speed mode allows transfers of up to 3.4 Mbit/s.

The TWI slave in High-speed mode requires that the peripheral clock runs at a minimum of 14 MHz if slave clock stretching is enabled (SCLWSDIS bit at ‘0’). If slave clock stretching is disabled (SCLWSDIS bit at ‘1’), the peripheral clock must run at a minimum of 11 MHz (assuming the system has no latency).

Notes:
1. When slave clock stretching is disabled, FLEX_TWI_RHR must always be read before receiving the next data (MASTER write frame). It is strongly recommended to use either the polling method on the RXRDY flag in FLEX_TWI_SR, or the DMA. If the receive is managed by an interrupt, the TWI interrupt priority must be set to the right level and its latency minimized to avoid receive overrun.
2. When slave clock stretching is disabled, FLEX_TWI_THR must be filled with the first data to send before the beginning of the frame (MASTER read frame). It is strongly recommended to use either the polling method on the TXRDY flag in FLEX_TWI_SR, or the DMA. If the transmit is managed by an interrupt, the TWI interrupt priority must be set to the right level and its latency minimized to avoid transmit underrun.

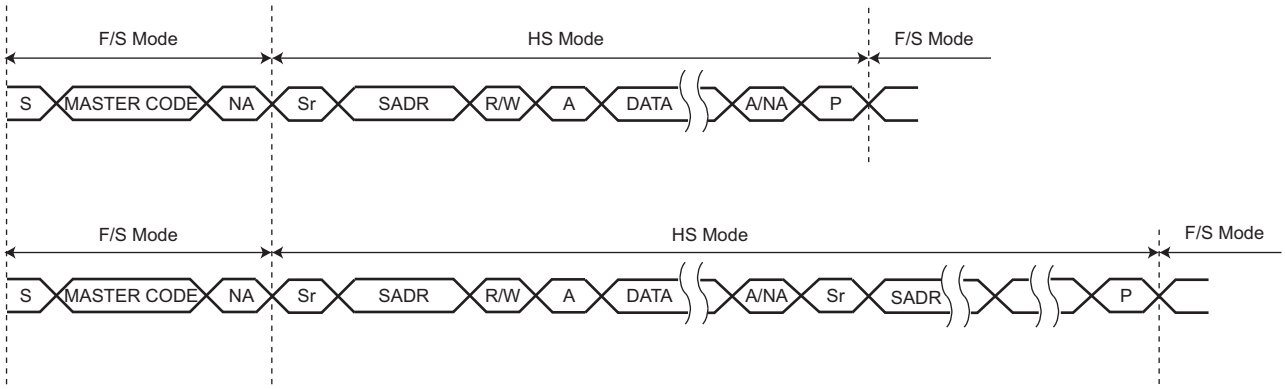
**Read/Write Operation**

A TWI high-speed frame always begins with the following sequence:

1. START condition (S)
2. Master Code (0000 1XXX)
3. Not-acknowledge (NACK)

When the TWI is programmed in Slave mode and TWI High-speed mode is activated, master code matching is activated and internal timings are set to match the TWI High-speed mode requirements.

**Figure 44-125. High-Speed Mode Read/Write**




### Usage

TWI High-speed mode usage is the same as the standard TWI (see Section 44.9.3.13).

### 44.9.5.6 Alternative Command

In Slave mode, the Alternative Command mode is used when the SMBus mode is enabled to send or check the PEC byte.

The Alternative Command mode is enabled by setting the ACMEN bit of the TWIHS Control Register, and the transfer is configured in TWIHS_ACR.

For a combined transfer with PEC, only the NPEC bit in TWIHS_ACR must be set as the PEC byte is sent once at the end of the frame.

See Section 44.9.5.8 “Slave Read Write Flowcharts” for detailed flowcharts.

### 44.9.5.7 TWI Asynchronous and Partial Wakeup (SleepWalking)

The TWI module includes an asynchronous start condition detector, it is capable of waking the device up from a Sleep mode upon an address match (and optionally an additional data match), including Sleep modes where the TWI peripheral clock is stopped.

The FLEX_TWI_RHR register must be read prior to enable the asynchronous and partial wakeup.

After detecting the START condition on the bus, the TWI will stretch TWCK until the TWI peripheral clock has started. The time required for starting the TWI peripheral depends on which Sleep mode the device is in. After the TWI peripheral clock has started, the TWI releases its TWCK stretching and receives one byte of data (slave address) on the bus. At this time, only a limited part of the device, including the TWI module, receives a clock, thus saving power. If the address phase causes a TWIS address match (and optionally if the first data byte causes data match as well), the entire device is wakened and normal TWI address matching actions are performed. Normal TWI transfer then follows. If the TWI module is not addressed (or if the optional data match fails), the TWI peripheral clock is automatically stopped and the device returns to its original Sleep mode.

The TWI module has the capability to match on more than one address. The SADR1EN, SADR2EN and SADR3EN bits in FLEX_TWI_SMR allow to enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in FLEX_TWI_SWMR. The SleepWalking matching process can be extended to the first received data byte if DATAMEN bit in FLEX_TWI_SMR is set, in that case a complete matching includes address matching and first received data matching. The DATAM field in FLEX_TWI_SWMR allows to configure the data to match on the first received byte.

When the system is in Active mode and the TWI enters asynchronous partial Wakeup mode, the flag SVACC must be programmed as the unique source of the TWI interrupt and the data match comparison must be disabled.

When the system exits Wait mode as the result of a matching condition, the SVACC flag is used to determine if the TWI is the source of the exit from Wait mode.

**Figure 44-126. Address Match and Data Matching Disabled**

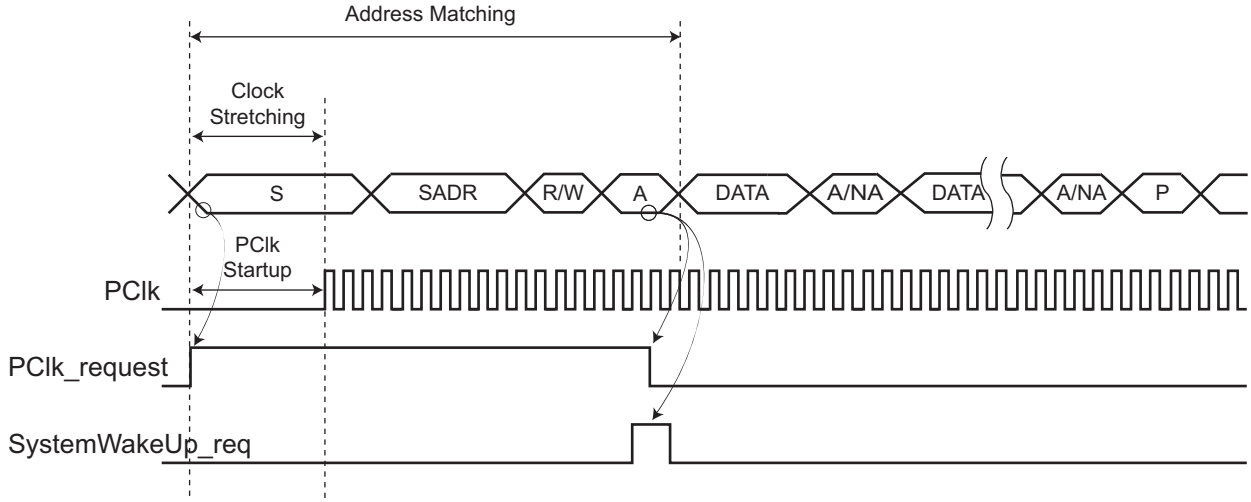



**Figure 44-127. Address Does Not Match and Data Matching Disabled**

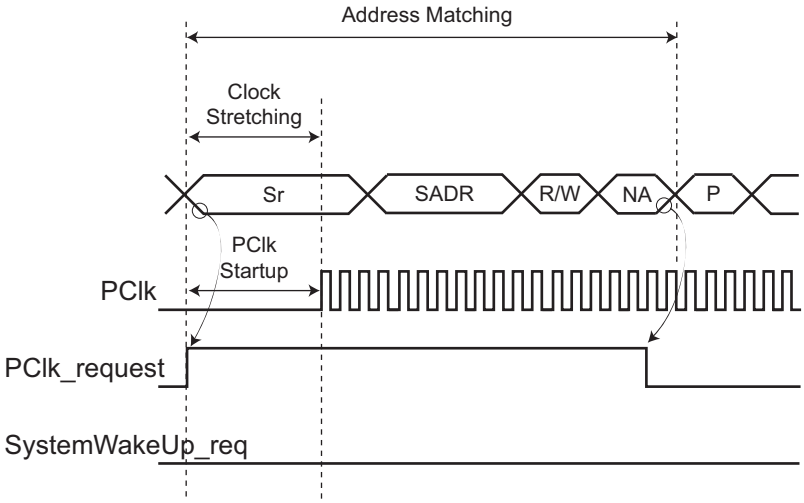


**Figure 44-128. Address and Data Match (Data Matching Enabled)**

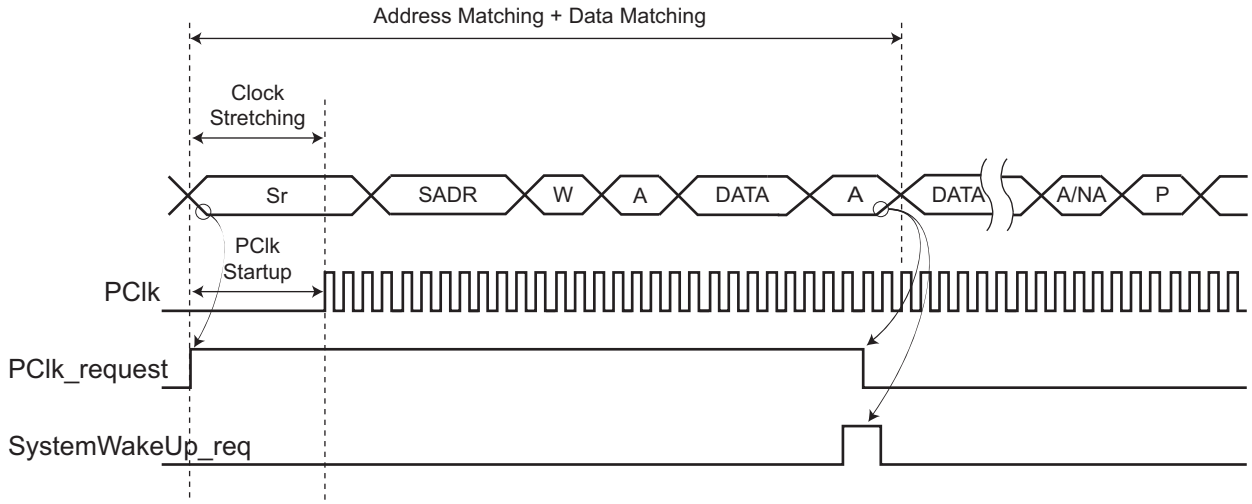
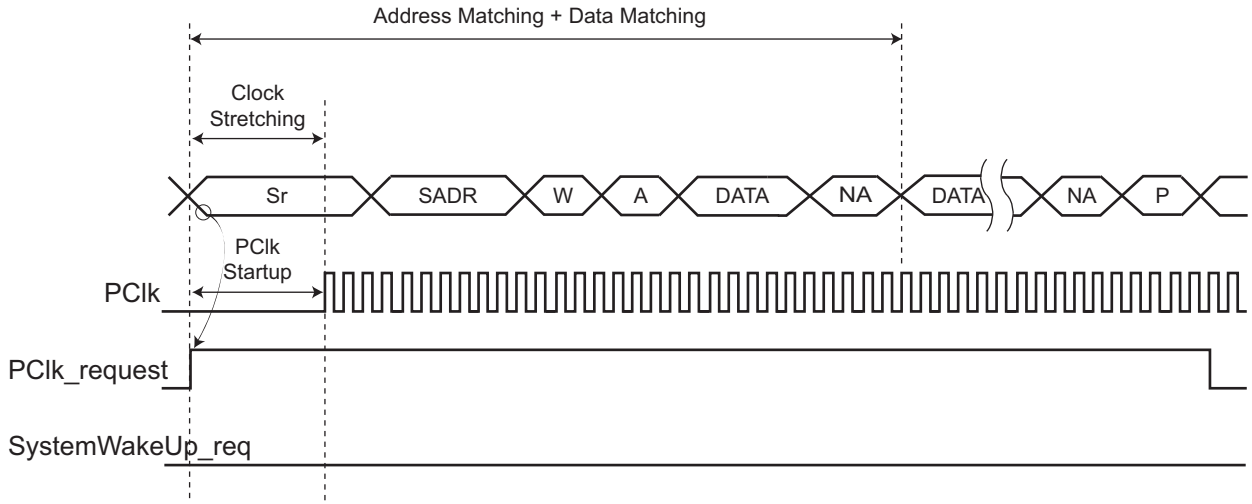



**Figure 44-129. Address Matches and Data Do Not Match (Data Matching Enabled)**




#### 44.9.5.8 Slave Read Write Flowcharts

The flowchart shown in Figure 44-130 gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the Interrupt Enable Register (FLEX_TWI_IER) be configured first.

**Figure 44-130. Read Write Flowchart in Slave Mode**

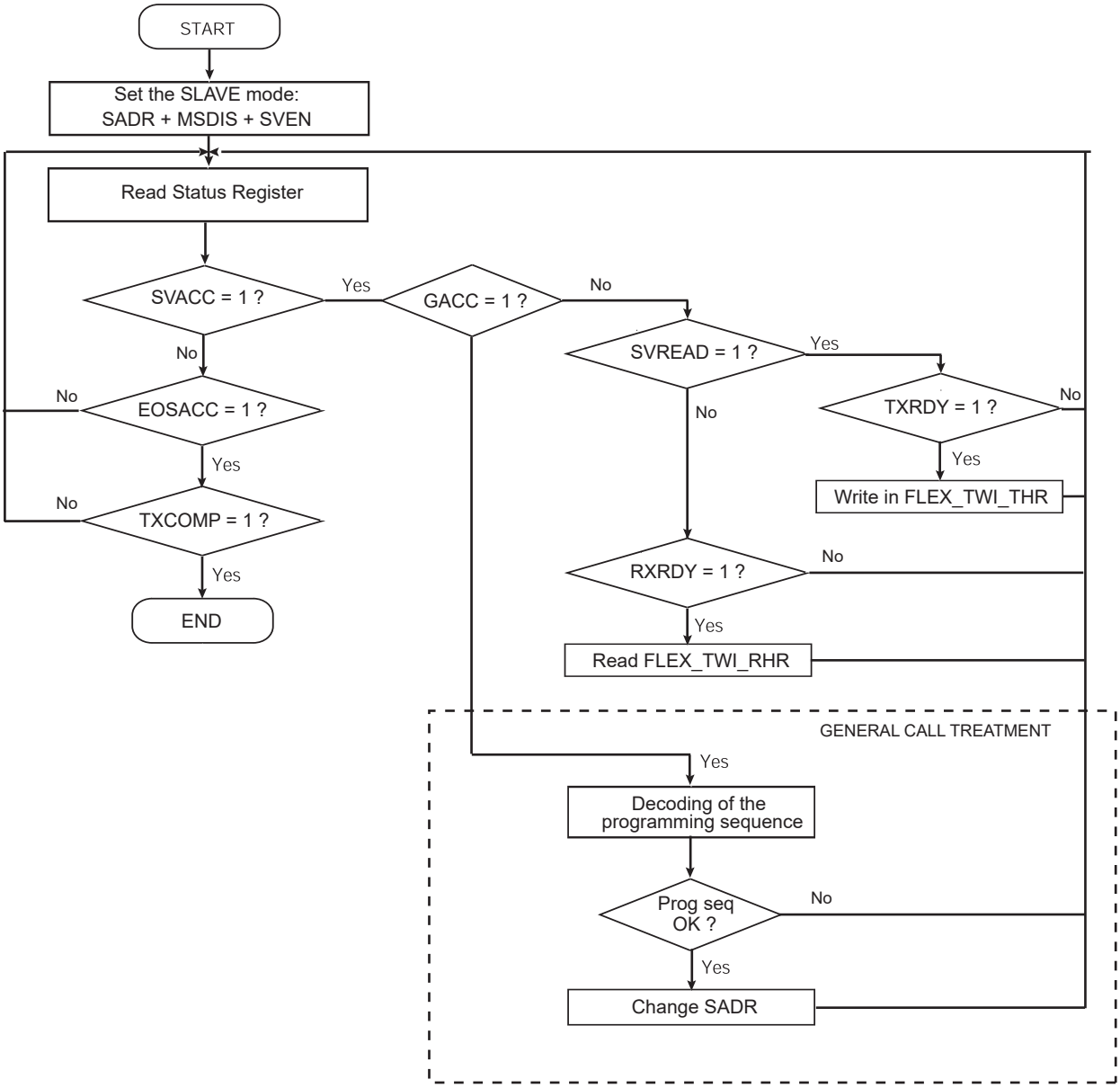


**Figure 44-131. Read Write Flowchart in Slave Mode with SMBus PEC**

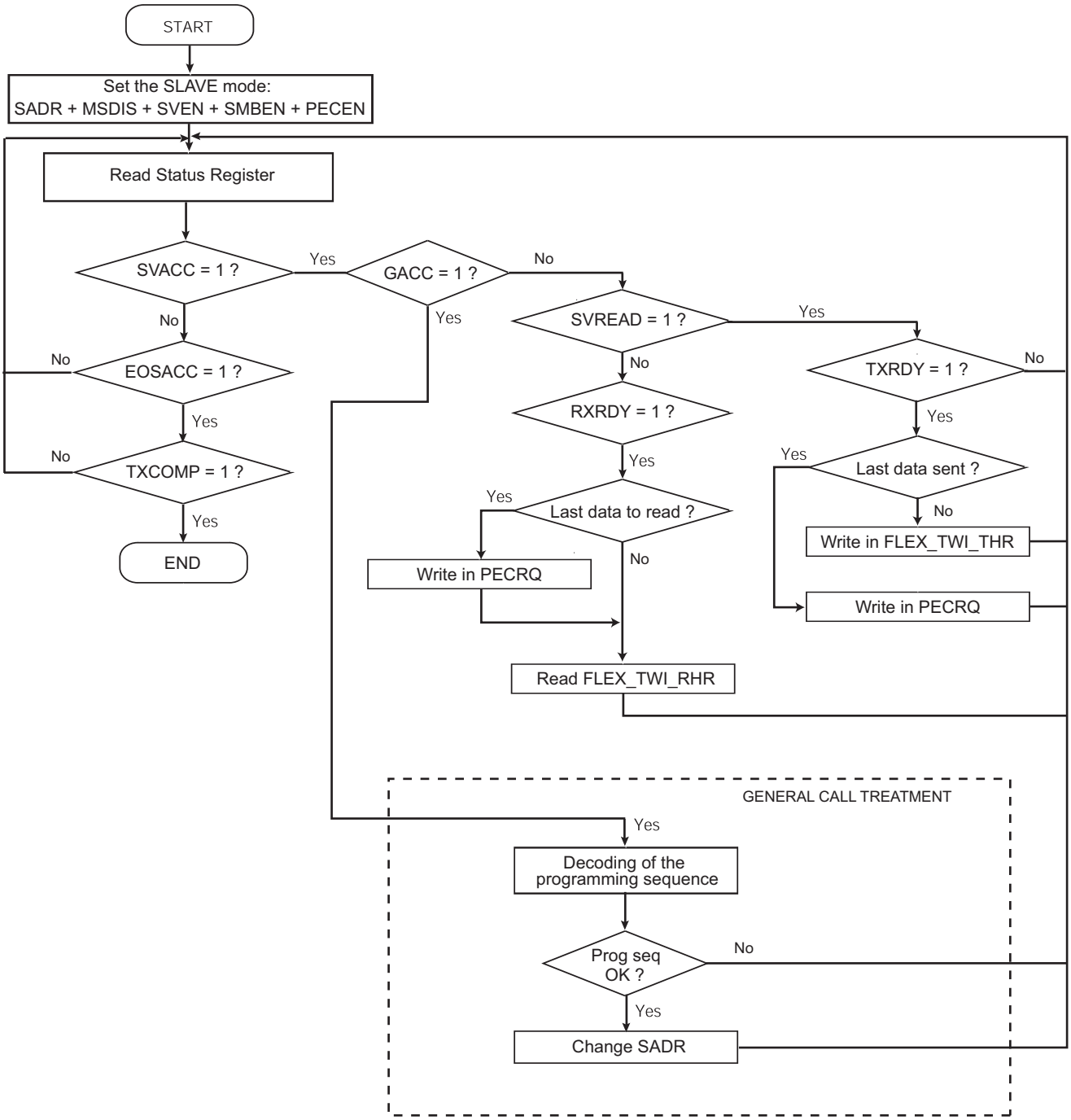
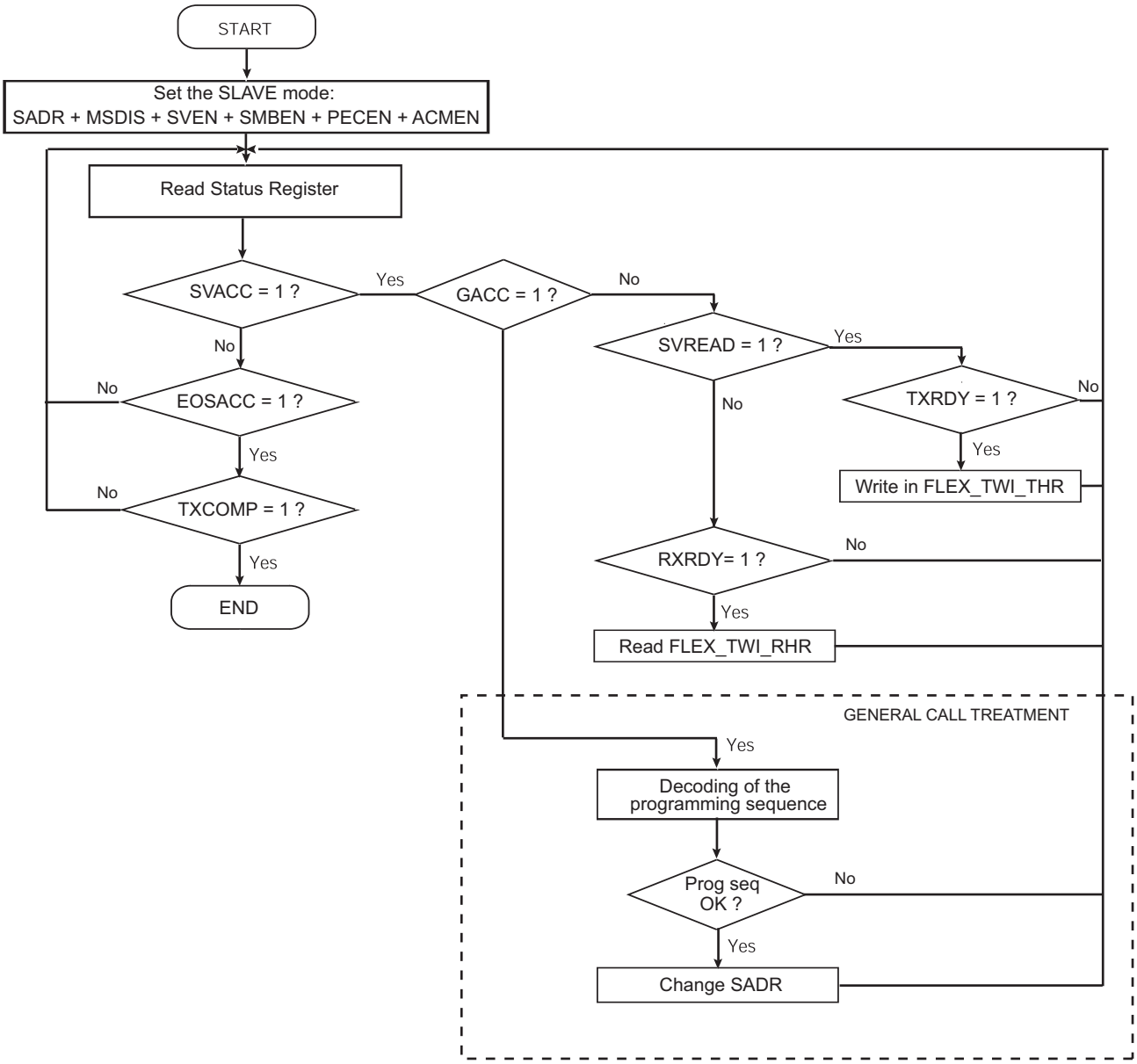


**Figure 44-132. Read Write Flowchart in Slave Mode with SMBus PEC and Alternative Command Mode**

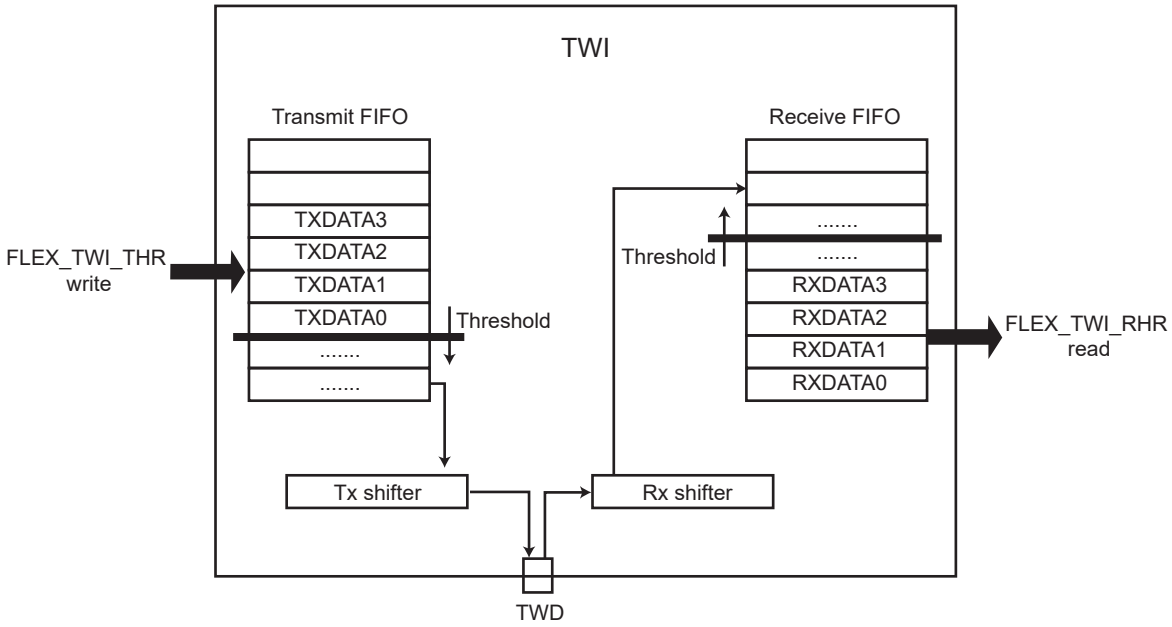



### 44.9.5.9 FIFOs

The TWI includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in FLEX_TWI_CR. It is recommended to disable both Master and Slave modes before enabling or disabling the FIFO (MSDIS and SVDIS bit in FLEX_TWI_CR).

Writing the FIFOEN bit to ‘1’ will enable a 16-data Transmit FIFO and a 16-data Receive FIFO.

**Figure 44-133. FIFOs Block Diagram**




### Sending/Receiving Data with FIFO Enabled

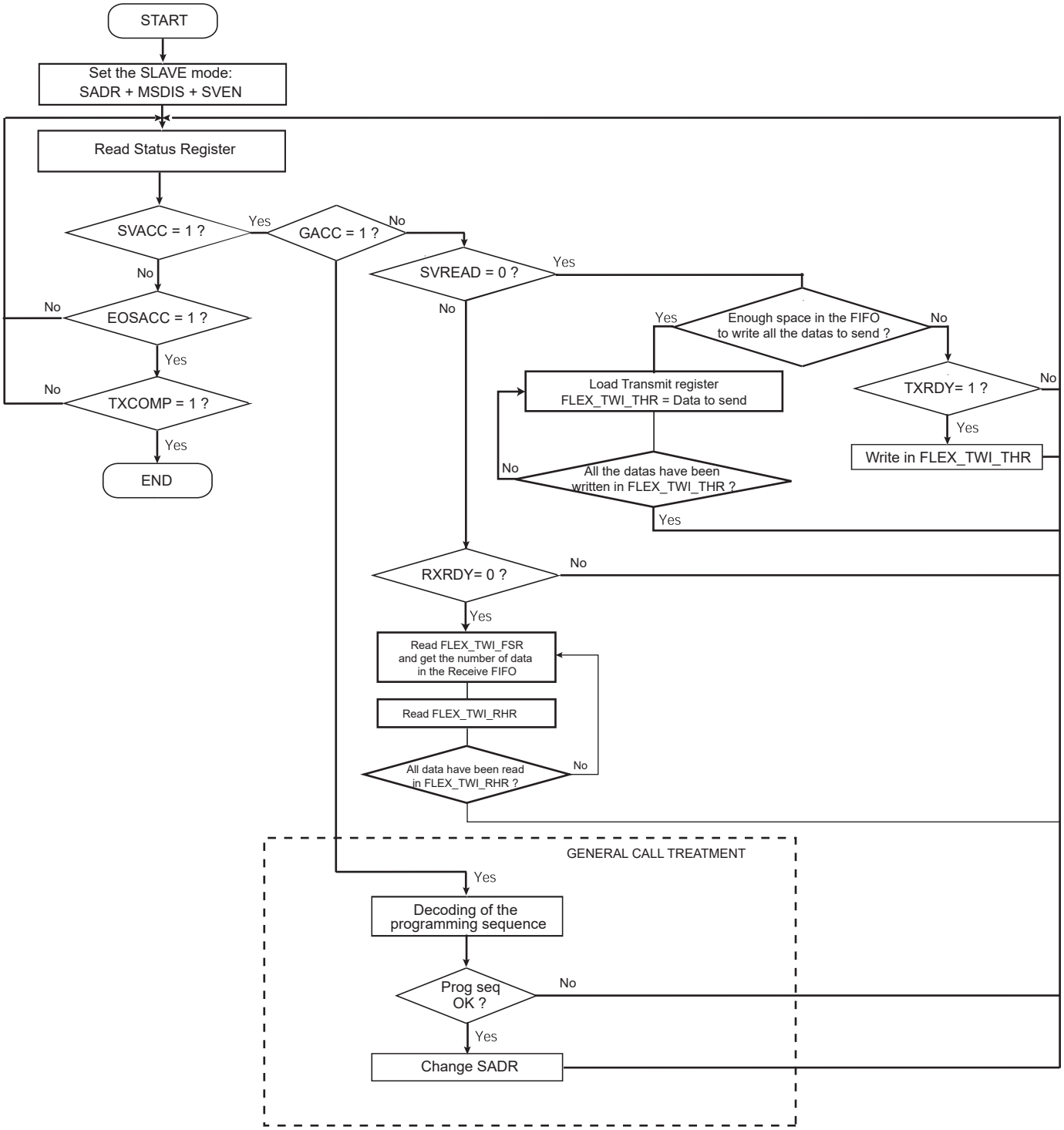
With the Transmit FIFO enabled, any write access to FLEX_TWI_THR will bring the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TXRDY flag state to send multiple data without DMAC.

Knowing the number of data to send and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in FLEX_TWI_THR without checking the TXRDY flag between each access. The Transmit FIFO state can be checked reading the TXFL field in FLEX_TWI_FLR.

With Receive FIFO enabled, any read access on FLEX_TWI_RHR will pull out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RXRDY flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RXRDY flag set to ‘1’), the exact number of data can be checked with the RXFL field in FLEX_TWI_FLR and all the data read successively in FLEX_TWI_RHR without checking the RXRDY flag between each access.

**Figure 44-134. Sending/Receiving Data with FIFO Flowchart**




### Clearing/Flushing FIFOs

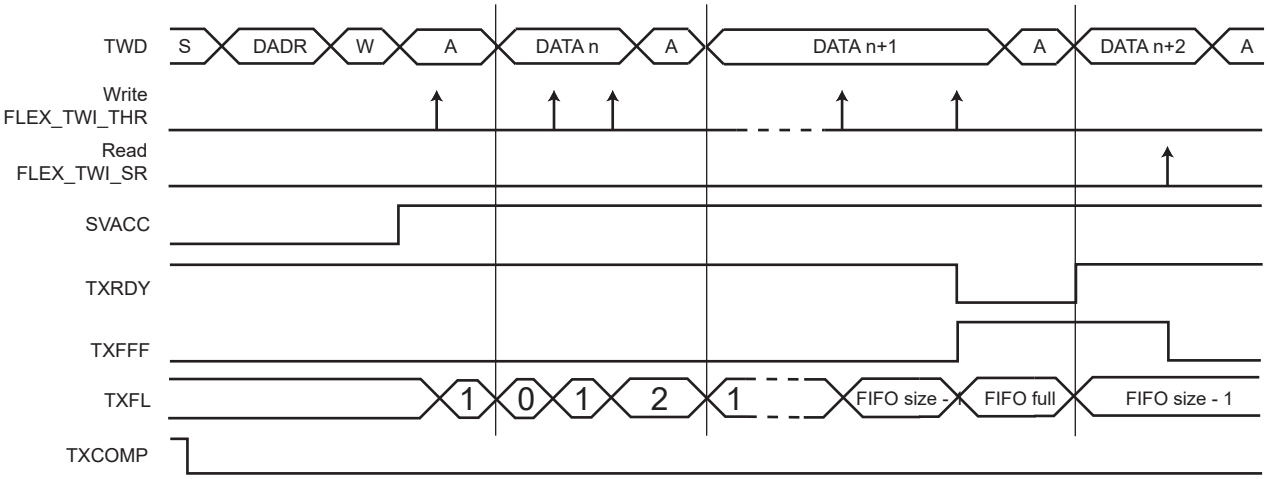
Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in FLEX_TWI_CR.

### TXRDY and RXRDY Behavior

If FIFOs are enabled, the behavior of the TXRDY and RXRDY flags will be slightly different.

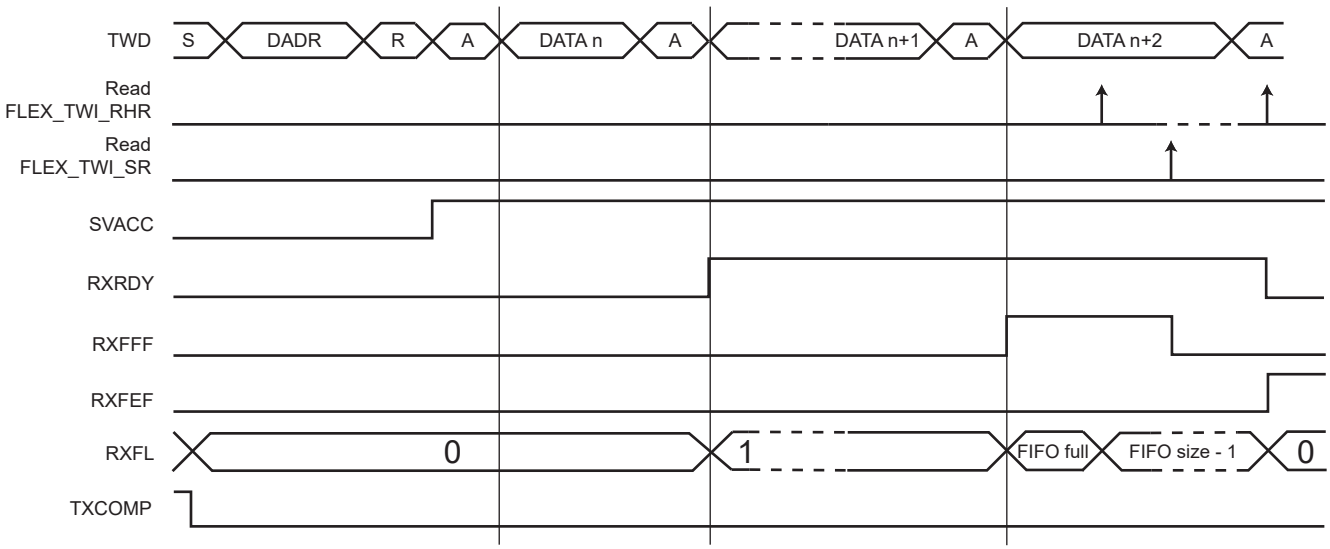
TXRDY will indicate if a data can be written in the Transmit FIFO. By default, the TXRDY flag will then stay at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 44-135. TXRDY in Single Data Mode and TXRDYM = 0x0**




RXRDY will indicate if an unread data is present in the Receive FIFO. By default, the RXRDY flag will then be at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 44-136. RXRDY in Single Data Mode and RXRDYM = 0x0**




TXRDY and RXRDY behavior can be modified using the TXRDYM and RXRDYM fields in FLEX_TWI_FMR. In Single Data Mode, there is no need to modify the TXRDY and RXRDY behavior; however, it may be useful in Multiple Data mode.

### Slave Multiple Data Mode

When the FIFOs are enabled, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write/read up to four data in one FLEX_TWI_THR/FLEX_TWI_RHR register access.

The number of data to write/read is defined by the size of the register access. If the access is a byte-size register access, only one data will be written/read; if the access is a halfword-size register access, then two data will be written/read and finally, if the access is a word-size register access, then four data will be written/read.

Written/Read data are always right-aligned, as shown in Section 44.10.70 “TWI Receive Holding Register (FIFO Enabled)” and Section 44.10.72 “TWI Transmit Holding Register (FIFO Enabled)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six FLEX_TWI_THR-byte write accesses.
- Perform three FLEX_TWI_THR-halfword write accesses.
- Perform one FLEX_TWI_THR-word write access and one FLEX_TWI_THR-halfword write access.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six FLEX_TWI_RHR-byte read accesses.
- Perform three FLEX_TWI_RHR-halfword read accesses.
- Perform one FLEX_TWI_RHR-word read access and one FLEX_TWI_RHR-halfword read access.

Multiple Data mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

• TXRDY and RXRDY configuration

In Multiple Data mode, the TXRDYM and RXRDYM fields in FLEX_TWI_FMR become useful.

As in Multiple Data mode, it is possible to write several data in the same access; it might be useful to configure the TXRDY flag behavior to indicate if 1, 2 or 4 data can be written in the FIFO depending on the access to perform on FLEX_TWI_THR.

If, for instance, four data are written each time in FLEX_TWI_THR it might be useful to configure the TXRDYM field to 0x2 value so that the TXRDY flag will be at ‘1’ only when at least four data can be written in the Transmit FIFO.

In the same way, if four data are read each time in FLEX_TWI_RHR, it might be useful to configure the RXRDYM field to 0x2 value so that the RXRDY flag will be at ‘1’ only when at least four unread data are in the Receive FIFO.

• DMAC

If DMAC transfer is used, it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

### Transmit FIFO Lock

If a frame is terminated early due to a not-acknowledge error (NACK flag), SMBus timeout error (TOUT flag) or master code acknowledge error (MACK flag), a lock is set on the Transmit FIFO preventing any new frame from being sent until it is cleared. This allows clearing the FIFO if needed, reset DMAC channels, etc., without any risk.

The LOCK bit in FLEX_TWI_SR is used to check the state of the Transmit FIFO lock.

The Transmit FIFO lock can be cleared by setting the TXFLCLR bit to ‘1’ in FLEX_TWI_CR.

### FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

  If the Transmit FIFO is full and a write access is performed on FLEX_TWI_THR it will generate a Transmit FIFO pointer error and set the TXFPTEF flag in FLEX_TWI_FSR.

  In Multiple Data mode, if the number of data written in FLEX_TWI_THR (according to the register access size) is bigger than the Transmit FIFO free space, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in FLEX_TWI_FSR.

- Receive FIFO:

  In Multiple Data mode, if the number of data read in FLEX_TWI_RHR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it generates a Receive FIFO pointer error and sets the RXFPTEF flag in FLEX_TWI_FSR.

  Pointer error should not happen if the FIFO state is checked before writing/reading in the FLEX_TWI_THR/FLEX_TWI_RHR registers. The FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a Transmit or Receive pointer error occurs, a software reset must be performed using the SWRST bit in FLEX_TWI_CR. Note that issuing a software while transmitting might leave a slave in an unknown state holding the TWD line. In such case, a Bus Clear Command will allow to make the slave release the TWD line (the first frame sent afterwards might not be received properly by the slave).

#### FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

  The Transmit FIFO threshold can be set using the TXFTHRES field in FLEX_TWI_FMR. Each time the Transmit FIFO goes from the ‘above threshold’ to the ‘equal or below threshold’ state, the TXFTHF flag in FLEX_TWI_FSR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

  The Receive FIFO threshold can be set using the RXFTHRES field in FLEX_TWI_FMR. Each time the Receive FIFO goes from the ‘below threshold’ to the ‘equal to or above threshold’ state, the RXFTHF flag in FLEX_TWI_FSR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using FLEX_TWI_FIER and FLEX_TWI_FIDR.

#### FIFO Flags

FIFOs come with a set of flags which can be configured each to generate an interrupt through FLEX_TWI_FIER and FLEX_TWI_FIDR.

FIFO flags state can be read in FLEX_TWI_FSR. They are cleared on FLEX_TWI_FSR read.

### 44.9.6 TWI Comparison Function on Received Character

The comparison function differs if the asynchronous partial wakeup (Sleepwalking) is enabled or not.

If asynchronous partial wakeup is disabled (see PMC section), the TWI has the capability to extend the address matching on up to three slave addresses. The SADR1EN, SADR2EN and SADR3EN bits in FLEX_TWI_SMR enable address matching on additional addresses which can be configured through SADR1, SADR2 and SADR3 fields in FLEX_TWI_SWMR. The DATAMEN bit had no effect.

The SVACC bit is set when there is a comparison match with the received slave address.

### 44.9.7 TWI Register Write Protection

The FLEXCOM operating mode (FLEX_MR.OPMODE) must be set to FLEX_MR_OPMODE_TWI to enable access to the write protection registers.

To prevent any single software error from corrupting TWI behavior, certain registers in the address space can be write-protected by setting the WPEN (Write Protection Enable) bit in the TWI Write Protection Mode Register (FLEX_TWI_WPMR).

If a write access to a write-protected register is detected, the Write Protection Violation Status (WPVS) flag in the TWI Write Protection Status Register (FLEX_TWI_WPSR) is set and the Write Protection Violation Source (WPVSRC) field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading FLEX_TWI_WPSR.

The following register(s) can be write-protected when WPEN is set:

- TWI Slave Mode Register
- TWI Clock Waveform Generator Register
- TWI SMBus Timing Register
- TWI SleepWalking Matching Register

## 44.10 Flexible Serial Communication Unit (FLEXCOM) User Interface

**Table 44-18. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x000 | FLEXCOM Mode Register | FLEX_MR | Read/Write | 0x0 |
| 0x004–0x00C | Reserved | – | – | – |
| 0x010 | FLEXCOM Receive Holding Register | FLEX_RHR | Read-only | 0x0 |
| 0x014–0x01C | Reserved | – | – | – |
| 0x020 | FLEXCOM Transmit Holding Register | FLEX_THR | Read/Write | 0x0 |
| 0x024–0x0FC | Reserved | – | – | – |
| 0x100–0x1FC | Reserved | – | – | – |
| 0x200 | USART Control Register | FLEX_US_CR | Write-only | – |
| 0x204 | USART Mode Register | FLEX_US_MR | Read/Write | – |
| 0x208 | USART Interrupt Enable Register | FLEX_US_IER | Write-only | – |
| 0x20C | USART Interrupt Disable Register | FLEX_US_IDR | Write-only | – |
| 0x210 | USART Interrupt Mask Register | FLEX_US_IMR | Read-only | 0x0 |
| 0x214 | USART Channel Status Register | FLEX_US_CSR | Read-only | – |
| 0x218 | USART Receive Holding Register | FLEX_US_RHR | Read-only | 0x0 |
| 0x21C | USART Transmit Holding Register | FLEX_US_THR | Write-only | – |
| 0x220 | USART Baud Rate Generator Register | FLEX_US_BRGR | Read/Write | 0x0 |
| 0x224 | USART Receiver Timeout Register | FLEX_US_RTOR | Read/Write | 0x0 |
| 0x228 | USART Transmitter Timeguard Register | FLEX_US_TTGR | Read/Write | 0x0 |
| 0x22C–0x23C | Reserved | – | – | – |
| 0x240 | USART FI DI Ratio Register | FLEX_US_FIDI | Read/Write | 0x174 |
| 0x244 | USART Number of Errors Register | FLEX_US_NER | Read-only | – |
| 0x248 | Reserved | – | – | – |
| 0x24C | USART IrDA Filter Register | FLEX_US_IF | Read/Write | 0x0 |
| 0x250 | USART Manchester Configuration Register | FLEX_US_MAN | Read/Write | 0xB0011004 |
| 0x254 | USART LIN Mode Register | FLEX_US_LINMR | Read/Write | 0x0 |
| 0x258 | USART LIN Identifier Register | FLEX_US_LINIR | Read/Write(1) | 0x0 |
| 0x25C | USART LIN Baud Rate Register | FLEX_US_LINBRR | Read-only | 0x0 |
| 0x260–0x288 | Reserved | – | – | – |
| 0x290 | USART Comparison Register | FLEX_US_CMPR | Read/Write | 0x0 |
| 0x2A0 | USART FIFO Mode Register | FLEX_US_FMR | Read/Write | 0x0 |
| 0x2A4 | USART FIFO Level Register | FLEX_US_FLR | Read-only | 0x0 |
| 0x2A8 | USART FIFO Interrupt Enable Register | FLEX_US_FIER | Write-only | – |
| 0x2AC | USART FIFO Interrupt Disable Register | FLEX_US_FIDR | Write-only | – |
| 0x2B0 | USART FIFO Interrupt Mask Register | FLEX_US_FIMR | Read-only | 0x0 |
| 0x2B4 | USART FIFO Event Status Register | FLEX_US_FESR | Read-only | 0x0 |

**Table 44-18. Register Mapping (Continued)**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x2B8–0x2E0 | Reserved | – | – | – |
| 0x2E4 | USART Write Protection Mode Register | FLEX_US_WPMR | Read/Write | 0x0 |
| 0x2E8 | USART Write Protection Status Register | FLEX_US_WPSR | Read-only | 0x0 |
| 0x2EC–0x2F8 | Reserved | – | – | – |
| 0x300–0x3FC | Reserved | – | – | – |
| 0x400 | SPI Control Register | FLEX_SPI_CR | Write-only | – |
| 0x404 | SPI Mode Register | FLEX_SPI_MR | Read/Write | 0x0 |
| 0x408 | SPI Receive Data Register | FLEX_SPI_RDR | Read-only | 0x0 |
| 0x40C | SPI Transmit Data Register | FLEX_SPI_TDR | Write-only | – |
| 0x410 | SPI Status Register | FLEX_SPI_SR | Read-only | 0x0 |
| 0x414 | SPI Interrupt Enable Register | FLEX_SPI_IER | Write-only | – |
| 0x418 | SPI Interrupt Disable Register | FLEX_SPI_IDR | Write-only | – |
| 0x41C | SPI Interrupt Mask Register | FLEX_SPI_IMR | Read-only | 0x0 |
| 0x420–0x42C | Reserved | – | – | – |
| 0x430 | SPI Chip Select Register 0 | FLEX_SPI_CSR0 | Read/Write | 0x0 |
| 0x434 | SPI Chip Select Register 1 | FLEX_SPI_CSR1 | Read/Write | 0x0 |
| 0x438–0x43C | Reserved | – | – | – |
| 0x440 | SPI FIFO Mode Register | FLEX_SPI_FMR | Read/Write | 0x0 |
| 0x444 | SPI FIFO Level Register | FLEX_SPI_FLR | Read-only | 0x0 |
| 0x448 | SPI Comparison Register | FLEX_SPI_CMPR | Read/Write | 0x0 |
| 0x44C–0x4E0 | Reserved | – | – | – |
| 0x4E4 | SPI Write Protection Mode Register | FLEX_SPI_WPMR | Read/Write | 0x0 |
| 0x4E8 | SPI Write Protection Status Register | FLEX_SPI_WPSR | Read-only | 0x0 |
| 0x4EC–0x4F8 | Reserved | – | – | – |
| 0x500–0x5FC | Reserved | – | – | – |
| 0x600 | TWI Control Register | FLEX_TWI_CR | Write-only | – |
| 0x604 | TWI Master Mode Register | FLEX_TWI_MMR | Read/Write | 0x00000000 |
| 0x608 | TWI Slave Mode Register | FLEX_TWI_SMR | Read/Write | 0x00000000 |
| 0x60C | TWI Internal Address Register | FLEX_TWI_IADR | Read/Write | 0x00000000 |
| 0x610 | TWI Clock Waveform Generator Register | FLEX_TWI_CWGR | Read/Write | 0x00000000 |
| 0x614–0x61C | Reserved | – | – | – |
| 0x620 | TWI Status Register | FLEX_TWI_SR | Read-only | 0x0300F009 |
| 0x624 | TWI Interrupt Enable Register | FLEX_TWI_IER | Write-only | – |
| 0x628 | TWI Interrupt Disable Register | FLEX_TWI_IDR | Write-only | – |
| 0x62C | TWI Interrupt Mask Register | FLEX_TWI_IMR | Read-only | 0x00000000 |
| 0x630 | TWI Receive Holding Register | FLEX_TWI_RHR | Read-only | 0x00000000 |
| 0x634 | TWI Transmit Holding Register | FLEX_TWI_THR | Write-only | – |

**Table 44-18. Register Mapping (Continued)**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x638 | TWI SMBus Timing Register | FLEX_TWI_SMBTR | Read/Write | 0x00000000 |
| 0x63C | Reserved | – | – | – |
| 0x640 | TWI Alternative Command Register | FLEX_TWI_ACR | Read/Write | 0x0 |
| 0x644 | TWI Filter Register | FLEX_TWI_FILTR | Read/Write | 0x00000000 |
| 0x648 | Reserved | – | – | – |
| 0x64C | TWI SleepWalking Matching Register | FLEX_TWI_SWMR | Read/Write | 0x00000000 |
| 0x650 | TWI FIFO Mode Register | FLEX_TWI_FMR | Read/Write | 0x0 |
| 0x654 | TWI FIFO Level Register | FLEX_TWI_FLR | Read-only | 0x0 |
| 0x658–0x65C | Reserved | – | – | – |
| 0x660 | TWI FIFO Status Register | FLEX_TWI_FSR | Read-only | 0x0 |
| 0x664 | TWI FIFO Interrupt Enable Register | FLEX_TWI_FIER | Write-only | – |
| 0x668 | TWI FIFO Interrupt Disable Register | FLEX_TWI_FIDR | Write-only | – |
| 0x66C | TWI FIFO Interrupt Mask Register | FLEX_TWI_FIMR | Read-only | 0x0 |
| 0x670–0x6CC | Reserved | – | – | – |
| 0x6D0 | Reserved | – | – | – |
| 0x6D4–0x6E0 | Reserved | – | – | – |
| 0x6E4 | TWI Write Protection Mode Register | FLEX_TWI_WPMR | Read/Write | 0x00000000 |
| 0x6E8 | TWI Write Protection Status Register | FLEX_TWI_WPSR | Read-only | 0x00000000 |
| 0x6EC–0x6FC | Reserved | – | – | – |
| 0x700–0x7FC | Reserved | – | – | – |

Notes: 1. Write is possible only in LIN master node configuration.

### 44.10.1 FLEXCOM Mode Register

**Name:** FLEX_MR

**Address:** 0xF8034000 (0), 0xF8038000 (1), 0xFC010000 (2), 0xFC014000 (3), 0xFC018000 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | OPMODE | |

- **OPMODE: FLEXCOM Operating Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NO_COM | No communication |
| 1 | USART | All UART related protocols are selected (RS232, RS485, IrDA, ISO7816, LIN,)<br>SPI/TWI related registers are not accessible and have no impact on IOs. |
| 2 | SPI | SPI operating mode is selected.<br>USART/TWI related registers are not accessible and have no impact on IOs. |
| 3 | TWI | All TWI related protocols are selected (TWI, SMBus).<br>USART/SPI related registers are not accessible and have no impact on IOs. |

### 44.10.2 FLEXCOM Transmit Holding Register

**Name:** FLEX_THR

**Address:** 0xF8034020 (0), 0xF8038020 (1), 0xFC010020 (2), 0xFC014020 (3), 0xFC018020 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXDATA | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA | | | | | | | |

- **TXDATA: Transmit Data**

This register is a mirror of:

- USART Transmit Holding Register (FLEX_US_THR) if FLEX_MR.OPMODE field equals 1
- SPI Transmit Data Register (FLEX_SPI_TDR) if FLEX_MR.OPMODE field equals 2
- TWI Transmit Holding Register (FLEX_TWI_THR) if FLEX_MR.OPMODE field equals 3

### 44.10.3 FLEXCOM Receive Holding Register

**Name:** FLEX_RHR

**Address:** 0xF8034010 (0), 0xF8038010 (1), 0xFC010010 (2), 0xFC014010 (3), 0xFC018010 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXDATA | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA | | | | | | | |

- **RXDATA: Receive Data**

This register is a mirror of:

- USART Receive Holding Register (FLEX_US_RHR) if FLEX_MR.OPMODE field equals 1
- SPI Receive Data Register (FLEX_SPI_RDR) if FLEX_MR.OPMODE field equals 2
- TWI Transmit Holding Register (FLEX_TWI_RHR) if FLEX_MR.OPMODE field equals 3

### 44.10.4 USART Control Register

**Name:** FLEX_US_CR

**Address:** 0xF8034200 (0), 0xF8038200 (1), 0xFC010200 (2), 0xFC014200 (3), 0xFC018200 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FIFODIS | FIFOEN | – | REQCLR | – | TXFLCLR | RXFCLR | TXFCLR |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | LINWKUP | LINABT | RTSDIS | RTSEN | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RETTO | RSTNACK | RSTIT | SENDA | STTTO | STPBRK | STTBRK | RSTSTA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDIS | TXEN | RXDIS | RXEN | RSTTX | RSTRX | – | – |

For SPI control, see Section 44.10.5 “USART Control Register (SPI_MODE)”.

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, MANE, LINBE, LINISFE, LINIPE, LINCE, LINSNRE, LINSTE, LINHTE, LINID, LINTC, LINBK, CMP and RXBRK in FLEX_US_CSR. Also resets the status bits TXFEF, TXFFF, TXFTHF, RXFEF, RXFFF, RXFTHF, TXFPTEF, RXFPTEF in FLEX_US_FESR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in FLEX_US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Clear TIMEOUT Flag and Start Timeout After Next Character Received**

0: No effect.

1: Starts waiting for a character before clocking the timeout counter. Immediately disables a timeout period in progress. Resets the status bit TIMEOUT in FLEX_US_CSR.

- **SENDA: Send Address**

0: No effect.

1: In Multidrop mode only, the next character written to FLEX_US_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITER in FLEX_US_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in FLEX_US_CSR.

- **RETTO: Start Timeout Immediately**

0: No effect

1: Immediately restarts timeout period.

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the RTS pin to 1 if FLEX_US_MR.USART_MODE field = 2, else drives the RTS pin to 0 if FLEX_US_MR.USART_MODE field = 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the RTS pin to 0 if FLEX_US_MR.USART_MODE field = 2, else drives the RTS pin to 1 if FLEX_US_MR.USART_MODE field = 0.

- **LINABT: Abort LIN Transmission**

0: No effect.

1: Aborts the current LIN transmission.

- **LINWKUP: Send LIN Wakeup Signal**

0: No effect:

1: Sends a wakeup signal on the LIN bus.

- **TXFCLR: Transmit FIFO Clear**

0: No effect.

1: Clears the Transmit FIFO, Transmit FIFO will become empty.

- **RXFCLR: Receive FIFO Clear**

0: No effect.

1: Clears the Receive FIFO, Receive FIFO will become empty.

- **TXFLCLR: Transmit FIFO Lock CLEAR**

0: No effect.

1: Clears the Transmit FIFO Lock

- **REQCLR: Request to Clear the Comparison Trigger**

- **FIFOEN: FIFO Enable**

0: No effect.

1: Enables the Transmit and Receive FIFOs

- **FIFODIS: FIFO Disable**

0: No effect.

1: Disables the Transmit and Receive FIFOs

### 44.10.5 USART Control Register (SPI_MODE)

**Name:** FLEX_US_CR (SPI_MODE)

**Address:** 0xF8034200 (0), 0xF8038200 (1), 0xFC010200 (2), 0xFC014200 (3), 0xFC018200 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | RCS | FCS | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | RSTSTA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDIS | TXEN | RXDIS | RXEN | RSTTX | RSTRX | – | – |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits OVRE, UNRE in FLEX_US_CSR.

- **FCS: Force SPI Chip Select**

Applicable if USART operates in SPI Master mode (USART_MODE = 0xE):

0: No effect.

1: Forces the Slave Select Line NSS (RTS pin) to 0, even if USART is not transmitting, in order to address SPI slave devices supporting the CSAAT mode (Chip Select Active After Transfer).

- **RCS: Release SPI Chip Select**

Applicable if USART operates in SPI Master mode (USART_MODE = 0xE):

0: No effect.

1: Releases the Slave Select Line NSS (RTS pin).

### 44.10.6 USART Mode Register

**Name:** FLEX_US_MR

**Address:** 0xF8034204 (0), 0xF8038204 (1), 0xFC010204 (2), 0xFC014204 (3), 0xFC018204 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ONEBIT | MODSYNC | MAN | FILTER | – | MAX_ITERATION | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| INVDATA | VAR_SYNC | DSNACK | INACK | OVER | CLKO | MODE9 | MSBF |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CHMODE | | NBSTOP | | PAR | | | SYNC |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CHRL | | USCLKS | | USART_MODE | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

For SPI configuration, see Section 44.10.7 “USART Mode Register (SPI_MODE)”.

- **USART_MODE: USART Mode of Operation**

| Value | Name | Description |
|---|---|---|
| 0x0 | NORMAL | Normal mode |
| 0x1 | RS485 | RS485 |
| 0x2 | HW_HANDSHAKING | Hardware handshaking |
| 0x4 | IS07816_T_0 | IS07816 Protocol: T = 0 |
| 0x6 | IS07816_T_1 | IS07816 Protocol: T = 1 |
| 0x8 | IRDA | IrDA |
| 0xA | LIN_MASTER | LIN Master mode |
| 0xB | LIN_SLAVE | LIN Slave mode |
| 0xE | SPI_MASTER | SPI Master mode (CLKO must be written to 1 and USCLKS = 0, 1 or 2) |
| 0xF | SPI_SLAVE | SPI Slave mode |

- **USCLKS: Clock Selection**

| Value | Name | Description |
|---|---|---|
| 0 | MCK | Peripheral clock is selected |
| 1 | DIV | Peripheral clock divided (DIV = 8) is selected |
| 2 | GCLK | PMC generic clock is selected. If the SCK pin is driven (CLKO = 1), the CD field must be greater than 1. |
| 3 | SCK | External pin SCK is selected |

- **CHRL: Character Length**

| Value | Name | Description |
|---|---|---|
| 0 | 5_BIT | Character length is 5 bits |
| 1 | 6_BIT | Character length is 6 bits |
| 2 | 7_BIT | Character length is 7 bits |
| 3 | 8_BIT | Character length is 8 bits |

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous mode (UART).

1: USART operates in Synchronous mode.

- **PAR: Parity Type**

| Value | Name | Description |
|---|---|---|
| 0 | EVEN | Even parity |
| 1 | ODD | Odd parity |
| 2 | SPACE | Parity forced to 0 (Space) |
| 3 | MARK | Parity forced to 1 (Mark) |
| 4 | NO | No parity |
| 6 | MULTIDROP | Multidrop mode |

- **NBSTOP: Number of Stop Bits**

| Value | Name | Description |
|---|---|---|
| 0 | 1_BIT | 1 stop bit |
| 1 | 1_5_BIT | 1.5 stop bit (SYNC = 0) or reserved (SYNC = 1) |
| 2 | 2_BIT | 2 stop bits |

- **CHMODE: Channel Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal mode |
| 1 | AUTOMATIC | Automatic Echo. Receiver input is connected to the TXD pin. |
| 2 | LOCAL_LOOPBACK | Local Loopback. Transmitter output is connected to the Receiver Input. |
| 3 | REMOTE_LOOPBACK | Remote Loopback. RXD pin is internally connected to the TXD pin. |

- **MSBF: Bit Order**

0: Least significant bit is sent/received first.

1: Most significant bit is sent/received first.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin (Synchronous Slave mode or Asynchronous mode with external baud rate clock source).

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK (USART Synchronous Master mode).

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITER is asserted.

Note: The MAX_ITERATION field must be cleared if DSNACK is cleared.

- **INVDATA: Inverted Data**

0: The data field transmitted on TXD line is the same as the one written in FLEX_US_THR or the content read in FLEX_US_RHR is the same as RXD line. Normal mode of operation.

1: The data field transmitted on TXD line is inverted (voltage polarity only) compared to the value written in FLEX_US_THR or the content read in FLEX_US_RHR is inverted compared to what is received on RXD line (or ISO7816 IO line). Inverted mode of operation, useful for contactless card application. To be used with configuration bit MSBF.

- **VAR_SYNC: Variable Synchronization of Command/Data Sync Start Frame Delimiter**

0: User defined configuration of command or data sync field depending on MODSYNC value.

1: The sync field is updated when a character is written into FLEX_US_THR.

- **MAX_ITERATION: Maximum Number of Automatic Iteration**

0–7: Defines the maximum number of iterations in mode ISO7816, protocol T = 0.

- **FILTER: Receive Line Filter**

0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

- **MAN: Manchester Encoder/Decoder Enable**

0: Manchester encoder/decoder are disabled.

1: Manchester encoder/decoder are enabled.

- **MODSYNC: Manchester Synchronization Mode**

0:The Manchester start bit is a 0 to 1 transition

1: The Manchester start bit is a 1 to 0 transition.

- **ONEBIT: Start Frame Delimiter Selector**

0: Start frame delimiter is COMMAND or DATA SYNC.

1: Start frame delimiter is one bit.

### 44.10.7 USART Mode Register (SPI_MODE)

**Name:** FLEX_US_MR (SPI_MODE)

**Address:** 0xF8034204 (0), 0xF8038204 (1), 0xFC010204 (2), 0xFC014204 (3), 0xFC018204 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | WRDBT | – | – | – | CPOL |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | CPHA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CHRL | | USCLKS | | USART_MODE | | | |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **USART_MODE: USART Mode of Operation**

| Value | Name | Description |
|---|---|---|
| 0xE | SPI_MASTER | SPI master |
| 0xF | SPI_SLAVE | SPI slave |

- **USCLKS: Clock Selection**

| Value | Name | Description |
|---|---|---|
| 0 | MCK | Peripheral clock is selected |
| 1 | DIV | Peripheral clock Divided (DIV= 8) is selected |
| 2 | GCLK | A PMC generic clock is selected |
| 3 | SCK | External pin SCK is selected |

- **CHRL: Character Length**

| Value | Name | Description |
|---|---|---|
| 3 | 8_BIT | Character length is 8 bits |

- **CPHA: SPI Clock Phase**
  – Applicable if USART operates in SPI mode (USART_MODE = 0xE or 0xF):

0: Data are changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data are captured on the leading edge of SPCK and changed on the following edge of SPCK.

CPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CHMODE: Channel Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal mode |
| 1 | AUTOMATIC | Automatic Echo mode. Receiver input is connected to the TXD pin. |
| 2 | LOCAL_LOOPBACK | Local Loopback mode. Transmitter output is connected to the Receiver Input. |
| 3 | REMOTE_LOOPBACK | Remote Loopback mode. RXD pin is internally connected to the TXD pin. |

- **CPOL: SPI Clock Polarity**

Applicable if USART operates in SPI mode (slave or master, USART_MODE = 0xE or 0xF):

0: The inactive state value of SPCK is logic level zero.

1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with CPHA to produce the required clock/data relationship between master and slave devices.

- **WRDBT: Wait Read Data Before Transfer**

0: The character transmission starts as soon as a character is written into FLEX_US_THR (assuming TXRDY was set).

1: The character transmission starts when a character is written and only if RXRDY flag is cleared (Receive Holding Register has been read).

### 44.10.8 USART Interrupt Enable Register

**Name:** FLEX_US_IER

**Address:** 0xF8034208 (0), 0xF8038208 (1), 0xFC010208 (2), 0xFC014208 (3), 0xFC018208 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | MANE |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | – | – | ITER | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | RXBRK | TXRDY | RXRDY |

For SPI-s-specific configurations, see Section 44.10.9 “USART Interrupt Enable Register (SPI_MODE)”.

For LIN-specific configurations, see Section 44.10.10 “USART Interrupt Enable Register (LIN_MODE)”.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Timeout Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITER: Max number of Repetitions Reached Interrupt Enable**
- **NACK: Non Acknowledge Interrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **CMP: Comparison Interrupt Enable**
- **MANE: Manchester Error Interrupt Enable**

### 44.10.9 USART Interrupt Enable Register (SPI_MODE)

**Name:** FLEX_US_IER (SPI_MODE)

**Address:** 0xF8034208 (0), 0xF8038208 (1), 0xFC010208 (2), 0xFC014208 (3), 0xFC018208 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | NSSE | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNRE | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **UNRE: SPI Underrun Error Interrupt Enable**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event**
- **CMP: Comparison Interrupt Enable**

### 44.10.10 USART Interrupt Enable Register (LIN_MODE)

**Name:** FLEX_US_IER (LIN_MODE)

**Address:** 0xF8034208 (0), 0xF8038208 (1), 0xFC010208 (2), 0xFC014208 (3), 0xFC018208 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| LINHTE | LINSTE | LINSNRE | LINCE | LINIPE | LINISFE | LINBE | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| LINTC | LINID | LINBK | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xA or 0xB in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Enables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Timeout Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Enable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Enable**
- **LINTC: LIN Transfer Completed Interrupt Enable**
- **LINBE: LIN Bus Error Interrupt Enable**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Enable**
- **LINIPE: LIN Identifier Parity Interrupt Enable**
- **LINCE: LIN Checksum Error Interrupt Enable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Enable**

- **LINSTE: LIN Synch Tolerance Error Interrupt Enable**
- **LINHTE: LIN Header Timeout Error Interrupt Enable**

### 44.10.11 USART Interrupt Disable Register

**Name:** FLEX_US_IDR

**Address:** 0xF803420C (0), 0xF803820C (1), 0xFC01020C (2), 0xFC01420C (3), 0xFC01820C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | MANE |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | – | – | ITER | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | RXBRK | TXRDY | RXRDY |

For SPI-specific configurations, see Section 44.10.12 “USART Interrupt Disable Register (SPI_MODE)”.

For LIN-specific configurations, see Section 44.10.13 “USART Interrupt Disable Register (LIN_MODE)”.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Timeout Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITER: Max Number of Repetitions Reached Interrupt Disable**
- **NACK: Non Acknowledge Interrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **CMP: Comparison Interrupt Disable**
- **MANE: Manchester Error Interrupt Disable**

### 44.10.12 USART Interrupt Disable Register (SPI_MODE)

**Name:** FLEX_US_IDR (SPI_MODE)

**Address:** 0xF803420C (0), 0xF803820C (1), 0xFC01020C (2), 0xFC01420C (3), 0xFC01820C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | NSSE | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNRE | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **UNRE: SPI Underrun Error Interrupt Disable**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event**
- **CMP: Comparison Interrupt Disable**

### 44.10.13 USART Interrupt Disable Register (LIN_MODE)

**Name:** FLEX_US_IDR (LIN_MODE)

**Address:** 0xF803420C (0), 0xF803820C (1), 0xFC01020C (2), 0xFC01420C (3), 0xFC01820C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| LINHTE | LINSTE | LINSNRE | LINCE | LINIPE | LINISFE | LINBE | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| LINTC | LINID | LINBK | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xA or 0xB in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: No effect

1: Disables the corresponding interrupt.

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Timeout Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Disable**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Disable**
- **LINTC: LIN Transfer Completed Interrupt Disable**
- **LINBE: LIN Bus Error Interrupt Disable**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Disable**
- **LINIPE: LIN Identifier Parity Interrupt Disable**
- **LINCE: LIN Checksum Error Interrupt Disable**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Disable**

- **LINSTE: LIN Synch Tolerance Error Interrupt Disable**
- **LINHTE: LIN Header Timeout Error Interrupt Disable**

### 44.10.14 USART Interrupt Mask Register

**Name:** FLEX_US_IMR

**Address:** 0xF8034210 (0), 0xF8038210 (1), 0xFC010210 (2), 0xFC014210 (3), 0xFC018210 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | MANE |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | – | – | ITER | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | RXBRK | TXRDY | RXRDY |

For SPI-specific configurations, see Section 44.10.15 “USART Interrupt Mask Register (SPI_MODE)”.

For LIN-specific configurations, see Section 44.10.16 “USART Interrupt Mask Register (LIN_MODE)”.

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Timeout Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITER: Max Number of Repetitions Reached Interrupt Mask**
- **NACK: Non Acknowledge Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **CMP: Comparison Interrupt Mask**
- **MANE: Manchester Error Interrupt Mask**

### 44.10.15 USART Interrupt Mask Register (SPI_MODE)

**Name:** FLEX_US_IMR (SPI_MODE)

**Address:** 0xF8034210 (0), 0xF8038210 (1), 0xFC010210 (2), 0xFC014210 (3), 0xFC018210 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | CMP | – | – | NSSE | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNRE | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **UNRE: SPI Underrun Error Interrupt Mask**
- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event**
- **CMP: Comparison Interrupt Mask**

### 44.10.16 USART Interrupt Mask Register (LIN_MODE)

**Name:** FLEX_US_IMR (LIN_MODE)

**Address:** 0xF8034210 (0), 0xF8038210 (1), 0xFC010210 (2), 0xFC014210 (3), 0xFC018210 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| LINHTE | LINSTE | LINSNRE | LINCE | LINIPE | LINISFE | LINBE | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| LINTC | LINID | LINBK | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xA or 0xB in the USART Mode Register.

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Timeout Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **LINBK: LIN Break Sent or LIN Break Received Interrupt Mask**
- **LINID: LIN Identifier Sent or LIN Identifier Received Interrupt Mask**
- **LINTC: LIN Transfer Completed Interrupt Mask**
- **LINBE: LIN Bus Error Interrupt Mask**
- **LINISFE: LIN Inconsistent Synch Field Error Interrupt Mask**
- **LINIPE: LIN Identifier Parity Interrupt Mask**
- **LINCE: LIN Checksum Error Interrupt Mask**
- **LINSNRE: LIN Slave Not Responding Error Interrupt Mask**

- **LINSTE: LIN Synch Tolerance Error Interrupt Mask**
- **LINHTE: LIN Header Timeout Error Interrupt Mask**

### 44.10.17 USART Channel Status Register

**Name:** FLEX_US_CSR

**Address:** 0xF8034214 (0), 0xF8038214 (1), 0xFC010214 (2), 0xFC014214 (3), 0xFC018214 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | MANE |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| CTS | CMP | – | – | CTSIC | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | NACK | – | – | ITER | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | RXBRK | TXRDY | RXRDY |

For SPI-specific configurations, see Section 44.10.18 “USART Channel Status Register (SPI_MODE)”.

For LIN-specific configurations, see Section 44.10.19 “USART Channel Status Register (LIN_MODE)”.

- **RXRDY: Receiver Ready (cleared by reading FLEX_US_RHR)**

When FIFOs are disabled:

0: No complete character has been received since the last read of FLEX_US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and FLEX_US_RHR has not yet been read.

When FIFOs are enabled:

0: Receive FIFO is empty; no data to read

1: At least one unread data is in the Receive FIFO

RXRDY behavior with FIFO enabled is illustrated in Section 44.7.11.4 “TXRDY, RXRDY and TXEMPTY Behavior”.

- **TXRDY: Transmitter Ready (cleared by writing FLEX_US_THR)**

When FIFOs are disabled:

0: A character in FLEX_US_THR is waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in FLEX_US_THR.

When FIFOs are enabled:

0: Transmit FIFO is full and cannot accept more data

1: Transmit FIFO is not full; one or more data can be written according to TXRDYM field configuration

TXRDY behavior with FIFO enabled is illustrated in Section 44.7.11.4 “TXRDY, RXRDY and TXEMPTY Behavior”.

- **RXBRK: Break Received/End of Break**

0: No break received or end of break detected since the last RSTSTA command was issued.

1: Break received or end of break detected since the last RSTSTA command was issued.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA command was issued.

1: At least one overrun error has occurred since the last RSTSTA command was issued.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA command was issued.

1: At least one stop bit has been detected low since the last RSTSTA command was issued.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA command was issued.

1: At least one parity error has been detected since the last RSTSTA command was issued.

- **TIMEOUT: Receiver Timeout**

0: There has not been a timeout since the last Start Timeout command (STTTO in FLEX_US_CR) or the Timeout Register is 0.

1: There has been a timeout since the last Start Timeout command (STTTO in FLEX_US_CR).

- **TXEMPTY: Transmitter Empty (cleared by writing FLEX_US_THR)**

0: There are characters in either FLEX_US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in FLEX_US_THR, nor in the Transmit Shift Register.

- **ITER: Max Number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSTIT command was issued.

1: Maximum number of repetitions has been reached since the last RSTIT command was issued.

- **NACK: Non Acknowledge Interrupt**

0: Non acknowledge has not been detected since the last RSTNACK.

1: At least one non acknowledge has been detected since the last RSTNACK.

- **CTSIC: Clear to Send Input Change Flag**

0: No input change has been detected on the CTS pin since the last read of FLEX_US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of FLEX_US_CSR.

- **CMP: Comparison Status**

0: No received character matched the comparison criteria programmed in VAL1, VAL2 fields and CMPPAR bit in since the last RSTSTA command was issued.

1: A received character matched the comparison criteria since the last RSTSTA command was issued.

- **CTS: Image of CTS Input**

0: CTS input is driven low.

1: CTS input is driven high.

- **MANE: Manchester Error**

0: No Manchester error has been detected since the last RSTSTA command was issued.

1: At least one Manchester error has been detected since the last RSTSTA command was issued.

### 44.10.18 USART Channel Status Register (SPI_MODE)

**Name:** FLEX_US_CSR (SPI_MODE)

**Address:** 0xF8034214 (0), 0xF8038214 (1), 0xFC010214 (2), 0xFC014214 (3), 0xFC018214 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| NSS | CMP | – | – | NSSE | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | UNRE | TXEMPTY | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xE or 0xF in the USART Mode Register.

- **RXRDY: Receiver Ready (cleared by reading FLEX_US_RHR)**

0: No complete character has been received since the last read of FLEX_US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and FLEX_US_RHR has not yet been read.

- **TXRDY: Transmitter Ready (cleared by writing FLEX_US_THR)**

0: A character in FLEX_US_THR is waiting to be transferred to the Transmit Shift Register, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in FLEX_US_THR.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA command was issued.

1: At least one overrun error has occurred since the last RSTSTA command was issued.

- **TXEMPTY: Transmitter Empty (cleared by writing FLEX_US_THR)**

0: There are characters in either FLEX_US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in FLEX_US_THR, nor in the Transmit Shift Register.

- **UNRE: Underrun Error**

0: No SPI underrun error has occurred since the last RSTSTA command was issued.

1: At least one SPI underrun error has occurred since the last RSTSTA command was issued.

- **NSSE: NSS Line (Driving CTS Pin) Rising or Falling Edge Event (cleared on read)**

0: No NSS line event has been detected since the last read of FLEX_US_CSR.

1: A rising or falling edge has been detected on the NSS line since the last read of FLEX_US_CSR.

- **CMP: Comparison Match**

0: No received character matched the comparison criteria programmed in VAL1, VAL2 fields and CMPPAR bit in FLEX_US_CMPR since the last RSTSTA command was issued.

1: A received character matched the comparison criteria since the last RSTSTA command was issued.

- **NSS: Image of NSS Line**

0: NSS line is driven low (if NSSE = 1, falling edge occurred on NSS line).

1: NSS line is driven high (if NSSE = 1, rising edge occurred on NSS line).

### 44.10.19 USART Channel Status Register (LIN_MODE)

**Name:** FLEX_US_CSR (LIN_MODE)

**Address:** 0xF8034214 (0), 0xF8038214 (1), 0xFC010214 (2), 0xFC014214 (3), 0xFC018214 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| LINHTE | LINSTE | LINSNRE | LINCE | LINIPE | LINISFE | LINBE | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| LINBLS | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| LINTC | LINID | LINBK | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

This configuration is relevant only if USART_MODE = 0xA or 0xB in the USART Mode Register.

- **RXRDY: Receiver Ready (cleared by reading FLEX_US_RHR)**

0: No complete character has been received since the last read of FLEX_US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and FLEX_US_RHR has not yet been read.

- **TXRDY: Transmitter Ready (cleared by writing FLEX_US_THR)**

0: A character in FLEX_US_THR is waiting to be transferred to the Transmit Shift Register, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in FLEX_US_THR.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA command was issued.

1: At least one overrun error has occurred since the last RSTSTA command was issued.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA command was issued.

1: At least one stop bit has been detected low since the last RSTSTA command was issued.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA command was issued.

1: At least one parity error has been detected since the last RSTSTA command was issued.

- **TIMEOUT: Receiver Timeout**

0: There has not been a timeout since the last start timeout command (STTTO in FLEX_US_CR) or the Timeout Register is 0.

1: There has been a timeout since the last start timeout command (STTTO in FLEX_US_CR).

- **TXEMPTY: Transmitter Empty (cleared by writing FLEX_US_THR)**

0: There are characters in either FLEX_US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in FLEX_US_THR, nor in the Transmit Shift Register.

- **LINBK: LIN Break Sent or LIN Break Received**

Applicable if USART operates in LIN Master mode (USART_MODE = 0xA):

0: No LIN break has been sent since the last RSTSTA command was issued.

1:At least one LIN break has been sent since the last RSTSTA

If USART operates in LIN Slave mode (USART_MODE = 0xB):

0: No LIN break has received sent since the last RSTSTA command was issued.

1:At least one LIN break has been received since the last RSTSTA command was issued.

- **LINID: LIN Identifier Sent or LIN Identifier Received**

If USART operates in LIN Master mode (USART_MODE = 0xA):

0: No LIN identifier has been sent since the last RSTSTA command was issued.

1:At least one LIN identifier has been sent since the last RSTSTA command was issued.

If USART operates in LIN Slave mode (USART_MODE = 0xB):

0: No LIN identifier has been received since the last RSTSTA command was issued.

1:At least one LIN identifier has been received since the last RSTSTA

- **LINTC: LIN Transfer Completed**

0: The USART is idle or a LIN transfer is ongoing.

1: A LIN transfer has been completed since the last RSTSTA command was issued.

- **LINBLS: LIN Bus Line Status**

0: LIN bus line is set to 0.

1: LIN bus line is set to 1.

- **LINBE: LIN Bit Error**

0: No bit error has been detected since the last RSTSTA command was issued.

1: A bit error has been detected since the last RSTSTA command was issued.

- **LINISFE: LIN Inconsistent Synch Field Error**

0: No LIN inconsistent synch field error has been detected since the last RSTSTA

1: The USART is configured as a slave node and a LIN Inconsistent synch field error has been detected since the last RSTSTA command was issued.

- **LINIPE: LIN Identifier Parity Error**

0: No LIN identifier parity error has been detected since the last RSTSTA command was issued.

1: A LIN identifier parity error has been detected since the last RSTSTA command was issued.

- **LINCE: LIN Checksum Error**

0: No LIN checksum error has been detected since the last RSTSTA command was issued.

1: A LIN checksum error has been detected since the last RSTSTA command was issued.

- **LINSNRE: LIN Slave Not Responding Error**

0: No LIN slave not responding error has been detected since the last RSTSTA command was issued.

1: A LIN slave not responding error has been detected since the last RSTSTA command was issued.

- **LINSTE: LIN Synch Tolerance Error**

0: No LIN synch tolerance error has been detected since the last RSTSTA command was issued.

1: A LIN synch tolerance error has been detected since the last RSTSTA command was issued.

- **LINHTE: LIN Header Timeout Error**

0: No LIN header timeout error has been detected since the last RSTSTA command was issued.

1: A LIN header timeout error has been detected since the last RSTSTA command was issued.

### 44.10.20 USART Receive Holding Register

**Name:** FLEX_US_RHR

**Address:** 0xF8034218 (0), 0xF8038218 (1), 0xFC010218 (2), 0xFC014218 (3), 0xFC018218 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXSYNH | – | – | – | – | – | – | RXCHR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXCHR | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.7.11.5 “Single Data Mode” for details.

- **RXCHR: Received Character**

Last character received if RXRDY is set.

- **RXSYNH: Received Sync**

0: Last character received is a data.

1: Last character received is a command.

### 44.10.21 USART Receive Holding Register (FIFO Multi Data)

**Name:** FLEX_US_RHR (FIFO_MULTI_DATA)

**Address:** 0xF8034218 (0), 0xF8038218 (1), 0xFC010218 (2), 0xFC014218 (3), 0xFC018218 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXCHR3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RXCHR2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXCHR1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXCHR0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.7.11.6 “Multiple Data Mode” for details.

- **RXCHRx: Received Character**

First unread character in the Receive FIFO if RXRDY is set.

### 44.10.22 USART Transmit Holding Register

**Name:** FLEX_US_THR

**Address:** 0xF803421C (0), 0xF803821C (1), 0xFC01021C (2), 0xFC01421C (3), 0xFC01821C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXSYNH | – | – | – | – | – | – | TXCHR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXCHR | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.7.11.5 “Single Data Mode” for details.

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

- **TXSYNH: Sync Field to be Transmitted**

0: The next character sent is encoded as a data. Start frame delimiter is DATA SYNC.

1: The next character sent is encoded as a command. Start frame delimiter is COMMAND SYNC.

### 44.10.23 USART Transmit Holding Register (FIFO Multi Data)

**Name:** FLEX_US_THR (FIFO_MULTI_DATA)

**Address:** 0xF803421C (0), 0xF803821C (1), 0xFC01021C (2), 0xFC01421C (3), 0xFC01821C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TXCHR3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TXCHR2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXCHR1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXCHR0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.7.11.6 “Multiple Data Mode” for details.

- **TXCHRx: Character to be Transmitted**

Next character to be transmitted.

### 44.10.24 USART Baud Rate Generator Register

**Name:** FLEX_US_BRGR

**Address:** 0xF8034220 (0), 0xF8038220 (1), 0xFC010220 (2), 0xFC014220 (3), 0xFC018220 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | FP | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CD | | | | | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **CD: Clock Divider**

| | **USART_MODE ≠ ISO7816** | | | |
|---|---|---|---|---|
| | **SYNC = 0** | | **SYNC = 1 or USART_MODE = SPI (master or Slave)** | **USART_MODE = ISO7816** |
| **CD** | **OVER = 0** | **OVER = 1** | | |
| 0 | Baud Rate Clock Disabled | | | |
| 1 to 65535 | CD = Selected Clock / (16 × Baud Rate) | CD = Selected Clock / (8 × Baud Rate) | CD = Selected Clock / Baud Rate | CD = Selected Clock / (FI_DI_RATIO × Baud Rate) |

- **FP: Fractional Part**

0: Fractional divider is disabled.

1–7: Baud rate resolution, defined by FP × 1/8.

**Warning:** When the value of field FP is greater than 0, the SCK (oversampling clock) generates non-constant duty cycles. The SCK high duration is increased by “selected clock” period from time to time. The duty cycle depends on the value of the CD field.

### 44.10.25 USART Receiver Timeout Register

**Name:** FLEX_US_RTOR

**Address:** 0xF8034224 (0), 0xF8038224 (1), 0xFC010224 (2), 0xFC014224 (3), 0xFC018224 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | TO |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TO | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TO | | | | | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TO: Timeout Value**

0: The receiver timeout is disabled.

1–131071: The receiver timeout is enabled and the timeout delay is TO × bit period.

### 44.10.26 USART Transmitter Timeguard Register

**Name:** FLEX_US_TTGR

**Address:** 0xF8034228 (0), 0xF8038228 (1), 0xFC010228 (2), 0xFC014228 (3), 0xFC018228 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TG | | | | | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TG: Timeguard Value**

0: The transmitter timeguard is disabled.

1–255: The transmitter timeguard is enabled and TG is timeguard delay / bit period.

### 44.10.27 USART FI DI RATIO Register

**Name:** FLEX_US_FIDI

**Address:** 0xF8034240 (0), 0xF8038240 (1), 0xFC010240 (2), 0xFC014240 (3), 0xFC018240 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| FI_DI_RATIO | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FI_DI_RATIO | | | | | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the baud rate generator generates no signal.

1–2: Do not use.

3–65535: If ISO7816 mode is selected, the baud rate is the clock provided on SCK divided by FI_DI_RATIO.

### 44.10.28 USART Number of Errors Register

**Name:** FLEX_US_NER

**Address:** 0xF8034244 (0), 0xF8038244 (1), 0xFC010244 (2), 0xFC014244 (3), 0xFC018244 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NB_ERRORS | | | | | | | |

This register is relevant only if USART_MODE = 0x4 or 0x6 in the USART Mode Register.

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

### 44.10.29 USART IrDA FILTER Register

**Name:** FLEX_US_IF

**Address:** 0xF803424C (0), 0xF803824C (1), 0xFC01024C (2), 0xFC01424C (3), 0xFC01824C (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IRDA_FILTER | | | | | | | |

This register is relevant only if USART_MODE = 0x8 in the USART Mode Register.

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **IRDA_FILTER: IrDA Filter**

The IRDA_FILTER value must be defined to meet the following criteria:

$t_{peripheral\ clock} \times (IRDA\_FILTER + 3) < 1.41\ \mu s$

### 44.10.30 USART Manchester Configuration Register

**Name:** FLEX_US_MAN

**Address:** 0xF8034250 (0), 0xF8038250 (1), 0xFC010250 (2), 0xFC014250 (3), 0xFC018250 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXIDLEV | DRIFT | ONE | RX_MPOL | – | – | RX_PP | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | RX_PL | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | TX_MPOL | – | – | TX_PP | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | TX_PL | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **TX_PL: Transmitter Preamble Length**

0: The transmitter preamble pattern generation is disabled

1–15: The preamble length is TX_PL × Bit Period

- **TX_PP: Transmitter Preamble Pattern**

The following values assume that TX_MPOL field is not set:

| Value | Name | Description |
|---|---|---|
| 0 | ALL_ONE | The preamble is composed of ‘1’s |
| 1 | ALL_ZERO | The preamble is composed of ‘0’s |
| 2 | ZERO_ONE | The preamble is composed of ‘01’s |
| 3 | ONE_ZERO | The preamble is composed of ‘10’s |

- **TX_MPOL: Transmitter Manchester Polarity**

0: Logic zero is coded as a zero-to-one transition, Logic one is coded as a one-to-zero transition.

1: Logic zero is coded as a one-to-zero transition, Logic one is coded as a zero-to-one transition.

- **RX_PL: Receiver Preamble Length**

0: The receiver preamble pattern detection is disabled

1–15: The detected preamble length is RX_PL × Bit Period

- **RX_PP: Receiver Preamble Pattern detected**

The following values assume that RX_MPOL field is not set:

| Value | Name | Description |
|---|---|---|
| 0 | ALL_ONE | The preamble is composed of ‘1’s |
| 1 | ALL_ZERO | The preamble is composed of ‘0’s |
| 2 | ZERO_ONE | The preamble is composed of ‘01’s |
| 3 | ONE_ZERO | The preamble is composed of ‘10’s |

- **RX_MPOL: Receiver Manchester Polarity**

0: Logic zero is coded as a zero-to-one transition, Logic one is coded as a one-to-zero transition.

1: Logic zero is coded as a one-to-zero transition, Logic one is coded as a zero-to-one transition.

- **ONE: Must Be Set to 1**

Bit 29 must always be set to 1 when programming the FLEX_US_MAN register.

- **DRIFT: Drift Compensation**

0: The USART cannot recover from an important clock drift

1: The USART can recover from clock drift. The 16X Clock mode must be enabled.

- **RXIDLEV: Receiver Idle Value**

0: Receiver line idle value is 0.

1: Receiver line idle value is 1.

### 44.10.31 USART LIN Mode Register

**Name:** FLEX_US_LINMR

**Address:** 0xF8034254 (0), 0xF8038254 (1), 0xFC010254 (2), 0xFC014254 (3), 0xFC018254 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | SYNCDIS | PDCM |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| DLC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| WKUPTYP | FSDIS | DLM | CHKTYP | CHKDIS | PARDIS | NACT | |

This register is relevant only if USART_MODE = 0xA or 0xB in the USART Mode Register.

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **NACT: LIN Node Action**

| Value | Name | Description |
|---|---|---|
| 0 | PUBLISH | The USART transmits the response. |
| 1 | SUBSCRIBE | The USART receives the response. |
| 2 | IGNORE | The USART does not transmit and does not receive the response. |

Values which are not listed in the table must be considered as “reserved”.

- **PARDIS: Parity Disable**

0: In master node configuration, the identifier parity is computed and sent automatically. In master node and slave node configuration, the parity is checked automatically.

1: Whatever the node configuration is, the Identifier parity is not computed/sent and it is not checked.

- **CHKDIS: Checksum Disable**

0: In master node configuration, the checksum is computed and sent automatically. In slave node configuration, the checksum is checked automatically.

1: Whatever the node configuration is, the checksum is not computed/sent and it is not checked.

- **CHKTYP: Checksum Type**

0: LIN 2.0 “enhanced” checksum

1: LIN 1.3 “classic” checksum

- **DLM: Data Length Mode**

0: The response data length is defined by the DLC field of this register.

1: The response data length is defined by the bits 5 and 6 of the identifier (IDCHR in FLEX_US_LINIR).

- **FSDIS: Frame Slot Mode Disable**

0: The Frame Slot mode is enabled.

1: The Frame Slot mode is disabled.

- **WKUPTYP: Wakeup Signal Type**

0: Setting the LINWKUP bit in the control register sends a LIN 2.0 wakeup signal.

1: Setting the LINWKUP bit in the control register sends a LIN 1.3 wakeup signal.

- **DLC: Data Length Control**

0–255: Defines the response data length if DLM = 0,in that case the response data length is equal to DLC+1 bytes.

- **PDCM: DMAC Mode**

0: The LIN mode register FLEX_US_LINMR is not written by the DMAC.

1: The LIN mode register FLEX_US_LINMR (excepting that flag) is written by the DMAC.

- **SYNCDIS: Synchronization Disable**

0: The synchronization procedure is performed in LIN slave node configuration.

1: The synchronization procedure is not performed in LIN slave node configuration.

### 44.10.32 USART LIN Identifier Register

**Name:** FLEX_US_LINIR

**Address:** 0xF8034258 (0), 0xF8038258 (1), 0xFC010258 (2), 0xFC014258 (3), 0xFC018258 (4)

**Access:** Read/Write or Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IDCHR | | | | | | | |

This register is relevant only if USART_MODE = 0xA or 0xB in Section 44.10.6 “USART Mode Register”.

- **IDCHR: Identifier Character**

If USART_MODE = 0xA (master node configuration):

IDCHR is Read/Write and its value is the identifier character to be transmitted.

If USART_MODE = 0xB (slave node configuration):

IDCHR is Read-only and its value is the last identifier character that has been received.

### 44.10.33 USART LIN Baud Rate Register

**Name:** FLEX_US_LINBRR

**Address:** 0xF803425C (0), 0xF803825C (1), 0xFC01025C (2), 0xFC01425C (3), 0xFC01825C (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | LINFP | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| LINCD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| LINCD | | | | | | | |

This register is relevant only if USART_MODE = 0xA or 0xB in Section 44.10.6 “USART Mode Register”.

Returns the baud rate value after the synchronization process completion.

- **LINCD: Clock Divider after Synchronization**
- **LINFP: Fractional Part after Synchronization**

### 44.10.34 USART Comparison Register

**Name:** FLEX_US_CMPR

**Address:** 0xF8034290 (0), 0xF8038290 (1), 0xFC010290 (2), 0xFC014290 (3), 0xFC018290 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | VAL2 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| VAL2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | CMPPAR | – | CMPMODE | – | – | – | VAL1 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VAL1 | | | | | | | |

This register can only be written if the WPEN bit is cleared in the USART Write Protection Mode Register.

- **VAL1: First Comparison Value for Received Character**

0–511: The received character must be higher or equal to the value of VAL1 and lower or equal to VAL2 to set CMP flag in FLEX_US_CSR.

- **CMPMODE: Comparison Mode**

| Value | Name | Description |
|---|---|---|
| 0 | FLAG_ONLY | Any character is received and comparison function drives CMP flag. |
| 1 | START_CONDITION | Comparison condition must be met to start reception. |

- **CMPPAR: Compare Parity**

0: The parity is not checked and a bad parity cannot prevent from waking up the system.

1: The parity is checked and a matching condition on data can be cancelled by an error on parity bit, so no wakeup is performed.

- **VAL2: Second Comparison Value for Received Character**

0–511: The received character must be lower or equal to the value of VAL2 and higher or equal to VAL1 to set CMP flag in FLEX_US_CSR.

### 44.10.35 USART FIFO Mode Register

**Name:** FLEX_US_FMR

**Address:** 0xF80342A0 (0), 0xF80382A0 (1), 0xFC0102A0 (2), 0xFC0142A0 (3), 0xFC0182A0 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | RXFTHRES2 | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFTHRES | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | TXFTHRES | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FRTSC | – | RXRDYM | | – | – | TXRDYM | |

- **TXRDYM: Transmitter Ready Mode**

If FIFOs are enabled, the TXRDY flag (in FLEX_US_CSR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0 | ONE_DATA | TXRDY will be at level ‘1’ when at least one data can be written in the Transmit FIFO |
| 1 | TWO_DATA | TXRDY will be at level ‘1’ when at least two data can be written in the Transmit FIFO |
| 2 | FOUR_DATA | TXRDY will be at level ‘1’ when at least four data can be written in the Transmit FIFO |

- **RXRDYM: Receiver Ready Mode**

If FIFOs are enabled, the RXRDY flag (in FLEX_US_CSR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0 | ONE_DATA | RXRDY will be at level ‘1’ when at least one unread data is in the Receive FIFO |
| 1 | TWO_DATA | RXRDY will be at level ‘1’ when at least two unread data are in the Receive FIFO |
| 2 | FOUR_DATA | RXRDY will be at level ‘1’ when at least four unread data are in the Receive FIFO |

- **FRTSC: FIFO RTS pin Control enable (Hardware Handshaking mode only)**

0: RTS pin is not controlled by Receive FIFO thresholds.

1: RTS pin is controlled by Receive FIFO thresholds.

See Section 44.7.3.15 “Hardware Handshaking” for details.

- **TXFTHRES: Transmit FIFO Threshold**

0–32: Defines the Transmit FIFO threshold value (number of data). TXFTHF flag in FLEX_US_FESR will be set when Transmit FIFO goes from “above” threshold state to “equal or below” threshold state.

- **RXFTHRES: Receive FIFO Threshold**

0–32: Defines the Receive FIFO threshold value (number of data). RXFTHF flag in FLEX_US_FESR will be set when Receive FIFO goes from “below” threshold state to “equal to or above” threshold state.

- **RXFTHRES2: Receive FIFO Threshold 2**

0–32: Defines the Receive FIFO threshold 2 value (number of data). RXFTHF2 flag in FLEX_US_FESR will be set when Receive FIFO goes from “above” threshold state to “equal or below” threshold state.

### 44.10.36 USART FIFO Level Register

**Name:** FLEX_US_FLR

**Address:** 0xF80342A4 (0), 0xF80382A4 (1), 0xFC0102A4 (2), 0xFC0142A4 (3), 0xFC0182A4 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFL | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TXFL | | | | | |

- **TXFL: Transmit FIFO Level**

0: There is no data in the Transmit FIFO

1–32: Indicates the number of DATA in the Transmit FIFO

- **RXFL: Receive FIFO Level**

0: There is no unread data in the Receive FIFO

1–32: Indicates the number of unread DATA in the Receive FIFO

### 44.10.37 USART FIFO Interrupt Enable Register

**Name:** FLEX_US_FIER

**Address:** 0xF80342A8 (0), 0xF80382A8 (1), 0xFC0102A8 (2), 0xFC0142A8 (3), 0xFC0182A8 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | RXFTHF2 | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Enable**
- **TXFFF: TXFFF Interrupt Enable**
- **TXFTHF: TXFTHF Interrupt Enable**
- **RXFEF: RXFEF Interrupt Enable**
- **RXFFF: RXFFF Interrupt Enable**
- **RXFTHF: RXFTHF Interrupt Enable**
- **TXFPTEF: TXFPTEF Interrupt Enable**
- **RXFPTEF: RXFPTEF Interrupt Enable**
- **RXFTHF2: RXFTHF2 Interrupt Enable**

### 44.10.38 USART FIFO Interrupt Disable Register

**Name:** FLEX_US_FIDR

**Address:** 0xF80342AC (0), 0xF80382AC (1), 0xFC0102AC (2), 0xFC0142AC (3), 0xFC0182AC (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | RXFTHF2 | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Disable**
- **TXFFF: TXFFF Interrupt Disable**
- **TXFTHF: TXFTHF Interrupt Disable**
- **RXFEF: RXFEF Interrupt Disable**
- **RXFFF: RXFFF Interrupt Disable**
- **RXFTHF: RXFTHF Interrupt Disable**
- **TXFPTEF: TXFPTEF Interrupt Disable**
- **RXFPTEF: RXFPTEF Interrupt Disable**
- **RXFTHF2: RXFTHF2 Interrupt Disable**

### 44.10.39 USART FIFO Interrupt Mask Register

**Name:** FLEX_US_FIMR

**Address:** 0xF80342B0 (0), 0xF80382B0 (1), 0xFC0102B0 (2), 0xFC0142B0 (3), 0xFC0182B0 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | RXFTHF2 | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Mask**
- **TXFFF: TXFFF Interrupt Mask**
- **TXFTHF: TXFTHF Interrupt Mask**
- **RXFEF: RXFEF Interrupt Mask**
- **RXFFF: RXFFF Interrupt Mask**
- **RXFTHF: RXFTHF Interrupt Mask**
- **TXFPTEF: TXFPTEF Interrupt Mask**
- **RXFPTEF: RXFPTEF Interrupt Mask**
- **RXFTHF2: RXFTHF2 Interrupt Mask**

### 44.10.40 USART FIFO Event Status Register

**Name:** FLEX_US_FESR

**Address:** 0xF80342B4 (0), 0xF80382B4 (1), 0xFC0102B4 (2), 0xFC0142B4 (3), 0xFC0182B4 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | RXFTHF2 | TXFLOCK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: Transmit FIFO Empty Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Transmit FIFO is not empty.

1: Transmit FIFO has been emptied since the last RSTSTA command was issued.

- **TXFFF: Transmit FIFO Full Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Transmit FIFO is not full.

1: Transmit FIFO has been filled since the last RSTSTA command was issued.

- **TXFTHF: Transmit FIFO Threshold Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Number of DATA in Transmit FIFO is above TXFTHRES threshold.

1: Number of DATA in Transmit FIFO has reached TXFTHRES threshold since the last RSTSTA command was issued.

- **RXFEF: Receive FIFO Empty Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Receive FIFO is not empty.

1: Receive FIFO has been emptied since the last RSTSTA command was issued.

- **RXFFF: Receive FIFO Full Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Receive FIFO is not empty.

1: Receive FIFO has been filled since the last RSTSTA command was issued.

- **RXFTHF: Receive FIFO Threshold Flag (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Number of unread DATA in Receive FIFO is below RXFTHRES threshold.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES threshold since the last RSTSTA command was issued.

- **TXFPTEF: Transmit FIFO Pointer Error Flag**

0: No Transmit FIFO pointer occurred

1: Transmit FIFO pointer error occurred. Transceiver must be reset

See Section 44.7.11.8 “FIFO Pointer Error” for details.

- **RXFPTEF: Receive FIFO Pointer Error Flag**

0: No Receive FIFO pointer occurred

1: Receive FIFO pointer error occurred. Receiver must be reset

See Section 44.7.11.8 “FIFO Pointer Error” for details.

- **TXFLOCK: Transmit FIFO Lock**

0: The Transmit FIFO is not locked.

1: The Transmit FIFO is locked.

- **RXFTHF2: Receive FIFO Threshold Flag 2 (cleared by writing RSTSTA bit in FLEX_US_CR)**

0: Number of unread DATA in Receive FIFO is above RXFTHRES threshold.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES2 threshold since the last RSTSTA command was issued.

### 44.10.41 USART Write Protection Mode Register

**Name:** FLEX_US_WPMR

**Address:** 0xF80342E4 (0), 0xF80382E4 (1), 0xFC0102E4 (2), 0xFC0142E4 (3), 0xFC0182E4 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection on configuration registers if WPKEY corresponds to 0x555341 (“USA” in ASCII).

1: Enables the write protection on configuration registers if WPKEY corresponds to 0x555341 (“USA” in ASCII).

See Section 44.7.12 “USART Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x555341 | PASSWD | Writing any other value in this field aborts the write operation of bit WPEN. Always reads as 0. |

### 44.10.42 USART Write Protection Status Register

**Name:** FLEX_US_WPSR

**Address:** 0xF80342E8 (0), 0xF80382E8 (1), 0xFC0102E8 (2), 0xFC0142E8 (3), 0xFC0182E8 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of FLEX_US_WPSR.

1: A write protection violation has occurred since the last read of FLEX_US_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

### 44.10.43 SPI Control Register

**Name:** FLEX_SPI_CR

**Address:** 0xF8034400 (0), 0xF8038400 (1), 0xFC010400 (2), 0xFC014400 (3), 0xFC018400 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FIFODIS | FIFOEN | – | – | – | – | – | LASTXFER |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | RXFCLR | TXFCLR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | REQCLR | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | – | – | – | – | – | SPIDIS | SPIEN |

- **SPIEN: SPI Enable**

0: No effect.

1: Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0: No effect.

1: Disables the SPI.

If a transfer is in progress when SPIDIS is set, the SPI completes the transmission of the shifter register and does not start any new transfer, even if the FLEX_US_THR is loaded.

All pins are set in Input mode after completion of the transmission in progress, if any.

- **SWRST: SPI Software Reset**

0: No effect.

1: Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in Slave mode after software reset.

- **REQCLR: Request to Clear the Comparison Trigger**

SleepWalking enabled:

0: No effect.

1: Clears the potential clock request currently issued by SPI, thus the potential system wakeup is cancelled.

SleepWalking disabled:

0: No effect.

1: Restarts the comparison trigger to enable FLEX_SPI_RDR loading.

- **TXFCLR: Transmit FIFO Clear**

0: No effect.

1: Clears the Transmit FIFO, Transmit FIFO will become empty.

- **RXFCLR: Receive FIFO Clear**

0: No effect.

1: Clears the Receive FIFO, Receive FIFO will become empty.

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS will be de-asserted after the character written in TD has been transferred. When CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

Refer to Section 44.8.3.5 “Peripheral Selection” for more details.

- **FIFOEN: FIFO Enable**

0: No effect.

1: Enables the Transmit and Receive FIFOs

- **FIFODIS: FIFO Disable**

0: No effect.

1: Disables the Transmit and Receive FIFOs

### 44.10.44 SPI Mode Register

**Name:** FLEX_SPI_MR

**Address:** 0xF8034404 (0), 0xF8038404 (1), 0xFC010404 (2), 0xFC014404 (3), 0xFC018404 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCS | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | PCS | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | CMPMODE | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LLB | – | WDRBT | MODFDIS | BRSRCCLK | PCSDEC | PS | MSTR |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

- **MSTR: Master/Slave Mode**

0: SPI is in Slave mode.

1: SPI is in Master mode.

- **PS: Peripheral Select**

0: Fixed Peripheral Select

1: Variable Peripheral Select

- **PCSDEC: Chip Select Decode**

0: The chip selects are directly connected to a peripheral device.

1: The two NPCS chip select lines are connected to a 2- to 4-bit decoder.

When PCSDEC equals one, up to 3 Chip Select signals can be generated with the two NPCS lines using an external 2- to 4-bit decoder. The Chip Select registers define the characteristics of the 3 chip selects, with the following rules:

FLEX_SPI_CSR0 defines peripheral chip select signals 0 to 1.

FLEX_SPI_CSR1 defines peripheral chip select signal 2.

- **BRSRCCLK: Bit Rate Source Clock**

| Value | Name | Description |
|---|---|---|
| 0 | PERIPH_CLK | The peripheral clock is the source clock for the bit rate generation. |
| 1 | GCLK | GCLK is the source clock for the bit rate generation, thus the bit rate can be independent of the core/peripheral clock. |

Note: If the bit BRSRCCLK = 1, the SCBR field in FLEX_US_CSRx must be programmed with a value greater than 1.

- **MODFDIS: Mode Fault Detection**

0: Mode fault detection is enabled.

1: Mode fault detection is disabled.

- **WDRBT: Wait Data Read Before Transfer**

0: No Effect. In Master mode, a transfer can be initiated regardless of the FLEX_SPI_RDR state.

1: In Master mode, a transfer can start only if FLEX_SPI_RDR is empty, i.e., does not contain any unread data. This mode prevents overrun error in reception.

- **CMPMODE: Comparison Mode**

| Value | Name | Description |
|---|---|---|
| 0 | FLAG_ONLY | Any character is received and comparison function drives CMP flag. |
| 1 | START_CONDITION | Comparison condition must be met to start reception of all incoming characters until REQCLR is set. |

- **LLB: Local Loopback Enable**

0: Local loopback path disabled.

1: Local loopback path enabled.

LLB controls the local loopback on the data shift register for testing in Master mode only (MISO is internally connected on MOSI).

- **PCS: Peripheral Chip Select**

This field is only used if fixed peripheral select is active (PS = 0).

If PCSDEC = 0:

PCS = x0 NPCS[1:0] = 10

PCS = 01 NPCS[1:0] = 01

PCS = 11 forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[1:0] output signals = PCS

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay between the inactivation and the activation of NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is ≤ 6, six peripheral clock periods are inserted by default.

Otherwise, the following equations determine the delay:

If FLEX_SPI_MR.BRSRCCLK = 0: DLYBCS = Delay Between Chip Selects × $f_{peripheral\ clock}$

If FLEX_SPI_MR.BRSRCCLK = 1: DLYBCS = Delay Between Chip Selects × $f_{GCLK}$

### 44.10.45 SPI Receive Data Register

**Name:** FLEX_SPI_RDR

**Address:** 0xF8034408 (0), 0xF8038408 (1), 0xFC010408 (2), 0xFC014408 (3), 0xFC018408 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.8.7.5 “Single Data Mode” for details.

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

- **PCS: Peripheral Chip Select**

In Master mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits are read as zero.

Note: When using Variable Peripheral Select mode (PS = 1 in FLEX_SPI_MR), it is mandatory to set the FLEX_SPI_MR.WDRBT bit to 1 if the PCS field must be processed in FLEX_SPI_RDR.

### 44.10.46 SPI Receive Data Register (FIFO Multiple Data, 8-bit)

**Name:** FLEX_SPI_RDR (FIFO_MULTI_DATA_8)

**Address:** 0xF8034408 (0), 0xF8038408 (1), 0xFC010408 (2), 0xFC014408 (3), 0xFC018408 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RD3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RD2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.8.7.6 “Multiple Data Mode” for details.

- **RDx: Receive Data**

First unread data in the Receive FIFO. Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

### 44.10.47 SPI Receive Data Register (FIFO Multiple Data, 16-bit)

**Name:** FLEX_SPI_RDR (FIFO_MULTI_DATA_16)

**Address:** 0xF8034408 (0), 0xF8038408 (1), 0xFC010408 (2), 0xFC014408 (3), 0xFC018408 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RD1 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RD1 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD0 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.8.7.6 “Multiple Data Mode” for details.

- **RDx: Receive Data**

First unread data in the Receive FIFO. Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

### 44.10.48 SPI Transmit Data Register

**Name:** FLEX_SPI_TDR

**Address:** 0xF803440C (0), 0xF803840C (1), 0xFC01040C (2), 0xFC01440C (3), 0xFC01840C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | LASTXFER |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PCS | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TD | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TD | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.8.7.5 “Single Data Mode” for details.

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if variable peripheral select is active (PS = 1).

If PCSDEC = 0:

PCS = x0 NPCS[1:0] = 10

PCS = 01 NPCS[1:0] = 01

PCS = 11 forbidden (no peripheral is selected)

(x = don’t care)

If PCSDEC = 1:

NPCS[1:0] output signals = PCS

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS is de-asserted after the transfer of the character written in TD. When CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

This field is only used if variable peripheral select is active (PS = 1).

### 44.10.49 SPI Transmit Data Register (FIFO Multiple Data, 8- to 16-bit)

**Name:** FLEX_SPI_TDR (FIFO_MULTI_DATA)

**Address:** 0xF803440C (0), 0xF803840C (1), 0xFC01040C (2), 0xFC01440C (3), 0xFC01840C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TD1 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TD1 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TD0 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section 44.8.7.6 “Multiple Data Mode” for details.

- **TDx: Transmit Data**

Next data to write in the Transmit FIFO. Information to be transmitted must be written to the transmit data register in a right-justified format.

### 44.10.50 SPI Status Register

**Name:** FLEX_SPI_SR

**Address:** 0xF8034410 (0), 0xF8038410 (1), 0xFC010410 (2), 0xFC014410 (3), 0xFC018410 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | SPIENS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

- **RDRF: Receive Data Register Full (cleared by reading FLEX_SPI_RDR)**

When FIFOs are disabled:

0: No data has been received since the last read of FLEX_SPI_RDR.

1: Data has been received and the received data has been transferred from the shift register to FLEX_SPI_RDR since the last read of FLEX_SPI_RDR.

When FIFOs are enabled:

0: Receive FIFO is empty; no data to read

1: At least one unread data is in the Receive FIFO

RDRF behavior with FIFOs enabled is illustrated in Section 44.8.7.4 “TDRE, RDRF and TXEMPTY behavior”.

- **TDRE: Transmit Data Register Empty (cleared by writing FLEX_SPI_TDR)**

0: Data has been written to FLEX_SPI_TDR and not yet transferred to the shift register.

1: The last data written to FLEX_SPI_TDR has been transferred to the shift register.

TDRE = 0 when the SPI is disabled or at reset. The SPI enable command sets this bit to 1.

When FIFOs are enabled:

0: Transmit FIFO is full and cannot accept more data

1: Transmit FIFO is not full; one or more data can be written according to TXRDYM field configuration

TDRE behavior with FIFOs enabled is illustrated in Section 44.8.7.4 “TDRE, RDRF and TXEMPTY behavior”.

- **MODF: Mode Fault Error (cleared on read)**

0: No mode fault has been detected since the last read of FLEX_SPI_SR.

1: A mode fault occurred since the last read of FLEX_SPI_SR.

- **OVRES: Overrun Error Status (cleared on read)**

0: No overrun has been detected since the last read of FLEX_SPI_SR.

1: An overrun has occurred since the last read of FLEX_SPI_SR.

An overrun occurs when FLEX_SPI_RDR is loaded at least twice from the shift register since the last read of FLEX_SPI_RDR.

- **NSSR: NSS Rising (cleared on read)**

0: No rising edge detected on NSS pin since the last read of FLEX_SPI_SR.

1: A rising edge occurred on NSS pin since the last read of FLEX_SPI_SR.

- **TXEMPTY: Transmission Registers Empty (cleared by writing FLEX_SPI_TDR)**

0: As soon as data is written in FLEX_SPI_TDR.

1: FLEX_SPI_TDR and internal shift register are empty. If a transfer delay has been defined, TXEMPTY is set after the end of this delay.

- **UNDES: Underrun Error Status (Slave mode Only) (cleared on read)**

0: No underrun has been detected since the last read of FLEX_SPI_SR.

1: A transfer starts whereas no data has been loaded in FLEX_SPI_TDR, cleared when FLEX_SPI_SR is read.

- **SPIENS: SPI Enable Status**

0: SPI is disabled.

1: SPI is enabled.

- **CMP: Comparison Status (cleared on read)**

0: No received character matched the comparison criteria programmed in VAL1 and VAL2 fields in FLEX_SPI_CMPR since the last read of FLEX_SPI_SR.

1: A received character matched the comparison criteria since the last read of FLEX_SPI_SR.

- **TXFEF: Transmit FIFO Empty Flag (cleared on read)**

0: Transmit FIFO is not empty.

1: Transmit FIFO has been emptied since the last read of FLEX_SPI_SR.

- **TXFFF: Transmit FIFO Full Flag (cleared on read)**

0: Transmit FIFO is not full or TXFF flag has been cleared.

1: Transmit FIFO has been filled since the last read of FLEX_SPI_SR.

- **TXFTHF: Transmit FIFO Threshold Flag (cleared on read)**

0: Number of DATA in Transmit FIFO is above TXFTHRES threshold.

1: Number of DATA in Transmit FIFO has reached TXFTHRES threshold since the last read of FLEX_SPI_SR.

- **RXFEF: Receive FIFO Empty Flag**

0: Receive FIFO is not empty or RXFE flag has been cleared.

1: Receive FIFO has become empty (coming from “not empty” state to “empty” state).

- **RXFFF: Receive FIFO Full Flag**

0: Receive FIFO is not empty or RXFE flag has been cleared.

1: Receive FIFO has become full (coming from “not full” state to “full” state).

- **RXFTHF: Receive FIFO Threshold Flag**

0: Number of unread DATA in Receive FIFO is below RXFTHRES threshold or RXFTH flag has been cleared.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES threshold (coming from “below threshold” state to “equal to or above threshold” state).

- **TXFPTEF: Transmit FIFO Pointer Error Flag**

0: No Transmit FIFO pointer occurred

1: Transmit FIFO pointer error occurred. Transceiver must be reset

See Section 44.8.7.7 “FIFO Pointer Error” for details.

- **RXFPTEF: Receive FIFO Pointer Error Flag**

0: No Receive FIFO pointer occurred

1: Receive FIFO pointer error occurred. Receiver must be reset

See Section 44.8.7.7 “FIFO Pointer Error” for details.

### 44.10.51 SPI Interrupt Enable Register

**Name:** FLEX_SPI_IER

**Address:** 0xF8034414 (0), 0xF8038414 (1), 0xFC010414 (2), 0xFC014414 (3), 0xFC018414 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**
- **CMP: Comparison Interrupt Enable**
- **TXFEF: TXFEF Interrupt Enable**
- **TXFFF: TXFFF Interrupt Enable**
- **TXFTHF: TXFTHF Interrupt Enable**
- **RXFEF: RXFEF Interrupt Enable**
- **RXFFF: RXFFF Interrupt Enable**
- **RXFTHF: RXFTHF Interrupt Enable**
- **TXFPTEF: TXFPTEF Interrupt Enable**
- **RXFPTEF: RXFPTEF Interrupt Enable**

### 44.10.52 SPI Interrupt Disable Register

**Name:** FLEX_SPI_IDR

**Address:** 0xF8034418 (0), 0xF8038418 (1), 0xFC010418 (2), 0xFC014418 (3), 0xFC018418 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**
- **CMP: Comparison Interrupt Disable**
- **TXFEF: TXFEF Interrupt Disable**
- **TXFFF: TXFFF Interrupt Disable**
- **TXFTHF: TXFTHF Interrupt Disable**
- **RXFEF: RXFEF Interrupt Disable**
- **RXFFF: RXFFF Interrupt Disable**
- **RXFTHF: RXFTHF Interrupt Disable**
- **TXFPTEF: TXFPTEF Interrupt Disable**
- **RXFPTEF: RXFPTEF Interrupt Disable**

### 44.10.53 SPI Interrupt Mask Register

**Name:** FLEX_SPI_IMR

**Address:** 0xF803441C (0), 0xF803841C (1), 0xFC01041C (2), 0xFC01441C (3), 0xFC01841C (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**
- **CMP: Comparison Interrupt Mask**
- **TXFEF: TXFEF Interrupt Mask**
- **TXFFF: TXFFF Interrupt Mask**
- **TXFTHF: TXFTHF Interrupt Mask**
- **RXFEF: RXFEF Interrupt Mask**
- **RXFFF: RXFFF Interrupt Mask**
- **RXFTHF: RXFTHF Interrupt Mask**
- **TXFPTEF: TXFPTEF Interrupt Mask**
- **RXFPTEF: RXFPTEF Interrupt Mask**

### 44.10.54 SPI Chip Select Register

**Name:** FLEX_SPI_CSRx[x = 0..1]

**Address:** 0xF8034430 (0), 0xF8038430 (1), 0xFC010430 (2), 0xFC014430 (3), 0xFC018430 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCT | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DLYBS | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SCBR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BITS | | | | CSAAT | CSNAAT | NCPHA | CPOL |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

Note: FLEX_SPI_CSRx must be written even if the user wants to use the default reset values. The BITS field is not updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0: The inactive state value of SPCK is logic level zero.

1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0: Data are changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data are captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSNAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0: The Peripheral Chip Select does not rise between two transfers if the FLEX_US_TDR is reloaded before the end of the first transfer and if the two transfers occur on the same Chip Select.

1: The Peripheral Chip Select rises systematically after each transfer performed on the same slave. It remains inactive after the end of transfer for a minimal duration of:

If FLEX_SPI_MR.BRSRCCLK = 0: $\frac{DLYBCS}{f_{\text{peripheral clock}}}$ (if DLYBCS ≠ 0)

If FLEX_SPI_MR.BRSRCCLK = 1: $\frac{DLYBCS}{f_{GCLK}}$

If DLYBCS < 6, a minimum of six periods is introduced.

- **CSAAT: Chip Select Active After Transfer**

0: The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1: The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

(See the note below the FLEX_SPI_CSRx bitmap.)

The BITS field determines the number of data bits transferred. Reserved values should not be used.

| Value | Name | Description |
|---|---|---|
| 0 | 8_BIT | 8 bits for transfer |
| 1 | 9_BIT | 9 bits for transfer |
| 2 | 10_BIT | 10 bits for transfer |
| 3 | 11_BIT | 11 bits for transfer |
| 4 | 12_BIT | 12 bits for transfer |
| 5 | 13_BIT | 13 bits for transfer |
| 6 | 14_BIT | 14 bits for transfer |
| 7 | 15_BIT | 15 bits for transfer |
| 8 | 16_BIT | 16 bits for transfer |
| 9–15 | – | Reserved |

- **SCBR: Serial Clock Bit Rate**

In Master mode, the SPI Interface uses a modulus counter to derive the SPCK bit rate from the clock defined by the bit BRSRCCLK. The bit rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK bit rate:

If FLEX_SPI_MR.BRSRCCLK = 0: SCBR = $f_{peripheral\ clock}$ / SPCK Bit Rate

If FLEX_SPI_MR.BRSRCCLK = 1: SCBR = $f_{GCLK}$ / SPCK Bit Rate

Programming the SCBR field to 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

If BRSRCCLK = 1 in FLEX_SPI_MR, SCBR must be programmed with a value greater than 1.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields in FLEX_SPI_CSRx is set to 1, the other SCBR fields in FLEX_SPI_CSRx must be set to 1 as well, if they are used to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS falling edge (activation) to the first valid SPCK transition.

When DLYBS = 0, the delay is half the SPCK clock period.

Otherwise, the following equations determine the delay:

If FLEX_SPI_MR.BRSRCCLK = 0: DLYBS = Delay Before SPCK $\times$ $f_{peripheral\ clock}$

If FLEX_SPI_MR.BRSRCCLK = 1: DLYBS = Delay Before SPCK $\times$ $f_{GCLK}$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT = 0, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equations determine the delay:

If FLEX_SPI_MR.BRSRCCLK = 0: DLYBCT = Delay Between Consecutive Transfers $\times$ $f_{peripheral\ clock}$ / 32

If FLEX_SPI_MR.BRSRCCLK = 1: DLYBCT = Delay Between Consecutive Transfers $\times$ $f_{GCLK}$ / 32

### 44.10.55 SPI FIFO Mode Register

**Name:** FLEX_SPI_FMR

**Address:** 0xF8034440 (0), 0xF8038440 (1), 0xFC010440 (2), 0xFC014440 (3), 0xFC018440 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | RXFTHRES | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | TXFTHRES | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | RXRDYM | | – | – | TXRDYM | |

- **TXRDYM: Transmitter Data Register Empty Mode**

If FIFOs are enabled, the TDRE flag (in FLEX_SPI_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | TDRE will be at level ‘1’ when at least one data can be written in the Transmit FIFO. |
| 0x1 | TWO_DATA | TDRE will be at level ‘1’ when at least two data can be written in the Transmit FIFO. |
| 0x2 | FOUR_DATA | TDRE will be at level ‘1’ when at least four data can be written in the Transmit FIFO. |

- **RXRDYM: Receiver Data Register Full Mode**

If FIFOs are enabled, the RDRF flag (in FLEX_SPI_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | RDRF will be at level ‘1’ when at least one unread data is in the Receive FIFO. |
| 0x1 | TWO_DATA | RDRF will be at level ‘1’ when at least two unread data are in the Receive FIFO. |
| 0x2 | FOUR_DATA | RDRF will be at level ‘1’ when at least four unread data are in the Receive FIFO. |

- **TXFTHRES: Transmit FIFO Threshold**

0–32: Defines the Transmit FIFO threshold value (number of data). TXFTH flag in FLEX_SPI_SR will be set when Transmit FIFO goes from “above” threshold state to “equal or below” threshold state.

- **RXFTHRES: Receive FIFO Threshold**

0–32: Defines the Receive FIFO threshold value (number of data). RXFTH flag in FLEX_SPI_SR will be set when Receive FIFO goes from “below” threshold state to “equal to or above” threshold state.

### 44.10.56 SPI FIFO Level Register

**Name:** FLEX_SPI_FLR

**Address:** 0xF8034444 (0), 0xF8038444 (1), 0xFC010444 (2), 0xFC014444 (3), 0xFC018444 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFL | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TXFL | | | | | |

- **TXFL: Transmit FIFO Level**

0: There is no data in the Transmit FIFO

1–32: Indicates the number of DATA in the Transmit FIFO

- **RXFL: Receive FIFO Level**

0: There is no unread data in the Receive FIFO

1–32: Indicates the number of unread DATA in the Receive FIFO

### 44.10.57 SPI Comparison Register

**Name:** FLEX_SPI_CMPR

**Address:** 0xF8034448 (0), 0xF8038448 (1), 0xFC010448 (2), 0xFC014448 (3), 0xFC018448 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| VAL2 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| VAL2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| VAL1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VAL1 | | | | | | | |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

- **VAL1: First Comparison Value for Received Character**

0–65535: The received character must be higher or equal to the value of VAL1 and lower or equal to VAL2 to set CMP flag in FLEX_SPI_SR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the SPI requests a system wakeup if the condition is met.

- **VAL2: Second Comparison Value for Received Character**

0–65535: The received character must be lower or equal to the value of VAL2 and higher or equal to VAL1 to set CMP flag in FLEX_SPI_CSR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the SPI requests a system wakeup if condition is met.

### 44.10.58 SPI Write Protection Mode Register

**Name:** FLEX_SPI_WPMR

**Address:** 0xF80344E4 (0), 0xF80384E4 (1), 0xFC0104E4 (2), 0xFC0144E4 (3), 0xFC0184E4 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

1: Enables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

See Section 44.8.8 “SPI Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x535049 | PASSWD | Writing any other value in this field aborts the write operation of bit WPEN.<br>Always reads as 0 |

### 44.10.59 SPI Write Protection Status Register

**Name:** FLEX_SPI_WPSR

**Address:** 0xF80344E8 (0), 0xF80384E8 (1), 0xFC0104E8 (2), 0xFC0144E8 (3), 0xFC0184E8 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protect violation has occurred since the last read of FLEX_SPI_WPSR.

1: A write protect violation has occurred since the last read of FLEX_SPI_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

### 44.10.60 TWI Control Register

Name: FLEX_TWI_CR

**Address:** 0xF8034600 (0), 0xF8038600 (1), 0xFC010600 (2), 0xFC014600 (3), 0xFC018600 (4)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | FIFODIS | FIFOEN | – | LOCKCLR | – | THRCLR |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | ACMDIS | ACMEN |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CLEAR | PECRQ | PECDIS | PECEN | SMBDIS | SMBEN | HSDIS | HSEN |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | QUICK | SVDIS | SVEN | MSDIS | MSEN | STOP | START |

- **START: Send a START Condition**

0: No effect.

1: A frame beginning with a START bit is transmitted according to the features defined in the TWI Master Mode Register (FLEX_TWI_MMR).

This action is necessary when the TWI peripheral needs to read data from a slave. When configured in Master mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (FLEX_TWI_THR).

- **STOP: Send a STOP Condition**

0: No effect.

1: STOP condition is sent just after completing the current byte transmission in Master Read mode.

  – In single data byte master read, both START and STOP must be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In Master Read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0: No effect.

1: Enables the Master mode (MSDIS must be written to 0).

Note: Switching from Slave to Master mode is only permitted when TXCOMP = 1.

- **MSDIS: TWI Master Mode Disabled**

0: No effect.

1: The Master mode is disabled, all pending data is transmitted. The shifter and holding characters (if it contains data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SVEN: TWI Slave Mode Enabled**

0: No effect.

1: Enables the Slave mode (SVDIS must be written to 0).

Note: Switching from Master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWI Slave Mode Disabled**

0: No effect.

1: The Slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBus Quick Command**

0: No effect.

1: If Master mode is enabled, a SMBus Quick Command is sent.

- **SWRST: Software Reset**

0: No effect.

1: Equivalent to a system reset.

- **HSEN: TWI High-Speed Mode Enabled**

0: No effect.

1: High-speed mode enabled.

- **HSDIS: TWI High-Speed Mode Disabled**

0: No effect.

1: High-speed mode disabled.

- **SMBEN: SMBus Mode Enabled**

0: No effect.

1: If SMBDIS = 0, SMBus mode enabled.

- **SMBDIS: SMBus Mode Disabled**

0: No effect.

1: SMBus mode disabled.

- **PECEN: Packet Error Checking Enable**

0: No effect.

1: SMBus PEC (CRC) generation and check enabled.

- **PECDIS: Packet Error Checking Disable**

0: No effect.

1: SMBus PEC (CRC) generation and check disabled.

- **PECRQ: PEC Request**

0: No effect.

1: A PEC check or transmission is requested.

- **CLEAR: Bus CLEAR Command**

0: No effect.

1: If Master mode is enabled, send a bus clear command.

- **ACMEN: Alternative Command Mode Enable**

0: No effect.

1: Alternative Command mode enabled.

- **ACMDIS: Alternative Command Mode Disable**

0: No effect.

1: Alternative Command mode disabled.

- **THRCLR: Transmit Holding Register Clear**

0: No effect.

1: Clear the Transmit Holding Register and set TXRDY, TXCOMP flags.

- **LOCKCLR: Lock Clear**

0: No effect.

1: Clear the TWI FSM lock.

- **FIFOEN: FIFO Enable**

0: No effect.

1: Enable the Transmit and Receive FIFOs

- **FIFODIS: FIFO Disable**

0: No effect.

1: Disable the Transmit and Receive FIFOs

### 44.10.61 TWI Master Mode Register

**Name:** FLEX_TWI_MMR

**Address:** 0xF8034604 (0), 0xF8038604 (1), 0xFC010604 (2), 0xFC014604 (3), 0xFC018604 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | DADR | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | MREAD | – | – | IADRSZ | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |

- **IADRSZ: Internal Device Address Size**

| Value | Name | Description |
|---|---|---|
| 0 | NONE | No internal device address |
| 1 | 1_BYTE | One-byte internal device address |
| 2 | 2_BYTE | Two-byte internal device address |
| 3 | 3_BYTE | Three-byte internal device address |

- **MREAD: Master Read Direction**

0: Master write direction.

1: Master read direction.

- **DADR: Device Address**

The device address is used to access slave devices in Read or Write mode. Those bits are only used in Master mode.

### 44.10.62 TWI Slave Mode Register

Name: FLEX_TWI_SMR

**Address:** 0xF8034608 (0), 0xF8038608 (1), 0xFC010608 (2), 0xFC014608 (3), 0xFC018608 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DATAMEN | SADR3EN | SADR2EN | SADR1EN | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | SADR | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | MASK | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | SCLWSDIS | – | – | SMHH | SMDA | – | NACKEN |

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

- **NACKEN: Slave Receiver Data Phase NACK Enable**

0: Normal value to be returned in the ACK cycle of the data phase in Slave Receiver mode.

1: NACK value to be returned in the ACK cycle of the data phase in Slave Receiver mode.

- **SMDA: SMBus Default Address**

0: Acknowledge of the SMBus Default Address disabled.

1: Acknowledge of the SMBus Default Address enabled.

- **SMHH: SMBus Host Header**

0: Acknowledge of the SMBus Host Header disabled.

1: Acknowledge of the SMBus Host Header enabled.

- **SCLWSDIS: Clock Wait State Disable**

0: No effect.

1: Clock stretching disabled in Slave mode, OVRE and UNRE will indicate overrun and underrun.

- **MASK: Slave Address Mask**

A mask can be applied on the slave device address in Slave mode in order to allow multiple address answer. For each bit of the MASK field set to one the corresponding SADR bit will be masked.

If MASK field is set to 0 no mask is applied to SADR field.

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in Read or Write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

- **SADR1EN: Slave Address 1 Enable**

0: Slave address 1 matching is disabled.

1: Slave address 1 matching is enabled.

- **SADR2EN: Slave Address 2 Enable**

0: Slave address 2 matching is disabled.

1: Slave address 2 matching is enabled.

- **SADR3EN: Slave Address 3 Enable**

0: Slave address 3 matching is disabled.

1: Slave address 3 matching is enabled.

- **DATAMEN: Data Matching Enable**

0: Data matching on first received data is disabled.

1: Data matching on first received data is enabled.

### 44.10.63 TWI Internal Address Register

Name: FLEX_TWI_IADR

**Address:** 0xF803460C (0), 0xF803860C (1), 0xFC01060C (2), 0xFC01460C (3), 0xFC01860C (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| IADR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| IADR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IADR | | | | | | | |

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

### 44.10.64 TWI Clock Waveform Generator Register

Name: FLEX_TWI_CWGR

**Address:** 0xF8034610 (0), 0xF8038610 (1), 0xFC010610 (2), 0xFC014610 (3), 0xFC018610 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | HOLD | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | – | – | BRSRCCLK | – | CKDIV | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CHDIV | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CLDIV | | | | | | | |

This register can only be written if the WPEN bit is cleared in the TWI Write Protection Mode Register.

FLEX_TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

If BRSRCCLK = 0: CLDIV = $((t_{low} / t_{peripheral\ clock}) - 3) / 2^{CKDIV}$

If BRSRCCLK = 1: CLDIV = $(t_{low} / t_{ext\_ck}) / 2^{CKDIV}$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

If BRSRCCLK = 0: CHDIV = $((t_{high} / t_{peripheral\ clock}) - 3) / 2^{CKDIV}$

If BRSRCCLK = 1: CHDIV = $(t_{high} / t_{ext\_ck}) / 2^{CKDIV}$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

- **BRSRCCLK: Bit Rate Source Clock**

| Value | Name | Description |
|---|---|---|
| 0 | PERIPH_CLK | The peripheral clock is the source clock for the bit rate generation. |
| 1 | GCLK | GCLK is the source clock for the bit rate generation, thus the bit rate can be independent of the core/peripheral clock. |

- **HOLD: TWD Hold Time Versus TWCK Falling**

If High-speed mode is selected TWD is internally modified on the TWCK falling edge to meet the $I^2C$ specified maximum hold time, else if High-speed mode is not configured TWD is kept unchanged after TWCK falling edge for a period of $(HOLD + 3) \times t_{peripheral\ clock}$.

- **CKSRC: Transfer Rate Clock Source**

| Value | Name | Description |
|---|---|---|
| 0 | PERIPH_CLK | The peripheral clock is used to generate the TWI bitrate. |
| 1 | GCLK | GCLK is used to generate the TWI bitrate. |

### 44.10.65 TWI Status Register

Name: FLEX_TWI_SR

**Address:** 0xF8034620 (0), 0xF8038620 (1), 0xFC010620 (2), 0xFC014620 (3), 0xFC018620 (4)

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | SDA | SCL |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| LOCK | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | EOSACC | SCLWS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | SVREAD | TXRDY | RXRDY | TXCOMP |

- **TXCOMP: Transmission Completed (cleared by writing FLEX_TWI_THR)**

TXCOMP used in Master mode:

0: During the length of the current frame.

1: When both holding register and internal shifter are empty and STOP condition has been sent.

*TXCOMP behavior in Master mode* can be seen in Figure 44-87 and in Figure 44-89.

TXCOMP used in Slave mode:

0: As soon as a Start is detected.

1: After a Stop or a Repeated Start + an address different from SADR is detected.

*TXCOMP behavior in Slave mode* can be seen in Figure 44-121, Figure 44-122, Figure 44-123 and Figure 44-124.

- **RXRDY: Receive Holding Register Ready (cleared when reading FLEX_TWI_RHR)**

When FIFOs are disabled:

0: No character has been received since the last FLEX_TWI_RHR read operation.

1: A byte has been received in FLEX_TWI_RHR since the last read.

*RXRDY behavior in Master mode* can be seen in Figure 44-89.

*RXRDY behavior in Slave mode* can be seen in Figure 44-119, Figure 44-122, Figure 44-123 and Figure 44-124.

When FIFOs are enabled:

0: Receive FIFO is empty; no data to read

1: At least one unread data is in the Receive FIFO

RXRDY behavior with FIFO enabled is illustrated in Section "TXRDY and RXRDY Behavior" and Section "TXRDY and RXRDY Behavior".

- **TXRDY: Transmit Holding Register Ready (cleared by writing FLEX_TWI_THR)**

TXRDY used in Master mode:

0: The transmit holding register has not been transferred into the internal shifter. Set to 0 when writing into FLEX_TWI_THR.

1: As soon as a data byte is transferred from FLEX_TWI_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enables TWI).

*TXRDY behavior in Master mode* can be seen in Figure 44-85, Figure 44-86 and Figure 44-87.

TXRDY used in Slave mode:

0: As soon as data is written in FLEX_TWI_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1: Indicates that FLEX_TWI_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission will be stopped. Thus when TRDY = NACK = 1, the user must not fill FLEX_TWI_THR to avoid losing it.

*TXRDY behavior in Slave mode* can be seen in Figure 44-118, Figure 44-121, Figure 44-123 and Figure 44-124.

When FIFOs are enabled:

0: Transmit FIFO is full and cannot accept more data

1: Transmit FIFO is not full; one or more data can be written according to TXRDYM field configuration

TXRDY behavior with FIFOs enabled is illustrated in Section "TXRDY and RXRDY Behavior" and Section "TXRDY and RXRDY Behavior".

- **SVREAD: Slave Read**

This bit is only used in Slave mode. When SVACC is low (no slave access has been detected) SVREAD is irrelevant.

0: Indicates that a write access is performed by a master.

1: Indicates that a read access is performed by a master.

*SVREAD behavior* can be seen in Figure 44-118, Figure 44-119, Figure 44-123 and Figure 44-124.

- **SVACC: Slave Access**

This bit is only used in Slave mode.

0: TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1: Indicates that the address decoding sequence has matched (A master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

*SVACC behavior* can be seen in Figure 44-118, Figure 44-119, Figure 44-123 and Figure 44-124.

- **GACC: General Call Access (cleared on read)**

This bit is only used in Slave mode.

0: No general call has been detected.

1: A general call has been detected. After the detection of general call, if need be, the user may acknowledge this access and decode the following bytes and respond according to the value of the bytes.

*GACC behavior* can be seen in Figure 44-120.

- **OVRE: Overrun Error (cleared on read)**

This bit is only used in Slave mode if clock stretching is disabled.

0: FLEX_TWI_RHR has not been loaded while RXRDY was set.

1: FLEX_TWI_RHR has been loaded while RXRDY was set. Reset by read in FLEX_TWI_SR when TXCOMP is set.

- **UNRE: Underrun Error (cleared on read)**

This bit is only used in Slave mode if clock stretching is disabled.

0: FLEX_TWI_THR has been filled on time.

1: FLEX_TWI_THR has not been filled on time.

- **NACK: Not Acknowledged (cleared on read)**

NACK used in Master mode:

0: Each data byte has been correctly received by the far-end side TWI slave component.

1: A data or address byte has not been acknowledged by the slave component. Set at the same time as TXCOMP.

NACK used in Slave Read mode:

0: Each data byte has been correctly received by the master.

1: In Read mode, a data byte has not been acknowledged by the master. When NACK is set the user must not fill FLEX_TWI_THR even if TXRDY is set, because it means that the master will stop the data transfer or re initiate it.

Note that in Slave Write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (cleared on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State**

This bit is only used in Slave mode.

0: The clock is not stretched.

1: The clock is stretched. FLEX_TWI_THR / FLEX_TWI_RHR buffer is not filled / emptied before the transmission / reception of a new character.

*SCLWS behavior* can be seen in Figure 44-121 and Figure 44-122.

- **EOSACC: End Of Slave Access (cleared on read)**

This bit is only used in Slave mode.

0: A slave access is being performing.

1: The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

*EOSACC behavior* can be seen in Figure 44-123 and Figure 44-124.

- **MCACK: Master Code Acknowledge (cleared on read)**

MACK used in Slave mode:

0: No master code has been received.

1: A master code has been received.

- **TOUT: Timeout Error (cleared on read)**

0: No SMBus timeout occurred.

1: SMBus timeout occurred.

- **PECERR: PEC Error (cleared on read)**

0: No SMBus PEC error occurred.

1: A SMBus PEC error occurred.

- **SMBDAM: SMBus Default Address Match (cleared on read)**

0: No SMBus Default Address received.

1: A SMBus Default Address was received.

- **SMBHHM: SMBus Host Header Address Match (cleared on read)**

0: No SMBus Host Header Address received.

1: A SMBus Host Header Address was received.

- **LOCK: TWI Lock Due to Frame Errors**

0: The TWI is not locked.

1: The TWI is locked due to frame errors (see Section 44.9.3.12 “Handling Errors in Alternative Command” and Section 44.9.3.14 “FIFOs”).

- **SCL: SCL line value**

0: SCL line sampled value is ‘0’.

1: SCL line sampled value is ‘1.’

- **SDA: SDA line value**

0: SDA line sampled value is ‘0’.

1: SDA line sampled value is ‘1’.

### 44.10.66 TWI Interrupt Enable Register

Name: FLEX_TWI_IER

**Address:** 0xF8034624 (0), 0xF8038624 (1), 0xFC010624 (2), 0xFC014624 (3), 0xFC018624 (4)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **TXCOMP: Transmission Completed Interrupt Enable**
- **RXRDY: Receive Holding Register Ready Interrupt Enable**
- **TXRDY: Transmit Holding Register Ready Interrupt Enable**
- **SVACC: Slave Access Interrupt Enable**
- **GACC: General Call Access Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **UNRE: Underrun Error Interrupt Enable**
- **NACK: Not Acknowledge Interrupt Enable**
- **ARBLST: Arbitration Lost Interrupt Enable**
- **SCL_WS: Clock Wait State Interrupt Enable**
- **EOSACC: End Of Slave Access Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **MCACK: Master Code Acknowledge Interrupt Enable**

- **TOUT: Timeout Error Interrupt Enable**
- **PECERR: PEC Error Interrupt Enable**
- **SMBDAM: SMBus Default Address Match Interrupt Enable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Enable**

### 44.10.67 TWI Interrupt Disable Register

Name: FLEX_TWI_IDR

**Address:** 0xF8034628 (0), 0xF8038628 (1), 0xFC010628 (2), 0xFC014628 (3), 0xFC018628 (4)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **TXCOMP: Transmission Completed Interrupt Disable**
- **RXRDY: Receive Holding Register Ready Interrupt Disable**
- **TXRDY: Transmit Holding Register Ready Interrupt Disable**
- **SVACC: Slave Access Interrupt Disable**
- **GACC: General Call Access Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **UNRE: Underrun Error Interrupt Disable**
- **NACK: Not Acknowledge Interrupt Disable**
- **ARBLST: Arbitration Lost Interrupt Disable**
- **SCL_WS: Clock Wait State Interrupt Disable**
- **EOSACC: End Of Slave Access Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **MCACK: Master Code Acknowledge Interrupt Disable**

- **TOUT: Timeout Error Interrupt Disable**
- **PECERR: PEC Error Interrupt Disable**
- **SMBDAM: SMBus Default Address Match Interrupt Disable**
- **SMBHHM: SMBus Host Header Address Match Interrupt Disable**

### 44.10.68 TWI Interrupt Mask Register

Name: FLEX_TWI_IMR

**Address:** 0xF803462C (0), 0xF803862C (1), 0xFC01062C (2), 0xFC01462C (3), 0xFC01862C (4)

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | SMBHHM | SMBDAM | PECERR | TOUT | – | MCACK |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXBUFE | RXBUFF | ENDTX | ENDRX | EOSACC | SCL_WS | ARBLST | NACK |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNRE | OVRE | GACC | SVACC | – | TXRDY | RXRDY | TXCOMP |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **TXCOMP: Transmission Completed Interrupt Mask**
- **RXRDY: Receive Holding Register Ready Interrupt Mask**
- **TXRDY: Transmit Holding Register Ready Interrupt Mask**
- **SVACC: Slave Access Interrupt Mask**
- **GACC: General Call Access Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **UNRE: Underrun Error Interrupt Mask**
- **NACK: Not Acknowledge Interrupt Mask**
- **ARBLST: Arbitration Lost Interrupt Mask**
- **SCL_WS: Clock Wait State Interrupt Mask**
- **EOSACC: End Of Slave Access Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **MCACK: Master Code Acknowledge Interrupt Mask**

- **TOUT: Timeout Error Interrupt Mask**
- **PECERR: PEC Error Interrupt Mask**
- **SMBDAM: SMBus Default Address Match Interrupt Mask**
- **SMBHHM: SMBus Host Header Address Match Interrupt Mask**

### 44.10.69 TWI Receive Holding Register

Name: FLEX_TWI_RHR

**Address:** 0xF8034630 (0), 0xF8038630 (1), 0xFC010630 (2), 0xFC014630 (3), 0xFC018630 (4)

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA | | | | | | | |

- **RXDATA: Master or Slave Receive Holding Data**

### 44.10.70 TWI Receive Holding Register (FIFO Enabled)

Name: FLEX_TWI_RHR (FIFO_ENABLED)

**Address:** 0xF8034630 (0), 0xF8038630 (1), 0xFC010630 (2), 0xFC014630 (3), 0xFC018630 (4)

Access: Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXDATA3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RXDATA2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RXDATA1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXDATA0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section "Master Multiple Data Mode" and Section "Slave Multiple Data Mode" for details.

- **RXDATA0: Master or Slave Receive Holding Data 0**
- **RXDATA1: Master or Slave Receive Holding Data 1**
- **RXDATA2: Master or Slave Receive Holding Data 2**
- **RXDATA3: Master or Slave Receive Holding Data 3**

### 44.10.71 TWI Transmit Holding Register

Name: FLEX_TWI_THR

**Address:** 0xF8034634 (0), 0xF8038634 (1), 0xFC010634 (2), 0xFC014634 (3), 0xFC018634 (4)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA | | | | | | | |

- **TXDATA: Master or Slave Transmit Holding Data**

### 44.10.72 TWI Transmit Holding Register (FIFO Enabled)

Name: FLEX_TWI_THR (FIFO_ENABLED)

**Address:** 0xF8034634 (0), 0xF8038634 (1), 0xFC010634 (2), 0xFC014634 (3), 0xFC018634 (4)

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TXDATA3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TXDATA2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TXDATA1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDATA0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in FLEX_US_CR), refer to Section "Master Multiple Data Mode" and Section "Slave Multiple Data Mode" for details.

- **TXDATA0: Master or Slave Transmit Holding Data 0**
- **TXDATA1: Master or Slave Transmit Holding Data 1**
- **TXDATA2: Master or Slave Transmit Holding Data 2**
- **TXDATA3: Master or Slave Transmit Holding Data 3**

### 44.10.73 TWI SMBus Timing Register

Name: FLEX_TWI_SMBTR

**Address:** 0xF8034638 (0), 0xF8038638 (1), 0xFC010638 (2), 0xFC014638 (3), 0xFC018638 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| THMAX | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| TLOWM | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TLOWS | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | PRESC | | | |

- **PRESC: SMBus Clock Prescaler**

Used to specify how to prescale the TLOWS, TLOWM and THMAX counters in SMBTR. Counters are prescaled according to the following formula: PRESC = Log(fMCK / $f_{Prescaled}$) / Log(2) - 1

- **TLOWS: Slave Clock Stretch Maximum Cycles**

0: TLOW:SEXT timeout check disabled.

1–255: Clock cycles in slave maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:SEXT.

- **TLOWM: Master Clock Stretch Maximum Cycles**

0: TLOW:MEXT timeout check disabled.

1–255: Clock cycles in master maximum clock stretch count. Prescaled by PRESC. Used to time TLOW:MEXT.

- **THMAX: Clock High Maximum Cycles**

Clock cycles in clock high maximum count. Prescaled by PRESC. Used for bus free detection. Used to time THIGH:MAX.

### 44.10.74 TWI Alternative Command Register

Name: FLEX_TWI_ACR

**Address:** 0xF8034640 (0), 0xF8038640 (1), 0xFC010640 (2), 0xFC014640 (3), 0xFC018640 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | NPEC | NDIR |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| NDATAL | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | PEC | DIR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DATAL | | | | | | | |

- **DATAL: Data Length**

0: No data to send (see Section 44.9.3.11 “Alternative Command”).

1–255: Number of bytes to send during the transfer.

- **DIR: Transfer Direction**

0: Write direction.

1: Read direction.

- **PEC: PEC Request (SMBus Mode only)**

0: The transfer does not use a PEC byte.

1: The transfer uses a PEC byte.

- **NDATAL: Next Data Length**

0: No data to send (see Section 44.9.3.11 “Alternative Command”).

1–255: Number of bytes to send for the next transfer.

- **NDIR: Next Transfer Direction**

0: Write direction.

1: Read direction.

- **NPEC: Next PEC Request (SMBus Mode only)**

0: The next transfer does not use a PEC byte.

1: The next transfer uses a PEC byte.

### 44.10.75 TWI Filter Register

Name: FLEX_TWI_FILTR

**Address:** 0xF8034644 (0), 0xF8038644 (1), 0xFC010644 (2), 0xFC014644 (3), 0xFC018644 (4)

Access: Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | THRES | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | PADFCFG | PADFEN | FILT |

- **FILT: RX Digital Filter**

0: No filtering applied on TWI inputs.

1: TWI input filtering is active. (Only in Standard and Fast modes)

Note: TWI digital input filtering follows a majority decision based on three samples from SDA/SCL lines at peripheral clock frequency.

- **PADFEN: PAD Filter Enable**

0: PAD analog filter is disabled.

1: PAD analog filter is enabled. (The analog filter must be enabled if High-speed mode is enabled.)

- **PADFCFG: PAD Filter Config**

See the electrical characteristics section for filter configuration details.

- **THRES: Digital Filter Threshold**

0: No filtering applied on TWI inputs.

1–7: Maximum pulse width of spikes which will be suppressed by the input filter, defined in peripheral clock cycles.

### 44.10.76 TWI SleepWalking Matching Register

**Name:** FLEX_TWI_SWMR

**Address:** 0xF803464C (0), 0xF803864C (1), 0xFC01064C (2), 0xFC01464C (3), 0xFC01864C (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DATAM | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| – | SADR3 | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | SADR2 | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | SADR1 | | | | | | |

- **SADR1: Slave Address 1**

Slave address 1. The TWI module will match on this additional address if SADR1EN bit is enabled.

- **SADR2: Slave Address 2**

Slave address 2. The TWI module will match on this additional address if SADR2EN bit is enabled.

- **SADR3: Slave Address 3**

Slave address 3. The TWI module will match on this additional address if SADR3EN bit is enabled.

- **DATAM: Data Match**

The TWI module will extend the SleepWalking matching process to the first received data comparing it with DATAM if DATAMEN bit is enabled.

### 44.10.77 TWI FIFO Mode Register

**Name:** FLEX_TWI_FMR

**Address:** 0xF8034650 (0), 0xF8038650 (1), 0xFC010650 (2), 0xFC014650 (3), 0xFC018650 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | RXFTHRES | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | TXFTHRES | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | RXRDYM | | – | – | TXRDYM | |

- **TXRDYM: Transmitter Ready Mode**

If FIFOs are enabled, the TXRDY flag (in FLEX_TWI_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | TXRDY will be at level ‘1’ when at least one data can be written in the Transmit FIFO |
| 0x1 | TWO_DATA | TXRDY will be at level ‘1’ when at least two data can be written in the Transmit FIFO |
| 0x2 | FOUR_DATA | TXRDY will be at level ‘1’ when at least four data can be written in the Transmit FIFO |

- **RXRDYM: Receiver Ready Mode**

If FIFOs are enabled, the RXRDY flag (in FLEX_TWI_SR) behaves as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | RXRDY will be at level ‘1’ when at least one unread data is in the Receive FIFO |
| 0x1 | TWO_DATA | RXRDY will be at level ‘1’ when at least two unread data are in the Receive FIFO |
| 0x2 | FOUR_DATA | RXRDY will be at level ‘1’ when at least four unread data are in the Receive FIFO |

- **TXFTHRES: Transmit FIFO Threshold**

0–16: Defines the Transmit FIFO threshold value (number of data). TXFTH flag in FLEX_TWI_FSR will be set when Transmit FIFO goes from “above” threshold state to “equal or below” threshold state.

- **RXFTHRES: Receive FIFO Threshold**

0–16: Defines the Receive FIFO threshold value (number of data). RXFTH flag in FLEX_TWI_FSR will be set when Receive FIFO goes from “below” threshold state to “equal to or above” threshold state.

### 44.10.78 TWI FIFO Level Register

**Name:** FLEX_TWI_FLR

**Address:** 0xF8034654 (0), 0xF8038654 (1), 0xFC010654 (2), 0xFC014654 (3), 0xFC018654 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFL | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TXFL | | | | | |

- **TXFL: Transmit FIFO Level**

0: There is no data in the Transmit FIFO

1–16: Indicates the number of DATA in the Transmit FIFO

- **RXFL: Receive FIFO Level**

0: There is no unread data in the Receive FIFO

1–16: Indicates the number of unread DATA in the Receive FIFO

### 44.10.79 TWI FIFO Status Register

**Name:** FLEX_TWI_FSR

**Address:** 0xF8034660 (0), 0xF8038660 (1), 0xFC010660 (2), 0xFC014660 (3), 0xFC018660 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: Transmit FIFO Empty Flag (cleared on read)**

0: Transmit FIFO is not empty.

1: Transmit FIFO has been emptied since the last read of FLEX_TWI_FSR.

- **TXFFF: Transmit FIFO Full Flag (cleared on read)**

0: Transmit FIFO is not full.

1: Transmit FIFO has been filled since the last read of FLEX_TWI_FSR.

- **TXFTHF: Transmit FIFO Threshold Flag (cleared on read)**

0: Number of DATA in Transmit FIFO is above TXFTHRES threshold.

1: Number of DATA in Transmit FIFO has reached TXFTHRES threshold since the last read of FLEX_TWI_FSR.

- **RXFEF: Receive FIFO Empty Flag**

0: Receive FIFO is not empty.

1: Receive FIFO has been emptied since the last read of FLEX_TWI_FSR.

- **RXFFF: Receive FIFO Full Flag**

0: Receive FIFO is not empty.

1: Receive FIFO has been filled since the last read of FLEX_TWI_FSR.

- **RXFTHF: Receive FIFO Threshold Flag**

0: Number of unread DATA in Receive FIFO is below RXFTHRES threshold.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES threshold since the last read of FLEX_TWI_FSR.

- **TXFPTEF: Transmit FIFO Pointer Error Flag**

0: No Transmit FIFO pointer occurred

1: Transmit FIFO pointer error occurred. Transceiver must be reset

See Section “FIFO Pointer Error” and Section “FIFO Pointer Error” for details.

- **RXFPTEF: Receive FIFO Pointer Error Flag**

0: No Receive FIFO pointer occurred

1: Receive FIFO pointer error occurred. Receiver must be reset

See Section “FIFO Pointer Error” and Section “FIFO Pointer Error” for details.

### 44.10.80 TWI FIFO Interrupt Enable Register

**Name:** FLEX_TWI_FIER

**Address:** 0xF8034664 (0), 0xF8038664 (1), 0xFC010664 (2), 0xFC014664 (3), 0xFC018664 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Enable**
- **TXFFF: TXFFF Interrupt Enable**
- **TXFTHF: TXFTHF Interrupt Enable**
- **RXFEF: RXFEF Interrupt Enable**
- **RXFFF: RXFFF Interrupt Enable**
- **RXFTHF: RXFTHF Interrupt Enable**
- **TXFPTEF: TXFPTEF Interrupt Enable**
- **RXFPTEF: RXFPTEF Interrupt Enable**

### 44.10.81 TWI FIFO Interrupt Disable Register

**Name:** FLEX_TWI_FIDR

**Address:** 0xF8034668 (0), 0xF8038668 (1), 0xFC010668 (2), 0xFC014668 (3), 0xFC018668 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Disable**
- **TXFFF: TXFFF Interrupt Disable**
- **TXFTHF: TXFTHF Interrupt Disable**
- **RXFEF: RXFEF Interrupt Disable**
- **RXFFF: RXFFF Interrupt Disable**
- **RXFTHF: RXFTHF Interrupt Disable**
- **TXFPTEF: TXFPTEF Interrupt Disable**
- **RXFPTEF: RXFPTEF Interrupt Disable**

### 44.10.82 TWI FIFO Interrupt Mask Register

**Name:** FLEX_TWI_FIMR

**Address:** 0xF803466C (0), 0xF803866C (1), 0xFC01066C (2), 0xFC01466C (3), 0xFC01866C (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |

- **TXFEF: TXFEF Interrupt Mask**
- **TXFFF: TXFFF Interrupt Mask**
- **TXFTHF: TXFTHF Interrupt Mask**
- **RXFEF: RXFEF Interrupt Mask**
- **RXFFF: RXFFF Interrupt Mask**
- **RXFTHF: RXFTHF Interrupt Mask**
- **TXFPTEF: TXFPTEF Interrupt Mask**
- **RXFPTEF: RXFPTEF Interrupt Mask**

### 44.10.83 TWI Write Protection Mode Register

**Name:** FLEX_TWI_WPMR

**Address:** 0xF80346E4 (0), 0xF80386E4 (1), 0xFC0106E4 (2), 0xFC0146E4 (3), 0xFC0186E4 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x545749 (“TWI” in ASCII).

See Section 44.9.7 “TWI Register Write Protection” for the list of registers that can be write-protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x545749 | PASSWD | Writing any other value in this field aborts the write operation of bit WPEN.<br>Always reads as 0 |

### 44.10.84 TWI Write Protection Status Register

**Name:** FLEX_TWI_WPSR

**Address:** 0xF80346E8 (0), 0xF80386E8 (1), 0xFC0106E8 (2), 0xFC0146E8 (3), 0xFC0186E8 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPVSRC | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPVSRC | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protect Violation Status**

0: No Write Protection Violation has occurred since the last read of FLEX_TWI_WPSR.

1: A Write Protection Violation has occurred since the last read of FLEX_TWI_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

# 45. Universal Asynchronous Receiver Transmitter (UART)

## 45.1 Description

The Universal Asynchronous Receiver Transmitter (UART) features a two-pin UART that can be used for communication and trace purposes and offers an ideal medium for in-situ programming solutions.

Moreover, the association with a DMA controller permits packet handling for these tasks with processor time reduced to a minimum.

## 45.2 Embedded Characteristics

- Two-pin UART
  - Independent Receiver and Transmitter with a Common Programmable Baud Rate Generator
  - Baud Rate can be Driven by Processor-Independent Generic Source Clock
  - Even, Odd, Mark or Space Parity Generation
  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Digital Filter on Receive Line
  - Interrupt Generation
  - Support for Two DMA Channels with Connection to Receiver and Transmitter
  - Supports Asynchronous Partial Wakeup on Receive Line Activity (SleepWalking)
  - Comparison Function on Received Character
  - Receiver Timeout
  - Register Write Protection

## 45.3 Block Diagram

**Figure 45-1. UART Block Diagram**

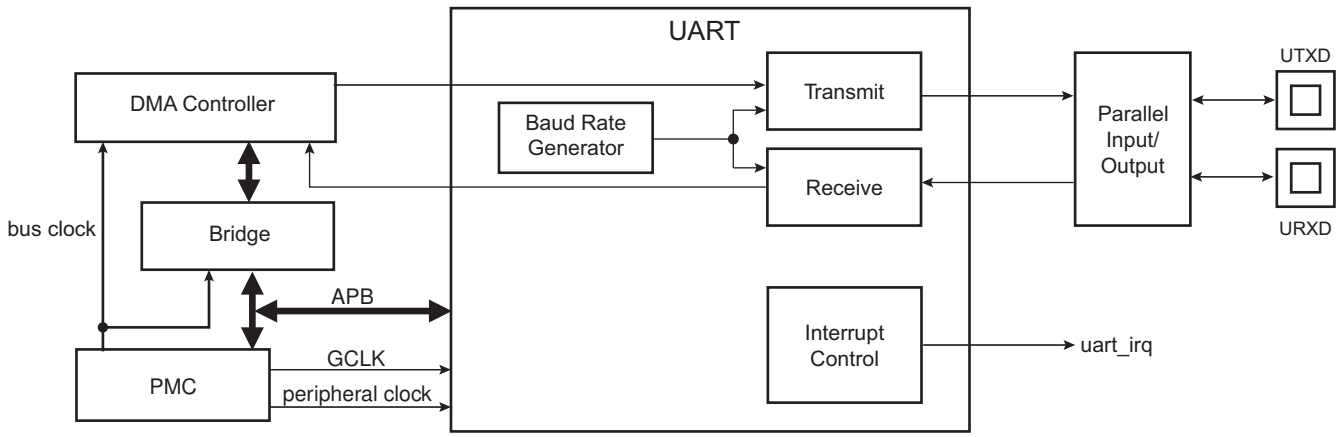



**Table 45-1. UART Pin Description**

| Pin Name | Description | Type |
|---|---|---|
| URXD | UART Receive Data | Input |
| UTXD | UART Transmit Data | Output |

## 45.4 Product Dependencies

### 45.4.1 I/O Lines

The UART pins are multiplexed with PIO lines. The user must first configure the corresponding PIO Controller to enable I/O line operations of the UART.

**Table 45-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| UART0 | URXD0 | PB26 | C |
| UART0 | UTXD0 | PB27 | C |
| UART1 | URXD1 | PC7 | E |
| UART1 | URXD1 | PD2 | A |
| UART1 | UTXD1 | PC8 | E |
| UART1 | UTXD1 | PD3 | A |
| UART2 | URXD2 | PD4 | B |
| UART2 | URXD2 | PD19 | C |
| UART2 | URXD2 | PD23 | A |
| UART2 | UTXD2 | PD5 | B |
| UART2 | UTXD2 | PD20 | C |
| UART2 | UTXD2 | PD24 | A |
| UART3 | URXD3 | PB11 | C |
| UART3 | URXD3 | PC12 | D |
| UART3 | URXD3 | PC31 | C |
| UART3 | UTXD3 | PB12 | C |
| UART3 | UTXD3 | PC13 | D |
| UART3 | UTXD3 | PD0 | C |
| UART4 | URXD4 | PB3 | A |
| UART4 | UTXD4 | PB4 | A |

### 45.4.2 Power Management

The UART clock can be controlled through the Power Management Controller (PMC). In this case, the user must first configure the PMC to enable the UART clock. Usually, the peripheral identifier used for this purpose is 1.

In SleepWalking mode (asynchronous partial wakeup), the PMC must be configured to enable SleepWalking for the UART in the Sleepwalking Enable Register (PMC_SLPWK_ER). Depending on the instructions (requests) provided by the UART to the PMC, the system clock may or may not be automatically provided to the UART.

### 45.4.3 Interrupt Sources

The UART interrupt line is connected to one of the interrupt sources of the Interrupt Controller. Interrupt handling requires programming of the Interrupt Controller before configuring the UART.

**Table 45-3. Peripheral IDs**

| Instance | ID |
|---|---|
| UART0 | 24 |
| UART1 | 25 |
| UART2 | 26 |
| UART3 | 27 |
| UART4 | 28 |

## 45.5 Functional Description

The UART operates in Asynchronous mode only and supports only 8-bit character handling (with parity). It has no clock pin.

The UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

### 45.5.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

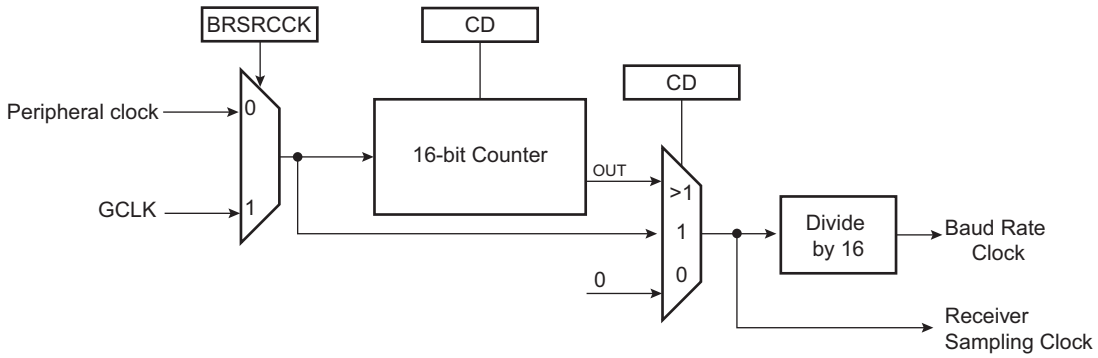
The baud rate clock is the peripheral clock divided by 16 times the clock divisor (CD) value written in the Baud Rate Generator register (UART_BRGR). If UART_BRGR is set to 0, the baud rate clock is disabled and the UART remains inactive. The maximum allowable baud rate is peripheral clock or GCLK divided by 16. The minimum allowable baud rate is peripheral clock divided by (16 x 65536). The clock source driving the baud rate generator (peripheral clock or GCLK) can be selected by writing the bit BRSRCCK in UART_MR.

If GCLK is selected, the baud rate is independent of the processor/bus clock. Thus the processor clock can be changed while UART is enabled. The processor clock frequency changes must be performed only by programming the field PRES in PMC_MCKR (see PMC section). Other methods to modify the processor/bus clock frequency (PLL multiplier, etc.) are forbidden when UART is enabled.

The peripheral clock frequency must be at least three times higher than GCLK.

**Figure 45-2.**

**Figure 45-3. Baud Rate Generator**



### 45.5.2 Receiver

#### 45.5.2.1 Receiver Reset, Enable and Disable

After device reset, the UART receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the Control Register (UART_CR) with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing UART_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The receiver can be put in reset state by writing UART_CR with the bit RSTRX at 1. In this case, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

#### 45.5.2.2 Start Detection and Data Sampling

The UART only supports asynchronous operations, and this affects only its receiver. The UART receiver detects the start of a received character by sampling the URXD signal until it detects a valid start bit. A low level (space) on URXD is interpreted as a valid start bit if it is detected for more than seven cycles of the sampling clock, which is 16 times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

When a valid start bit has been detected, the receiver samples the URXD at the theoretical midpoint of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the bit sampling point is eight cycles (0.5-bit period) after the start of the bit. The first sampling point is therefore 24 cycles (1.5-bit periods) after detecting the falling edge of the start bit.

Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

**Figure 45-4. Start Bit Detection**

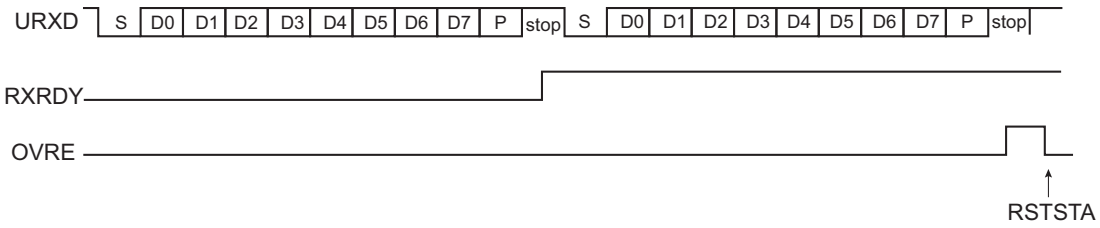
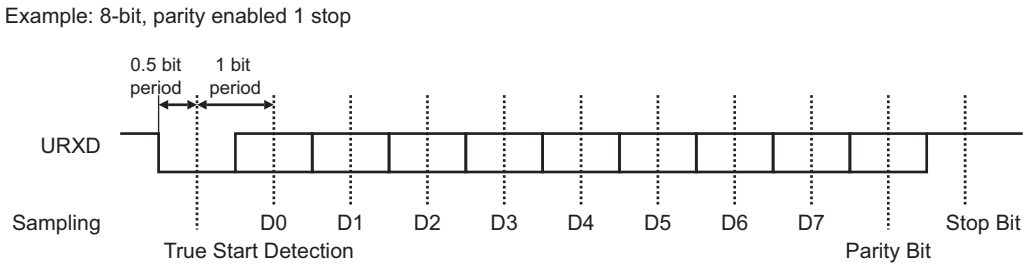


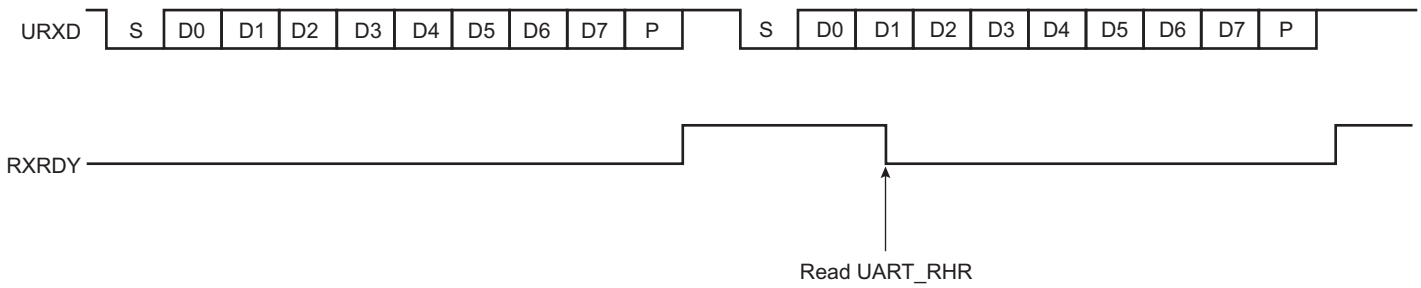

**Figure 45-5. Character Reception**




#### 45.5.2.3 Receiver Ready

When a complete character is received, it is transferred to the Receive Holding Register (UART_RHR) and the RXRDY status bit in the Status Register (UART_SR) is set. The bit RXRDY is automatically cleared when UART_RHR is read.

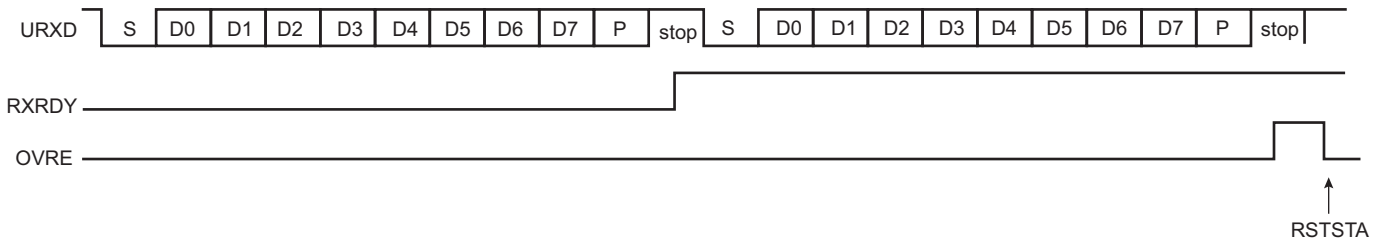
**Figure 45-6. Receiver Ready**




### 45.5.2.4 Receiver Overrun

The OVRE status bit in UART_SR is set if UART_RHR has not been read by the software (or the DMA Controller) since the last transfer, the RXRDY bit is still set and a new character is received. OVRE is cleared when the software writes a 1 to the bit RSTSTA (Reset Status) in UART_CR.

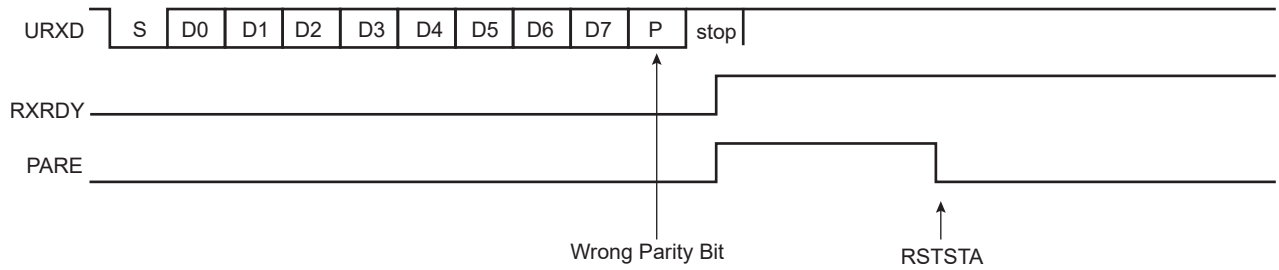
**Figure 45-7. Receiver Overrun**




### 45.5.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in the Mode Register (UART_MR). It then compares the result with the received parity bit. If different, the parity error bit PARE in UART_SR is set at the same time RXRDY is set. The parity bit is cleared when UART_CR is written with the bit RSTSTA (Reset Status) at 1. If a new character is received before the reset status command is written, the PARE bit remains at 1.

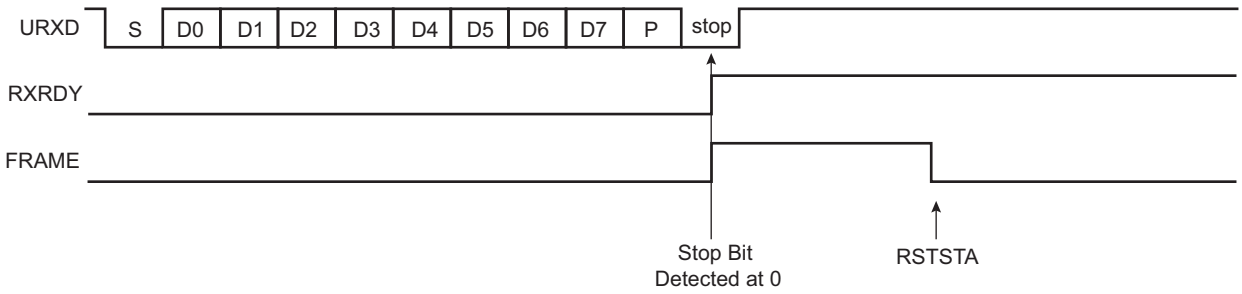
**Figure 45-8. Parity Error**




### 45.5.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in UART_SR is set at the same time the RXRDY bit is set. The FRAME bit remains high until the Control Register (UART_CR) is written with the bit RSTSTA at 1.

**Figure 45-9. Receiver Framing Error**




### 45.5.2.7 Receiver Digital Filter

The UART embeds a digital filter on the receive line. It is disabled by default and can be enabled by writing a logical 1 in the FILTER bit of UART_MR. When enabled, the receive line is sampled using the 16x bit clock and a three-sample filter (majority 2 over 3) determines the value of the line.

### 45.5.2.8 Receiver Timeout

The Receiver Timeout provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a timeout is detected, the bit TIMEOUT in the UART_SR rises and can generate an interrupt, thus indicating to the driver an end of frame.

The timeout delay period (during which the receiver waits for a new character) is programmed in the TO field of the Receiver Timeout register (UART_RTOR). If the TO field is written to 0, the Receiver Timeout is disabled and no timeout is detected. The TIMEOUT bit in the UART_SR remains at 0. Otherwise, the receiver loads an 8-bit counter with the value programmed in TO. This counter is decremented at each bit period and reloaded each time a new character is received. If the counter reaches 0, the TIMEOUT bit UART_SR rises. Then, the user can either:

- stop the counter clock until a new character is received. This is performed by writing a one to the STTTO (start Timeout) bit in the UART_CR. In this case, the idle state on RXD before a new character is received does not provide a timeout. This prevents having to handle an interrupt before a character is received and allows waiting for the next idle state on RXD after a frame is received, or
- obtain an interrupt while no character is received. This is performed by writing a one to the RETTO (Reload and Start Timeout) bit in the UART_CR. If RETTO is performed, the counter starts counting down immediately from the TO value. This enables generation of a periodic interrupt so that a user timeout can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a timeout. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the TO value. This enables generation of a periodic interrupt so that a user timeout can be handled, for example when no key is pressed on a keyboard.

Figure 45-10 shows the block diagram of the Receiver Timeout feature.

**Figure 45-10. Receiver Timeout Block Diagram**

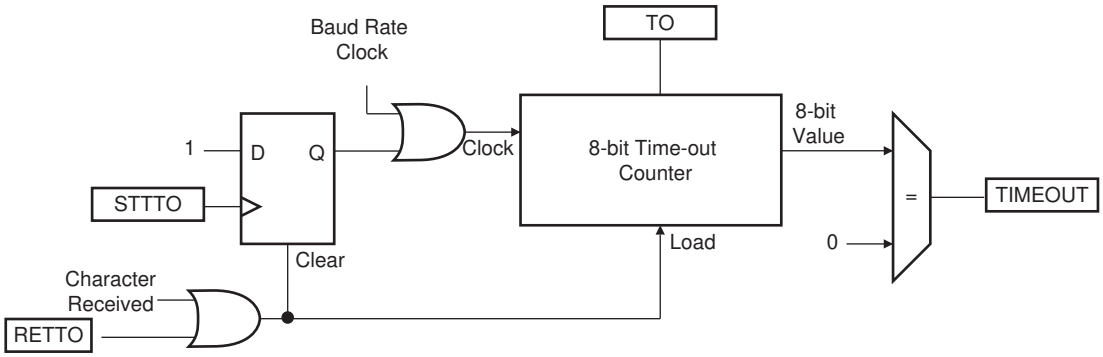



Table 45-4 gives the maximum timeout period for some standard baud rates.

**Table 45-4. Maximum Timeout Period**

| Baud Rate (bit/s) | Bit Time (µs) | Timeout (µs) |
|---|---|---|
| 600 | 1,667 | 425,085 |
| 1,200 | 833 | 212,415 |
| 2,400 | 417 | 106,335 |
| 4,800 | 208 | 53,040 |
| 9,600 | 104 | 26,520 |
| 14,400 | 69 | 17,595 |
| 19,200 | 52 | 13,260 |
| 28,800 | 35 | 8,925 |
| 38,400 | 26 | 6,630 |
| 56,000 | 18 | 4,590 |
| 57,600 | 17 | 4,335 |
| 200,000 | 5 | 1,275 |

## 45.5.3 Transmitter

### 45.5.3.1 Transmitter Reset, Enable and Disable

After device reset, the UART transmitter is disabled and must be enabled before being used. The transmitter is enabled by writing UART_CR with the bit TXEN at 1. From this command, the transmitter waits for a character to be written in the Transmit Holding Register (UART_THR) before actually starting the transmission.

The programmer can disable the transmitter by writing UART_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the internal shift register and/or a character has been written in the UART_THR, the characters are completed before the transmitter is actually stopped.

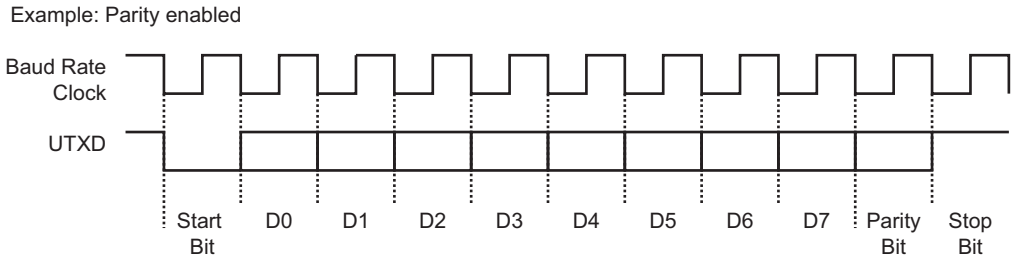
The programmer can also put the transmitter in its reset state by writing the UART_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

### 45.5.3.2 Transmit Format

The UART transmitter drives the pin UTXD at the baud rate clock speed. The line is driven depending on the format defined in UART_MR and the data stored in the internal shift register. One start bit at level 0, then the 8 data bits, from the lowest to the highest bit, one optional parity bit and one stop bit at 1 are consecutively shifted

out as shown in the following figure. The field PARE in UART_MR defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

**Figure 45-11. Character Transmission**

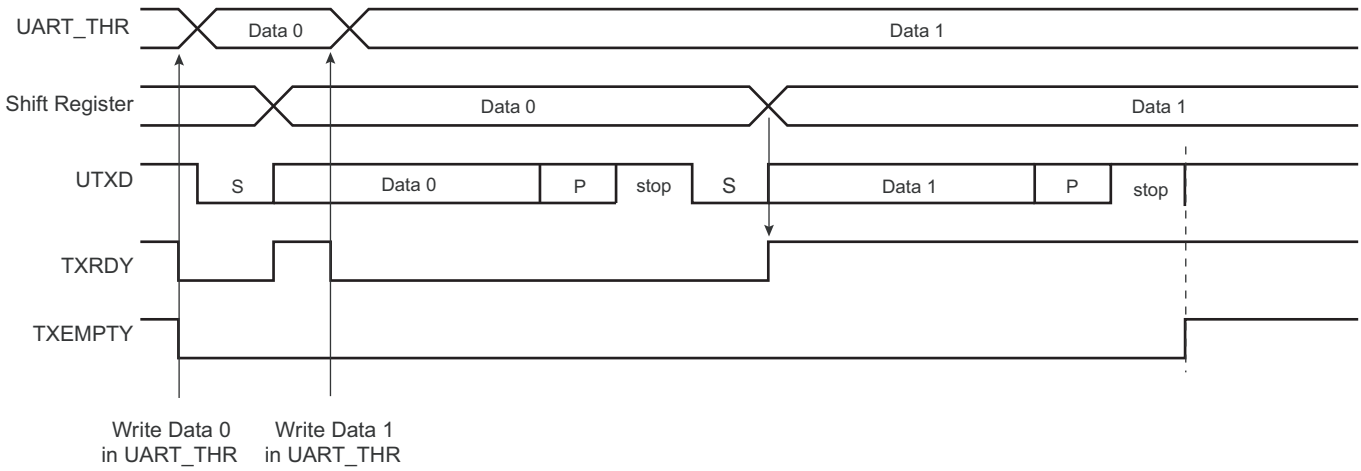



#### 45.5.3.3 Transmitter Control

When the transmitter is enabled, the bit TXRDY (Transmitter Ready) is set in UART_SR. The transmission starts when the programmer writes in the UART_THR, and after the written character is transferred from UART_THR to the internal shift register. The TXRDY bit remains high until a second character is written in UART_THR. As soon as the first character is completed, the last character written in UART_THR is transferred into the internal shift register and TXRDY rises again, showing that the holding register is empty.

When both the internal shift register and UART_THR are empty, i.e., all the characters written in UART_THR have been processed, the TXEMPTY bit rises after the last stop bit has been completed.

**Figure 45-12. Transmitter Control**




### 45.5.4 DMA Support

Both the receiver and the transmitter of the UART are connected to a DMA Controller (DMAC) channel.

The DMA Controller channels are programmed via registers that are mapped within the DMAC user interface.

### 45.5.5 Comparison Function on Received Character

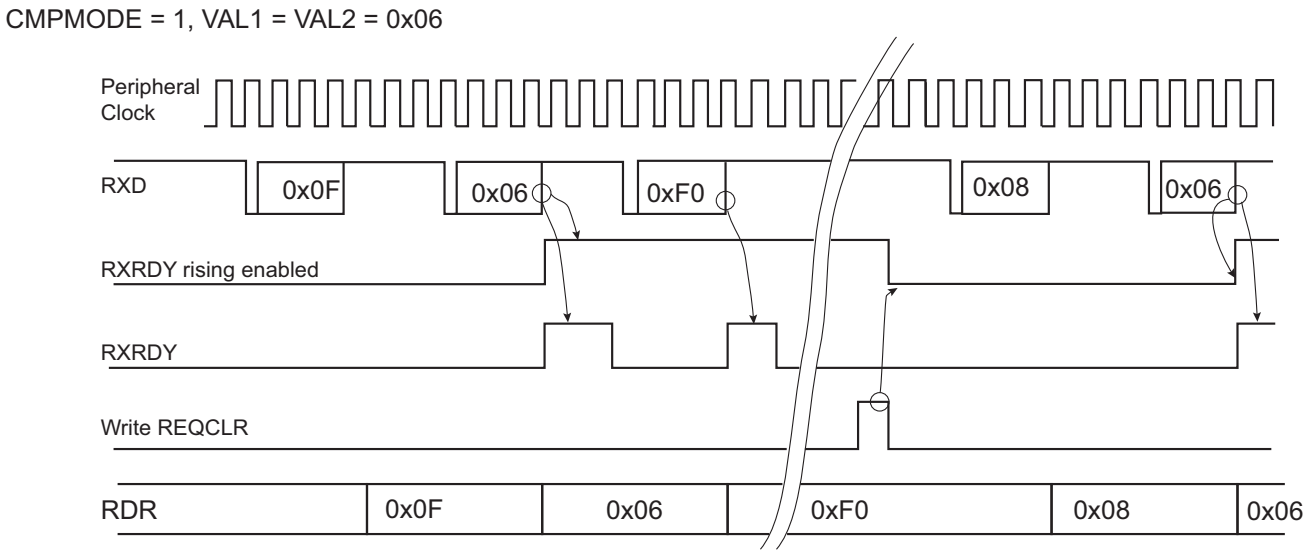
When a comparison is performed on a received character, the result of the comparison is reported on the CMP flag in UART_SR when UART_RHR is loaded with the new received character. The CMP flag is cleared by writing a one to the RSTSTA bit in UART_CR.

UART_CMPR (see Section 45.6.10 “UART Comparison Register”) can be programmed to provide different comparison methods. These are listed below:

- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to 1 if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the flag CMP is set to 1 if either received character equals VAL1 or VAL2.

By programming the CMPMODE bit to 1, the comparison function result triggers the start of the loading of UART_RHR (see Figure 45-13). The trigger condition occurs as soon as the received character value matches the condition defined by the programming of VAL1, VAL2 and CMPPAR in UART_CMPR. The comparison trigger event can be restarted by writing a one to the REQCLR bit in UART_CR.

**Figure 45-13. Receive Holding Register Management**




### 45.5.6 Asynchronous and Partial Wakeup (SleepWalking)

Asynchronous and partial wakeup (SleepWalking) is a means of data preprocessing that qualifies an incoming event, thus allowing the UART to decide whether or not to wake up the system. SleepWalking is used primarily when the system is in Wait mode (refer to section “Power Management Controller (PMC)”) but can also be enabled when the system is fully running.

No access must be performed in the UART between the enable of asynchronous partial wakeup and the wakeup performed by the UART.

If the system is in Wait mode and asynchronous and partial wakeup is enabled, the maximum baud rate that can be achieved equals 19200.

If the system is running or in Sleep mode, the maximum baud rate that can be achieved equals 115200 or higher. This limit is bounded by the peripheral clock frequency divided by 16.

The UART_RHR must be read before enabling asynchronous and partial wakeup.

When SleepWalking is enabled for the UART (see the PMC section), the PMC decodes a clock request from the UART. The request is generated as soon as there is a falling edge on the RXD line as this may indicate the beginning of a start bit. If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the UART.

As soon as the clock is provided by the PMC, the UART processes the received frame and compares the received character with VAL1 and VAL2 in UART_CMPR (Section 45.6.10 “UART Comparison Register”).

The UART instructs the PMC to disable the clock if the received character value does not meet the conditions defined by VAL1 and VAL2 fields in UART_CMPR (see Figure 45-15).

If the received character value meets the conditions, the UART instructs the PMC to exit the full system from Wait mode (see Figure 45-14).

The VAL1 and VAL2 fields can be programmed to provide different comparison methods and thus matching conditions.

- If VAL1 equals VAL2, then the comparison is performed on a single value and the wakeup is triggered if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 wakes up the system.
- If VAL1 is strictly higher than VAL2, then the wakeup is triggered if the received character equals VAL1 or VAL2.
- If VAL1 = 0 and VAL2 = 255, the wakeup is triggered as soon as a character is received.

The matching condition can be configured to include the parity bit (CMPPAR in UART_CMPR). Thus, if the received data matches the comparison condition defined by VAL1 and VAL2 but a parity error is encountered, the matching condition is cancelled and the UART instructs the PMC to disable the clock (see Figure 45-15).

If the processor and peripherals are running, the UART can be configured in Asynchronous and partial wakeup mode by enabling the PMC_SLPWK_ER (see PMC section). When activity is detected on the receive line, the UART requests the clock from the PMC and the comparison is performed. If there is a comparison match, the UART continues to request the clock. If there is no match, the clock is switched off for the UART only, until a new activity is detected.

The CMPMODE configuration has no effect when Asynchronous and Partial Wakeup mode is enabled for the UART (see PMC_SLPWK_ER in the PMC section).

When the system is kept in active/running mode and the UART enters Asynchronous and Partial Wakeup mode, the flag CMP must be programmed as the unique source of the UART interrupt.

When the system exits Wait mode as the result of a matching condition, the RXRDY flag is used to determine if the UART is the source of exit.

Note: If the SleepWalking function is enabled on the UART, a divide by 8 of the peripheral clock versus the bus clock is not possible. Other dividers can be used with no constraints.

**Figure 45-14. Asynchronous Wakeup Use Case Examples**

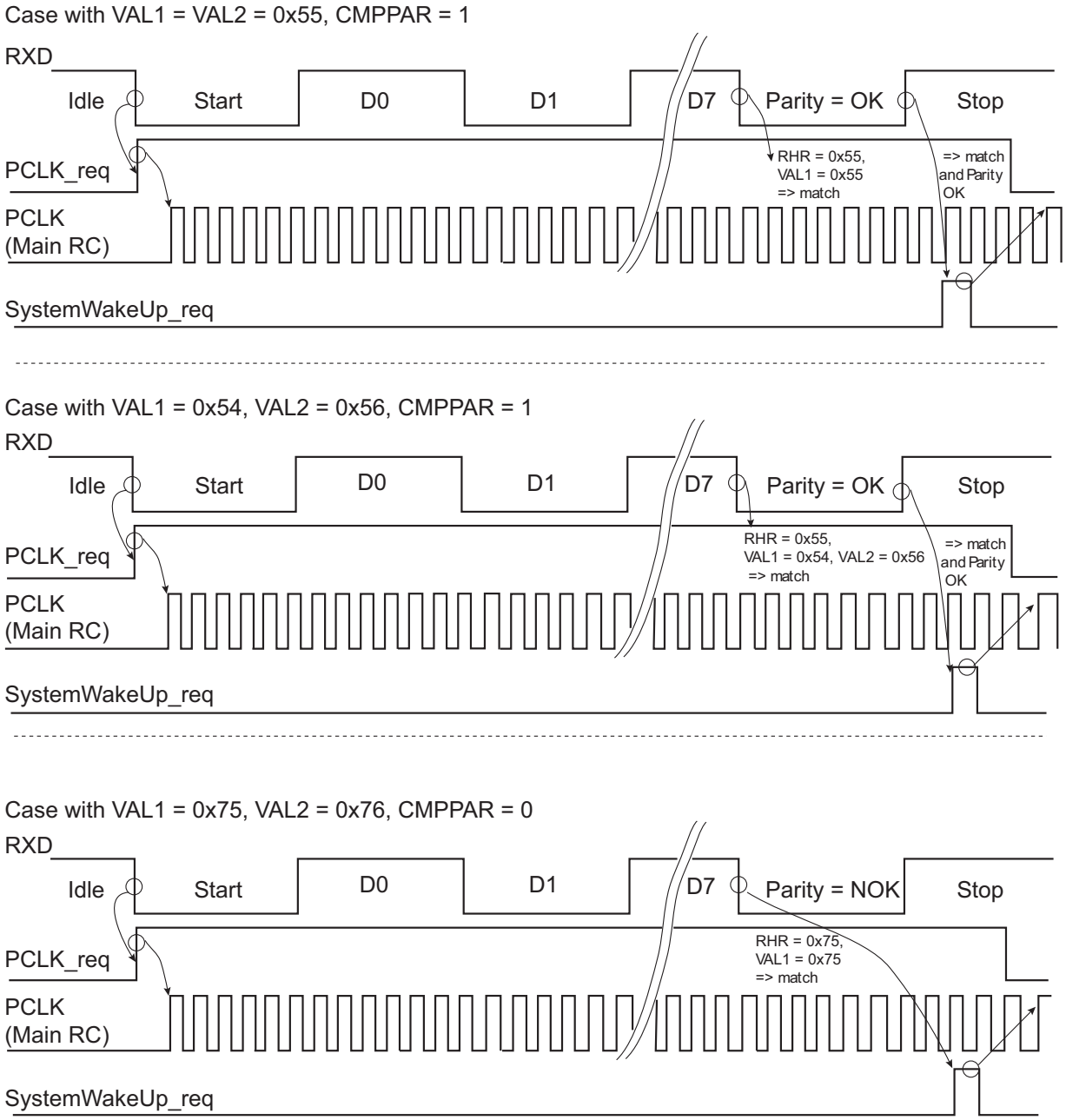
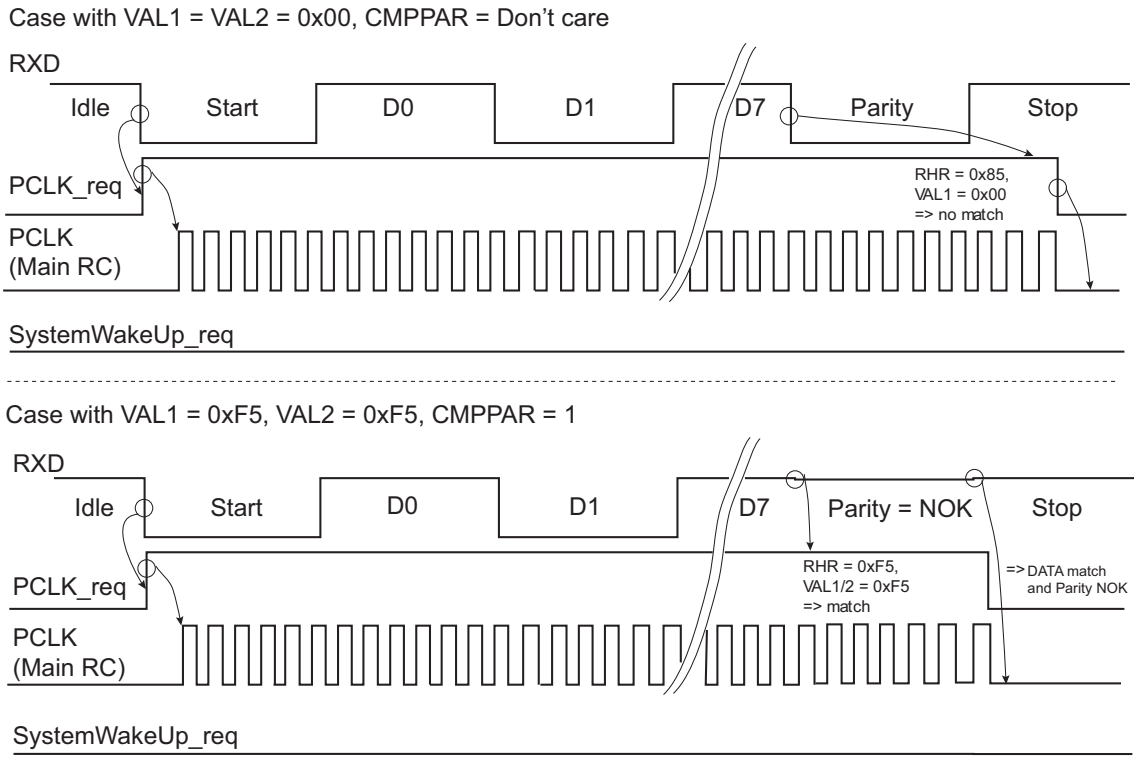


**Figure 45-15. Asynchronous Event Generating Only Partial Wakeup**




### 45.5.7 Register Write Protection

To prevent any single software error from corrupting UART behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the UART Write Protection Mode Register (UART_WPMR).

The following registers can be write-protected:

- UART Mode Register
- UART Baud Rate Generator Register
- UART Comparison Register

### 45.5.8 Test Modes

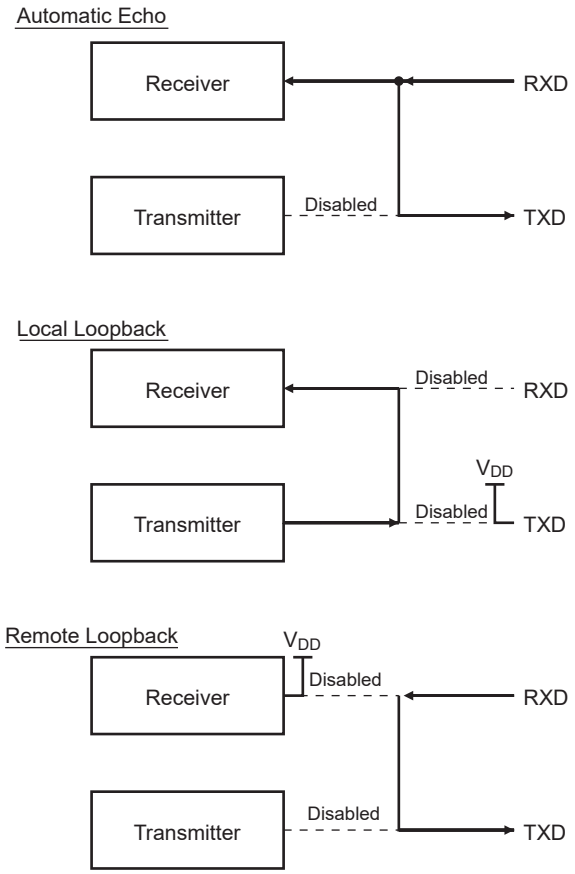
The UART supports three test modes. These modes of operation are programmed by using the CHMODE field in UART_MR.

The Automatic Echo mode allows a bit-by-bit retransmission. When a bit is received on the URXD line, it is sent to the UTXD line. The transmitter operates normally, but has no effect on the UTXD line.

The Local Loopback mode allows the transmitted characters to be received. UTXD and URXD pins are not used and the output of the transmitter is internally connected to the input of the receiver. The URXD pin level has no effect and the UTXD line is held high, as in idle state.

The Remote Loopback mode directly connects the URXD pin to the UTXD line. The transmitter and the receiver are disabled and have no effect. This mode allows a bit-by-bit retransmission.

**Figure 45-16. Test Modes**



## 45.6 Universal Asynchronous Receiver Transmitter (UART) User Interface

**Table 45-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | Control Register | UART_CR | Write-only | – |
| 0x0004 | Mode Register | UART_MR | Read/Write | 0x0 |
| 0x0008 | Interrupt Enable Register | UART_IER | Write-only | – |
| 0x000C | Interrupt Disable Register | UART_IDR | Write-only | – |
| 0x0010 | Interrupt Mask Register | UART_IMR | Read-only | 0x0 |
| 0x0014 | Status Register | UART_SR | Read-only | – |
| 0x0018 | Receive Holding Register | UART_RHR | Read-only | 0x0 |
| 0x001C | Transmit Holding Register | UART_THR | Write-only | – |
| 0x0020 | Baud Rate Generator Register | UART_BRGR | Read/Write | 0x0 |
| 0x0024 | Comparison Register | UART_CMPR | Read/Write | 0x0 |
| 0x0028 | Receiver Timeout Register | UART_RTOR | Read/Write | 0x0 |
| 0x002C–0x003C | Reserved | – | – | – |
| 0x0040–0x00E0 | Reserved | – | – | – |
| 0x00E4 | Write Protection Mode Register | UART_WPMR | Read/Write | 0x0 |
| 0x00E8 | Reserved | – | – | – |
| 0x00EC–0x00FC | Reserved | – | – | – |

### 45.6.1 UART Control Register

**Name:** UART_CR

**Address:** 0xF801C000 (0), 0xF8020000 (1), 0xF8024000 (2), 0xFC008000 (3), 0xFC00C000 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | REQCLR | STTTO | RETTO | – | RSTSTA |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXDIS | TXEN | RXDIS | RXEN | RSTTX | RSTRX | – | – |

- **RSTRX: Reset Receiver**

0: No effect.

1: The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0: No effect.

1: The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0: No effect.

1: The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0: No effect.

1: The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: The transmitter is disabled. If a character is being processed and a character has been written in the UART_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status**

0: No effect.

1: Resets the status bits PARE, FRAME, CMP and OVRE in the UART_SR.

- **RETTO: Rearm Timeout**

0: No effect.

1: Restarts timeout.

- **STTTO: Start Timeout**

0: No effect.

1: Starts waiting for a character before clocking the timeout counter. Resets status bit TIMEOUT in UART_SR.

- **REQCLR: Request Clear**

SleepWalking enabled:

0: No effect.

1: Bit REQCLR clears the potential clock request currently issued by UART, thus the potential system wakeup is cancelled.

SleepWalking disabled:

0: No effect.

1: Bit REQCLR restarts the comparison trigger to enable receive holding register loading.

### 45.6.2 UART Mode Register

**Name:** UART_MR

**Address:** 0xF801C004 (0), 0xF8020004 (1), 0xF8024004 (2), 0xFC008004 (3), 0xFC00C004 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CHMODE | | – | BRSRCCK | PAR | | | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | FILTER | – | – | – | – |

- **FILTER: Receiver Digital Filter**

0 (DISABLED): UART does not filter the receive line.

1 (ENABLED): UART filters the receive line using a three-sample filter (16x-bit clock) (2 over 3 majority).

- **PAR: Parity Type**

| Value | Name | Description |
|---|---|---|
| 0 | EVEN | Even Parity |
| 1 | ODD | Odd Parity |
| 2 | SPACE | Space: parity forced to 0 |
| 3 | MARK | Mark: parity forced to 1 |
| 4 | NO | No parity |

- **BRSRCCK: Baud Rate Source Clock**

0 (PERIPH_CLK): The baud rate is driven by the peripheral clock

1 (GCLK): The baud rate is driven by a PMC-programmable clock GCLK (see section Power Management Controller (PMC)).

- **CHMODE: Channel Mode**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Normal mode |
| 1 | AUTOMATIC | Automatic echo |
| 2 | LOCAL_LOOPBACK | Local loopback |
| 3 | REMOTE_LOOPBACK | Remote loopback |

### 45.6.3 UART Interrupt Enable Register

**Name:** UART_IER

**Address:** 0xF801C008 (0), 0xF8020008 (1), 0xF8024008 (2), 0xFC008008 (3), 0xFC00C008 (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CMP | – | – | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RXRDY: Enable RXRDY Interrupt**
- **TXRDY: Enable TXRDY Interrupt**
- **OVRE: Enable Overrun Error Interrupt**
- **FRAME: Enable Framing Error Interrupt**
- **PARE: Enable Parity Error Interrupt**
- **TIMEOUT: Enable Timeout Interrupt**
- **TXEMPTY: Enable TXEMPTY Interrupt**
- **CMP: Enable Comparison Interrupt**

### 45.6.4 UART Interrupt Disable Register

**Name:** UART_IDR

**Address:** 0xF801C00C (0), 0xF802000C (1), 0xF802400C (2), 0xFC00800C (3), 0xFC00C00C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CMP | – | – | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RXRDY: Disable RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **OVRE: Disable Overrun Error Interrupt**
- **FRAME: Disable Framing Error Interrupt**
- **PARE: Disable Parity Error Interrupt**
- **TIMEOUT: Disable Timeout Interrupt**
- **TXEMPTY: Disable TXEMPTY Interrupt**
- **CMP: Disable Comparison Interrupt**

### 45.6.5 UART Interrupt Mask Register

**Name:** UART_IMR

**Address:** 0xF801C010 (0), 0xF8020010 (1), 0xF8024010 (2), 0xFC008010 (3), 0xFC00C010 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CMP | – | – | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

- **RXRDY: Mask RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **OVRE: Mask Overrun Error Interrupt**
- **FRAME: Mask Framing Error Interrupt**
- **PARE: Mask Parity Error Interrupt**
- **TIMEOUT: Mask Timeout Interrupt**
- **TXEMPTY: Mask TXEMPTY Interrupt**
- **CMP: Mask Comparison Interrupt**

### 45.6.6 UART Status Register

**Name:** UART_SR

**Address:** 0xF801C014 (0), 0xF8020014 (1), 0xF8024014 (2), 0xFC008014 (3), 0xFC00C014 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CMP | – | – | – | – | – | TXEMPTY | TIMEOUT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PARE | FRAME | OVRE | – | – | – | TXRDY | RXRDY |

- **RXRDY: Receiver Ready**

0: No character has been received since the last read of the UART_RHR, or the receiver is disabled.

1: At least one complete character has been received, transferred to UART_RHR and not yet read.

- **TXRDY: Transmitter Ready**

0: A character has been written to UART_THR and not yet transferred to the internal shift register, or the transmitter is disabled.

1: There is no character written to UART_THR not yet transferred to the internal shift register.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No framing error has occurred since the last RSTSTA.

1: At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has occurred since the last RSTSTA.

1: At least one parity error has occurred since the last RSTSTA.

- **TIMEOUT: Receiver Timeout**

0: There has not been a timeout since the last Start Timeout command (STTTO in UART_CR) or the Timeout Register is 0.

1: There has been a timeout since the last Start Timeout command (STTTO in UART_CR).

- **TXEMPTY: Transmitter Empty**

0: There are characters in UART_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1: There are no characters in UART_THR and there are no characters being processed by the transmitter.

- **CMP: Comparison Match**

0: No received character matches the comparison criteria programmed in VAL1, VAL2 fields and in CMPPAR bit since the last RSTSTA.

1: The received character matches the comparison criteria.

### 45.6.7 UART Receiver Holding Register

**Name:** UART_RHR

**Address:** 0xF801C018 (0), 0xF8020018 (1), 0xF8024018 (2), 0xFC008018 (3), 0xFC00C018 (4)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RXCHR | | | | | | | |

- **RXCHR: Received Character**

Last received character if RXRDY is set.

### 45.6.8 UART Transmit Holding Register

**Name:** UART_THR

**Address:** 0xF801C01C (0), 0xF802001C (1), 0xF802401C (2), 0xFC00801C (3), 0xFC00C01C (4)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TXCHR | | | | | | | |

- **TXCHR: Character to be Transmitted**

Next character to be transmitted after the current character if TXRDY is not set.

### 45.6.9 UART Baud Rate Generator Register

**Name:** UART_BRGR

**Address:** 0xF801C020 (0), 0xF8020020 (1), 0xF8024020 (2), 0xFC008020 (3), 0xFC00C020 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CD | | | | | | | |

- **CD: Clock Divisor**

0: Baud rate clock is disabled

1 to 65,535:

If BRSRCCK = 0:

$$CD = \frac{f_{peripheral\ clock}}{16 \times Baud\ Rate}$$

If BRSRCCK = 1:

$$CD = \frac{f_{GCLKx}}{16 \times Baud\ Rate}$$

### 45.6.10 UART Comparison Register

**Name:** UART_CMPR

**Address:** 0xF801C024 (0), 0xF8020024 (1), 0xF8024024 (2), 0xFC008024 (3), 0xFC00C024 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| VAL2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | CMPPAR | – | CMPMODE | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VAL1 | | | | | | | |

- **VAL1: First Comparison Value for Received Character**

0–255: The received character must be higher or equal to the value of VAL1 and lower or equal to VAL2 to set CMP flag in UART_SR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the UART requests a system wakeup if the condition is met.

- **CMPMODE: Comparison Mode**

| Value | Name | Description |
|---|---|---|
| 0 | FLAG_ONLY | Any character is received and comparison function drives CMP flag. |
| 1 | START_CONDITION | Comparison condition must be met to start reception. |

- **CMPPAR: Compare Parity**

0: The parity is not checked and a bad parity cannot prevent from waking up the system.

1: The parity is checked and a matching condition on data can be cancelled by an error on parity bit, so no wakeup is performed.

- **VAL2: Second Comparison Value for Received Character**

0–255: The received character must be lower or equal to the value of VAL2 and higher or equal to VAL1 to set CMP flag in UART_SR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the UART requests a system wakeup if condition is met.

### 45.6.11 UART Receiver Timeout Register

**Name:** UART_RTOR

**Address:** 0xF801C028 (0), 0xF8020028 (1), 0xF8024028 (2), 0xFC008028 (3), 0xFC00C028 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TO | | | | | | | |

- **TO: Timeout Value**

0: The receiver timeout is disabled.

1–255: The receiver timeout is enabled and the timeout delay is TO x bit period.

### 45.6.12 UART Write Protection Mode Register

**Name:** UART_WPMR

**Address:** 0xF801C0E4 (0), 0xF80200E4 (1), 0xF80240E4 (2), 0xFC0080E4 (3), 0xFC00C0E4 (4)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x554152 (UART in ASCII).

1: Enables the write protection if WPKEY corresponds to 0x554152 (UART in ASCII).

See Section 45.5.7 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x554152 | PASSWD | Writing any other value in this field aborts the write operation.<br>Always reads as 0. |

# 46. Serial Peripheral Interface (SPI)

## 46.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a Shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the “master”' which controls the data flow, while the other devices act as “slaves" which have data shifted into and out by the master. Different CPUs can take turn being masters (multiple master protocol, contrary to single master protocol where one CPU is always the master while all of the others are always slaves). One master can simultaneously shift data into multiple slaves. However, only one slave can drive its output to write data back to the master at any given time.

A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

The SPI system consists of two data lines and two control lines:

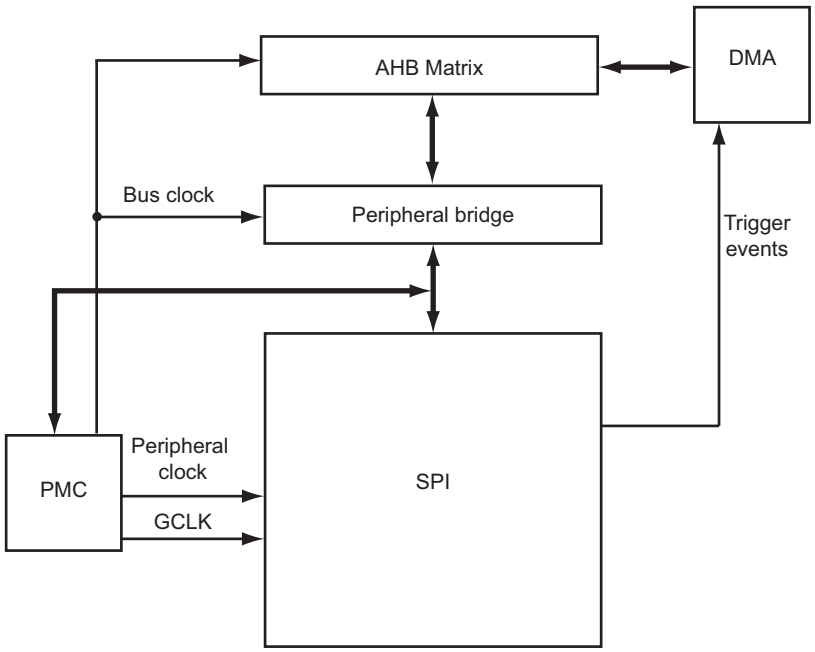
- Master Out Slave In (MOSI)—This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO)—This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK)—This control line is driven by the master and regulates the flow of the data bits. The master can transmit data at a variety of baud rates; there is one SPCK pulse for each bit that is transmitted.
- Slave Select (NSS)—This control line allows slaves to be turned on and off by hardware.

## 46.2 Embedded Characteristics

- Master or Slave Serial Peripheral Bus Interface
  - 8-bit to 16-bit programmable data length per chip select
  - Programmable phase and polarity per chip select
  - Programmable transfer delay between consecutive transfers and delay before SPI clock per chip select
  - Programmable delay between chip selects
  - Selectable mode fault detection
- Master Mode can drive SPCK up to Peripheral Clock
- 16-data Transmit and Receive FIFOs
- Master Mode Bit Rate can be Independent of the Processor/Peripheral Clock
- Slave mode operates on SPCK, asynchronously with core and bus clock
- Four chip selects with external decoder support allow communication with up to 15 peripherals
- Communication with Serial External Devices Supported
  - Serial memories, such as DataFlash and 3-wire EEPROMs
  - Serial peripherals, such as ADCs, DACs, LCD controllers, CAN controllers and sensors
  - External coprocessors
- Connection to DMA Channel Capabilities, Optimizing Data Transfers
  - One channel for the receiver
  - One channel for the transmitter
- Register Write Protection

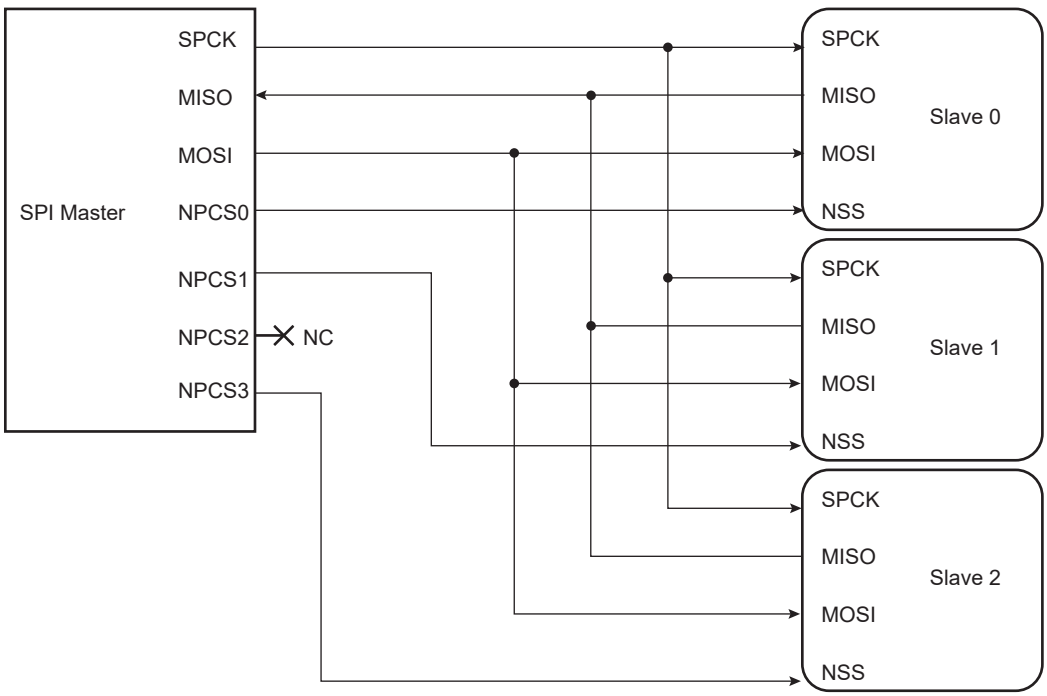
## 46.3 Block Diagram

**Figure 46-1. Block Diagram**




## 46.4 Application Block Diagram

**Figure 46-2. Application Block Diagram: Single Master/Multiple Slave Implementation**



## 46.5 Signal Description

**Table 46-1. Signal Description**

| Pin Name | Pin Description | Type | |
|---|---|---|---|
| | | Master | Slave |
| MISO | Master In Slave Out | Input | Output |
| MOSI | Master Out Slave In | Output | Input |
| SPCK | Serial Clock | Output | Input |
| NPCS1–NPCS3 | Peripheral Chip Selects | Output | Unused |
| NPCS0/NSS | Peripheral Chip Select/Slave Select | Output | Input |

## 46.6 Product Dependencies

### 46.6.1 I/O Lines

The pins used for interfacing the compliant external devices can be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

**Table 46-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| SPI0 | SPI0_MISO | PA16 | A |
| SPI0 | SPI0_MISO | PA31 | C |
| SPI0 | SPI0_MOSI | PA15 | A |
| SPI0 | SPI0_MOSI | PB0 | C |
| SPI0 | SPI0_NPCS0 | PA17 | A |
| SPI0 | SPI0_NPCS0 | PA30 | C |
| SPI0 | SPI0_NPCS1 | PA18 | A |
| SPI0 | SPI0_NPCS1 | PA29 | C |
| SPI0 | SPI0_NPCS2 | PA19 | A |
| SPI0 | SPI0_NPCS2 | PA27 | C |
| SPI0 | SPI0_NPCS3 | PA20 | A |
| SPI0 | SPI0_NPCS3 | PA28 | C |
| SPI0 | SPI0_SPCK | PA14 | A |
| SPI0 | SPI0_SPCK | PB1 | C |
| SPI1 | SPI1_MISO | PA24 | D |
| SPI1 | SPI1_MISO | PC3 | D |
| SPI1 | SPI1_MISO | PD27 | A |
| SPI1 | SPI1_MOSI | PA23 | D |
| SPI1 | SPI1_MOSI | PC2 | D |
| SPI1 | SPI1_MOSI | PD26 | A |
| SPI1 | SPI1_NPCS0 | PA25 | D |
| SPI1 | SPI1_NPCS0 | PC4 | D |

Table 46-2. I/O Lines (Continued)

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| SPI1 | SPI1_NPCS0 | PD28 | A |
| SPI1 | SPI1_NPCS1 | PA26 | D |
| SPI1 | SPI1_NPCS1 | PC5 | D |
| SPI1 | SPI1_NPCS1 | PD29 | A |
| SPI1 | SPI1_NPCS2 | PA27 | D |
| SPI1 | SPI1_NPCS2 | PC6 | D |
| SPI1 | SPI1_NPCS2 | PD30 | A |
| SPI1 | SPI1_NPCS3 | PA28 | D |
| SPI1 | SPI1_NPCS3 | PC7 | D |
| SPI1 | SPI1_SPCK | PA22 | D |
| SPI1 | SPI1_SPCK | PC1 | D |
| SPI1 | SPI1_SPCK | PD25 | A |

### 46.6.2 Power Management

The SPI can be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the SPI clock.

### 46.6.3 Interrupt

The SPI interface has an interrupt line connected to the interrupt controller. Handling the SPI interrupt requires programming the interrupt controller before configuring the SPI.

Table 46-3. Peripheral IDs

| Instance | ID |
|---|---|
| SPI0 | 33 |
| SPI1 | 34 |

### 46.6.4 Direct Memory Access Controller (DMAC)

The SPI interface can be used in conjunction with the DMAC in order to reduce processor overhead. For a full description of the DMAC, refer to the relevant section.

## 46.7 Functional Description

### 46.7.1 Modes of Operation

The SPI operates in Master mode or in Slave mode.

- The SPI operates in Master mode by setting the MSTR bit in the SPI Mode Register (SPI_MR):
  - Pins NPCS0 to NPCS3 are all configured as outputs
  - The SPCK pin is driven
  - The MISO line is wired on the receiver input
  - The MOSI line is driven as an output by the transmitter.
- The SPI operates in Slave mode if the MSTR bit in the SPI_MR is written to ‘0’:
  - The MISO line is driven by the transmitter output
  - The MOSI line is wired on the receiver input
  - The SPCK pin is driven by the transmitter to synchronize the receiver.
  - The NPCS0 pin becomes an input, and is used as a slave select signal (NSS)
  - NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operation. The baud rate generator is activated only in Master mode.

### 46.7.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the SPI chip select registers (SPI_CSRx). The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Consequently, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are connected and require different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

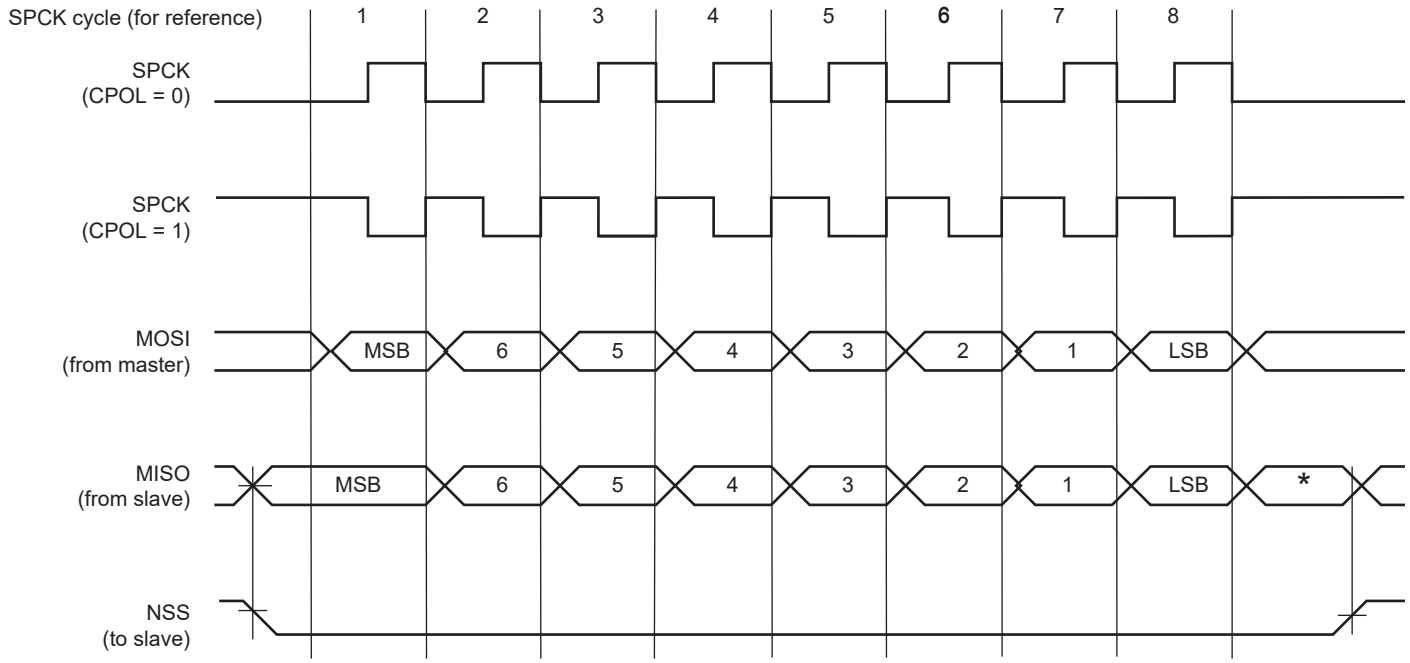
Table 46-4 shows the four modes and corresponding parameter settings.

**Table 46-4. SPI Bus Protocol Modes**

| SPI Mode | CPOL | NCPHA | Shift SPCK Edge | Capture SPCK Edge | SPCK Inactive Level |
|---|---|---|---|---|---|
| 0 | 0 | 1 | Falling | Rising | Low |
| 1 | 0 | 0 | Rising | Falling | Low |
| 2 | 1 | 1 | Rising | Falling | High |
| 3 | 1 | 0 | Falling | Rising | High |

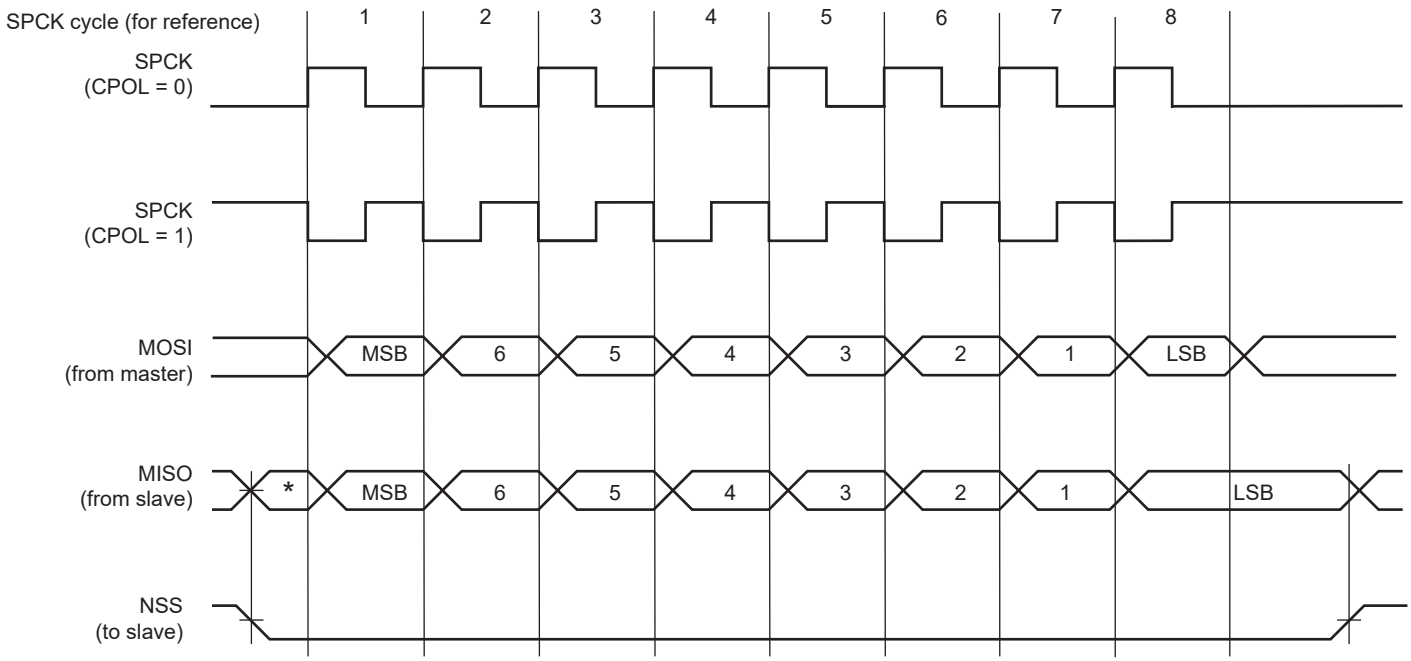
Figure 46-3 and Figure 46-4 show examples of data transfers.

**Figure 46-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**




* Not defined.

**Figure 46-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**




* Not defined.

### 46.7.3 Master Mode Operations

When configured in Master mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register (SPI_TDR) and the Receive Data Register (SPI_RDR), and a single shift register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer starts when the processor writes to the SPI_TDR. The written data is immediately transferred in the Shift register and the transfer on the SPI bus starts. While the data in the Shift register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift register. Data cannot be loaded in the SPI_RDR without transmitting data. If there is no data to transmit, dummy data can be used (SPI_TDR filled with ones). If the SPI_MR.WDRBT bit is set, transmission can occur only if the SPI_RDR has been read. If Receiving mode is not required, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the SPI Status register (SPI_SR) can be discarded.

Before writing the SPI_TDR, the PCS field in the SPI_MR must be set in order to select a slave.

If new data is written in the SPI_TDR during the transfer, it is kept in the SPI_TDR until the current transfer is completed. Then, the received data is transferred from the Shift register to the SPI_RDR, the data in the SPI_TDR is loaded in the Shift register and a new transfer starts.

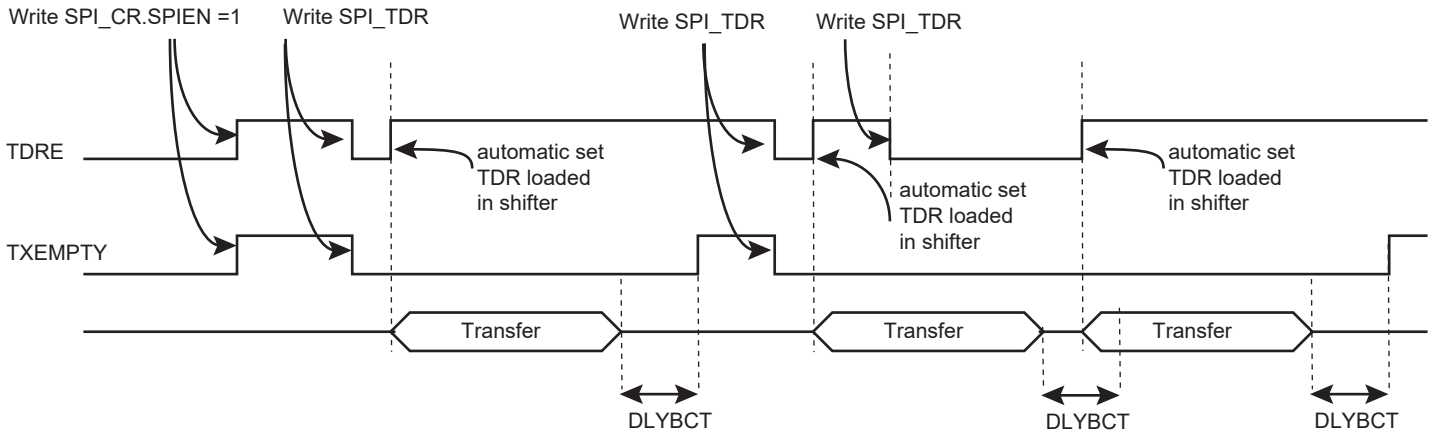
As soon as the SPI_TDR is written, the Transmit Data Register Empty (TDRE) flag in the SPI_SR is cleared. When the data written in the SPI_TDR is loaded into the Shift register, the TDRE flag in the SPI_SR is set. The TDRE bit is used to trigger the Transmit DMA channel.

See Figure 46-5.

The end of transfer is indicated by the TXEMPTY flag in the SPI_SR. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of this delay. The peripheral clock can be switched off at this time.

Note: When the SPI is enabled, the TDRE and TXEMPTY flags are set.

**Figure 46-5. TDRE and TXEMPTY flag behavior**



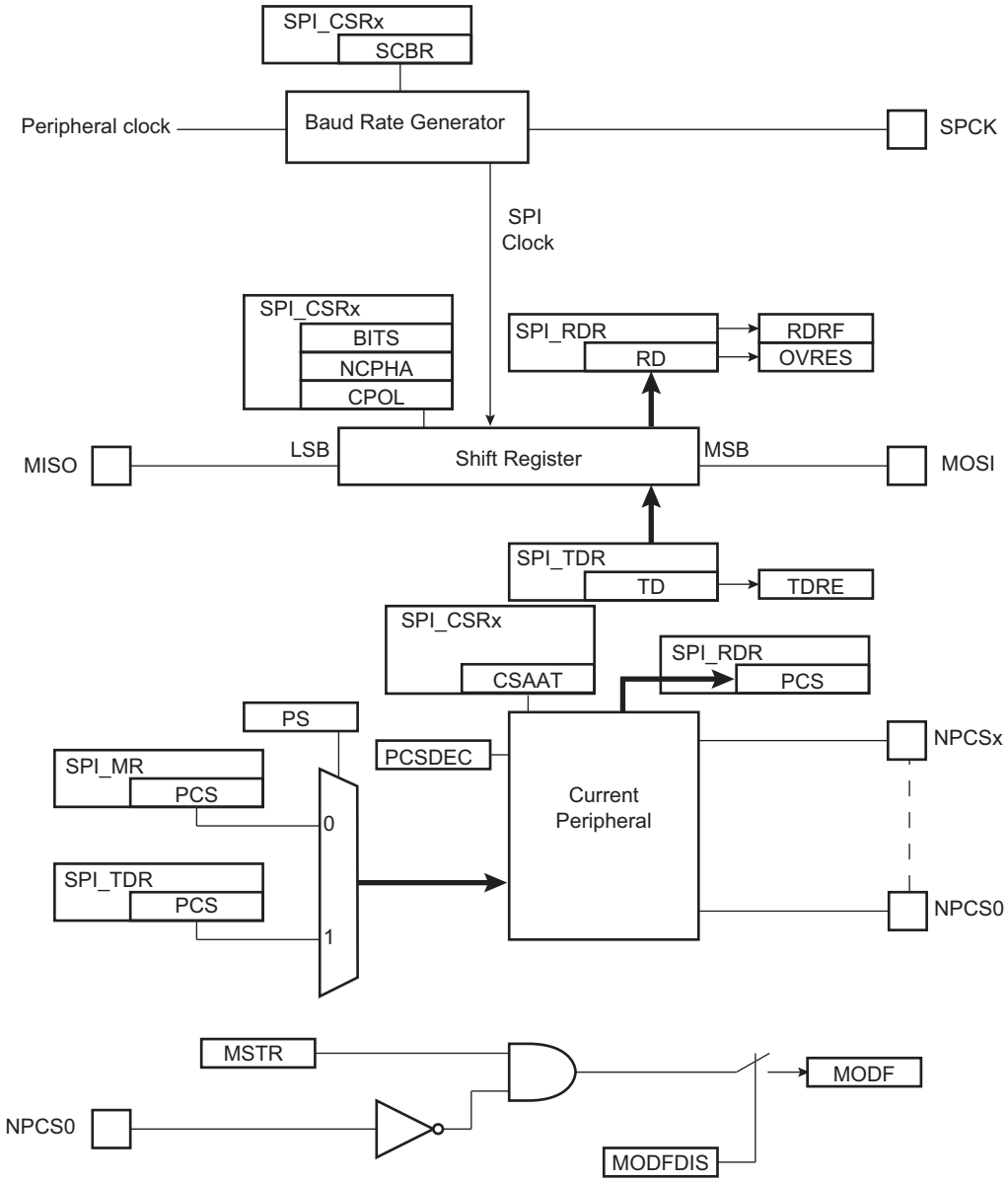

The transfer of received data from the Shift register to the SPI_RDR is indicated by the Receive Data Register Full (RDRF) bit in the SPI_SR. When the received data is read, the RDRF bit is cleared.

If the SPI_RDR has not been read before new data is received, the Overrun Error (OVRES) bit in the SPI_SR is set. As long as this flag is set, data is loaded in the SPI_RDR. The user has to read the SPI_SR to clear the OVRES bit.

Figure 46-6 shows a block diagram of the SPI when operating in Master mode. Figure 46-7 shows a flow chart describing how transfers are handled.

### 46.7.3.1 Master Mode Block Diagram

**Figure 46-6. Master Mode Block Diagram**



### 46.7.3.2 Master Mode Flow Diagram

**Figure 46-7. Master Mode Flow Diagram**

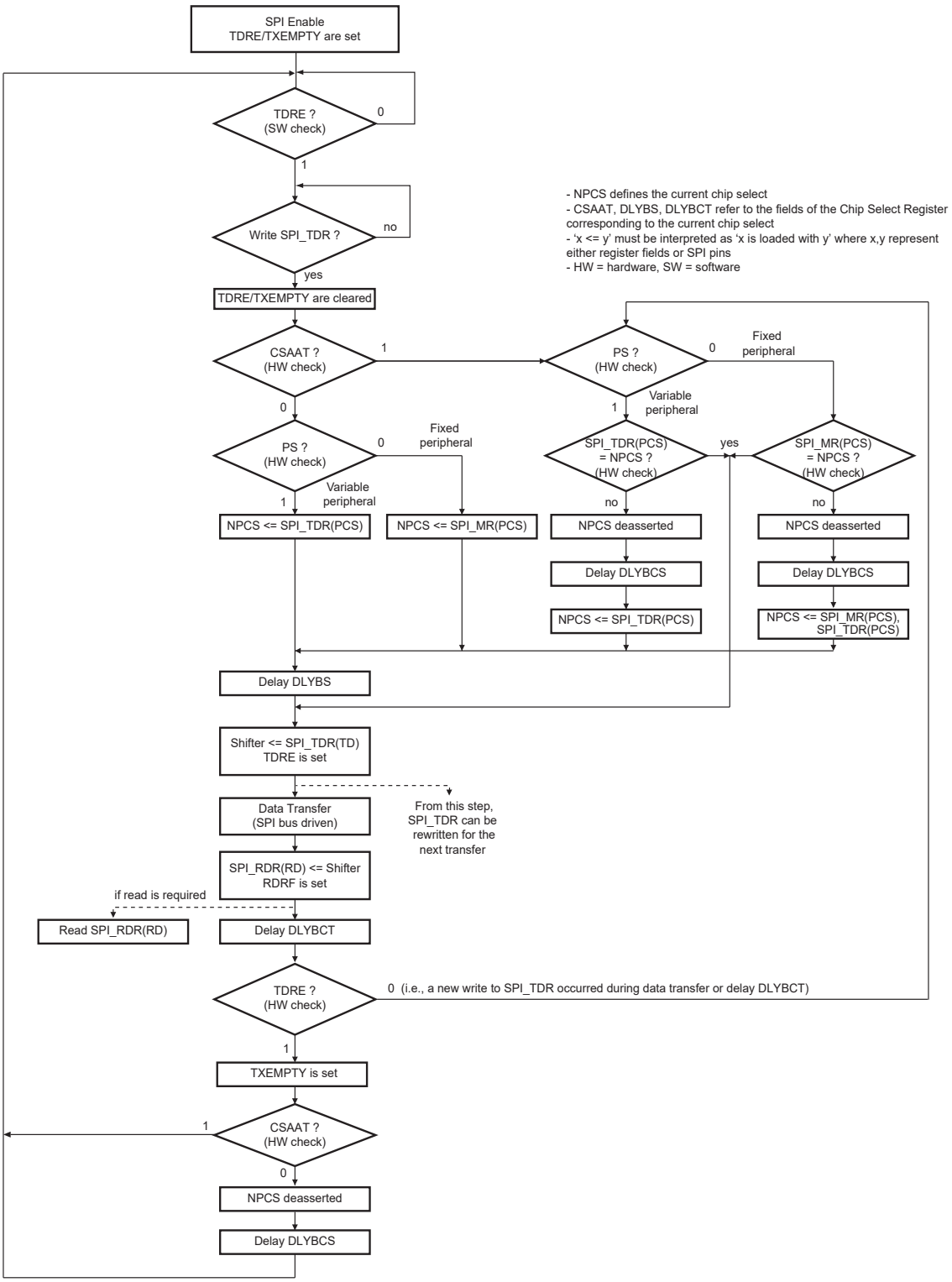
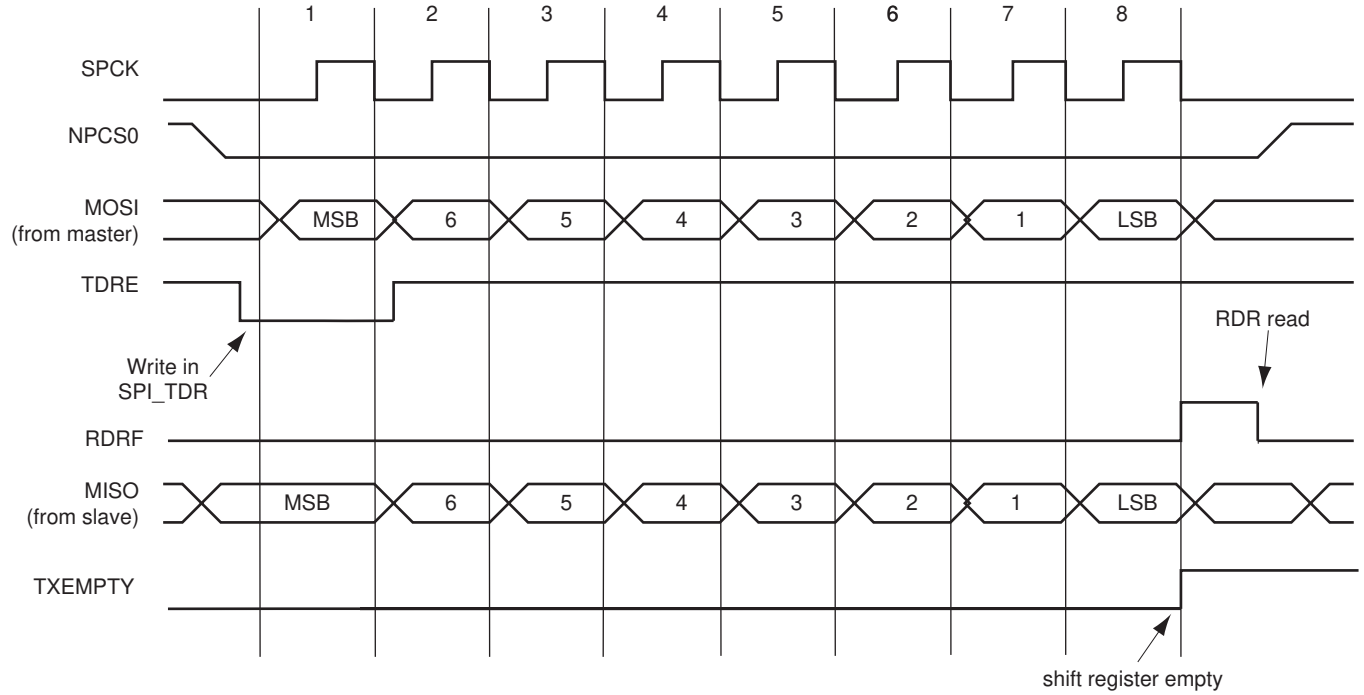


Figure 46-8 shows the behavior of Transmit Data Register Empty (TDRE), Receive Data Register (RDRF) and Transmission Register Empty (TXEMPTY) status flags within the SPI_SR during an 8-bit data transfer in Fixed mode without the DMA involved.

**Figure 46-8. Status Register Flags Behavior**




### 46.7.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the peripheral clock by a value between 1 and 255.

If the SCBR field in the SPI_CSRx is programmed to 1, the operating baud rate is peripheral clock (see the electrical characteristics section for the SPCK maximum frequency). Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it to a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

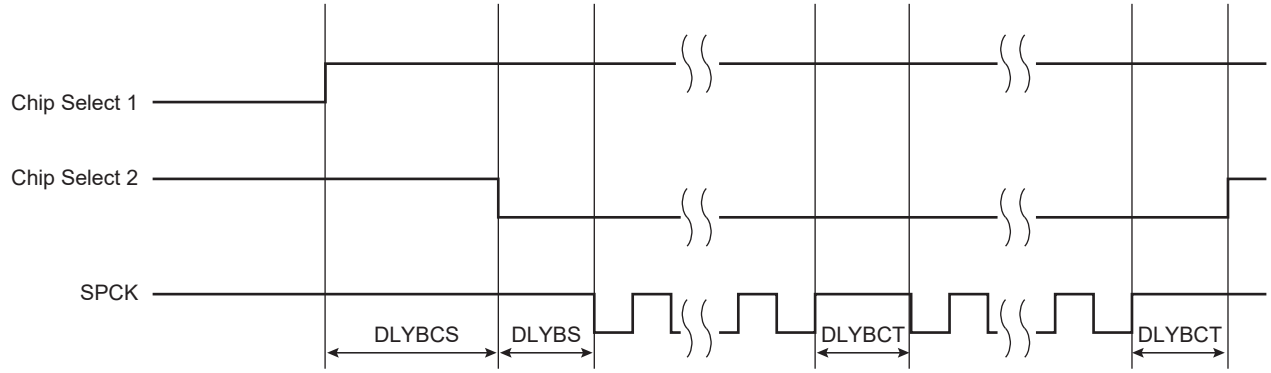
### 46.7.3.4 Transfer Delays

Figure 46-9 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

- Delay between the chip selects—programmable only once for all chip selects by writing the DLYBCS field in the SPI_MR. The SPI slave device deactivation delay is managed through DLYBCS. If there is only one SPI slave device connected to the master, the DLYBCS field does not need to be configured. If several slave devices are connected to a master, DLYBCS must be configured depending on the highest deactivation delay. Refer to the SPI slave device electrical characteristics.
- Delay before SPCK—independently programmable for each chip select by writing the DLYBS field. The SPI slave device activation delay is managed through DLYBS. Refer to the SPI slave device electrical characteristics to define DLYBS.
- Delay between consecutive transfers—independently programmable for each chip select by writing the DLYBCT field. The time required by the SPI slave device to process received data is managed through DLYBCT. This time depends on the SPI slave system activity.

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

**Figure 46-9. Programmable Delays**




#### 46.7.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all NPCS signals are high before and after each transfer.

- **Fixed Peripheral Select Mode**: SPI exchanges data with only one peripheral.
  Fixed Peripheral Select mode is enabled by clearing the PS bit in the SPI_MR. In this case, the current peripheral is defined by the PCS field in the SPI_MR and the PCS field in the SPI_TDR has no effect.
- **Variable Peripheral Select Mode:** Data can be exchanged with more than one peripheral without having to reprogram the NPCS field in the SPI_MR.
  Variable Peripheral Select mode is enabled by setting the PS bit in the SPI_MR. The PCS field in the SPI_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data. The value to write in the SPI_TDR has the following format:

  [xxxxxxx(7-bit) + LASTXFER(1-bit)(1)+ xxxx(4-bit) + PCS (4-bit) + DATA (8 to 16-bit)] with PCS equals the chip select to assert, as defined in Section 46.8.6 “SPI Transmit Data Register” and LASTXFER bit at 0 or 1 depending on the CSAAT bit.

Note: 1. Optional

CSAAT, LASTXFER and CSNAAT bits are discussed in Section 46.7.3.9 “Peripheral Deselection with DMA”.

If LASTXFER is used, the command must be issued after writing the last character. Instead of LASTXFER, the user can use the SPIDIS command. After the end of the DMA transfer, it is necessary to wait for the TXEMPTY flag and then write SPIDIS into the SPI Control Register (SPI_CR). This does not change the configuration register values). The NPCS is disabled after the last character transfer. Then, another DMA transfer can be started if the SPIEN has previously been written in the SPI_CR.

#### 46.7.3.6 SPI Direct Access Memory Controller (DMAC)

In both Fixed and Variable modes, the Direct Memory Access Controller (DMAC) can be used to reduce processor overhead.

The fixed peripheral selection allows buffer transfers with a single peripheral. Using the DMAC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, if the peripheral selection is modified, the SPI_MR must be reprogrammed.

The variable peripheral selection allows buffer transfers with multiple peripherals without reprogramming the SPI_MR. Data written in the SPI_TDR is 32 bits wide and defines the real data to be transmitted and the destination peripheral. Using the DMAC in this mode requires 32-bit wide buffers, with the data in the LSBs and the

PCS and LASTXFER fields in the MSBs. However, the SPI still controls the number of bits (8 to 16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in terms of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

#### 46.7.3.7 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 slave peripherals by decoding the four chip select lines, NPCS0 to NPCS3 with an external decoder/demultiplexer (refer to Figure 46-10). This can be enabled by setting the PCSDEC bit in the SPI_MR.

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e., one NPCS line driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

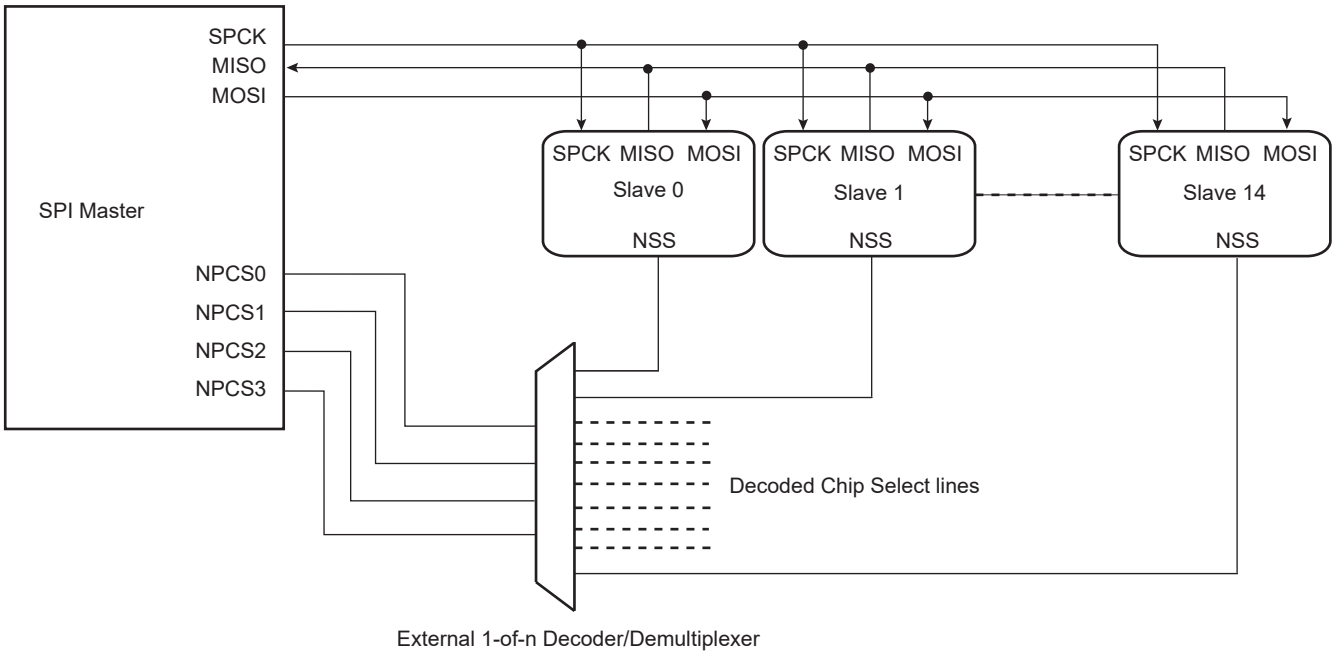
When operating with decoding, the SPI directly outputs the value defined by the PCS field on the NPCS lines of either SPI_MR or SPI_TDR (depending on PS).

As the SPI sets a default value of 0xF on the chip select lines (i.e., all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has four chip select registers (SPI_CSR0...SPI_CSR3). As a result, when external decoding is activated, each NPCS chip select defines the characteristics of up to four peripherals. As an example, SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 3, corresponding to the PCS values 0x0 to 0x3. Consequently, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14. Figure 46-10 shows this type of implementation.

If the CSAAT bit is used, with or without the DMAC, the Mode Fault detection for NPCS0 line must be disabled. This is not needed for all other chip select lines since Mode Fault detection is only on NPCS0.

**Figure 46-10. Chip Select Decoding Application Block Diagram: Single Master/Multiple Slave Implementation**



#### 46.7.3.8 Peripheral Deselection without DMA

During a transfer of more than one unit of data on a chip select without the DMA, the SPI_TDR is loaded by the processor, the TDRE flag rises as soon as the content of the SPI_TDR is transferred into the internal Shift register. When this flag is detected high, the SPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the chip select is not deasserted between the two transfers. But depending on the application software handling the SPI status register flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the SPI_TDR in time to keep the chip select active (low). A null DLYBCT value (delay between consecutive transfers) in the SPI_CSR, gives even less time for the processor to reload the SPI_TDR. With some SPI slave peripherals, if the chip select line must remain active (low) during a full set of transfers, communication errors can occur.

To facilitate interfacing with such devices, the chip select registers [SPI_CSR0...SPI_CSR3] can be programmed with the Chip Select Active After Transfer (CSAAT) bit at 1. This allows the chip select lines to remain in their current state (low = active) until a transfer to another chip select is required. Even if the SPI_TDR is not reloaded, the chip select remains active. To deassert the chip select line at the end of the transfer, the Last Transfer (LASTXFER) bit in SPI_CR must be set after writing the last data to transmit into SPI_TDR.

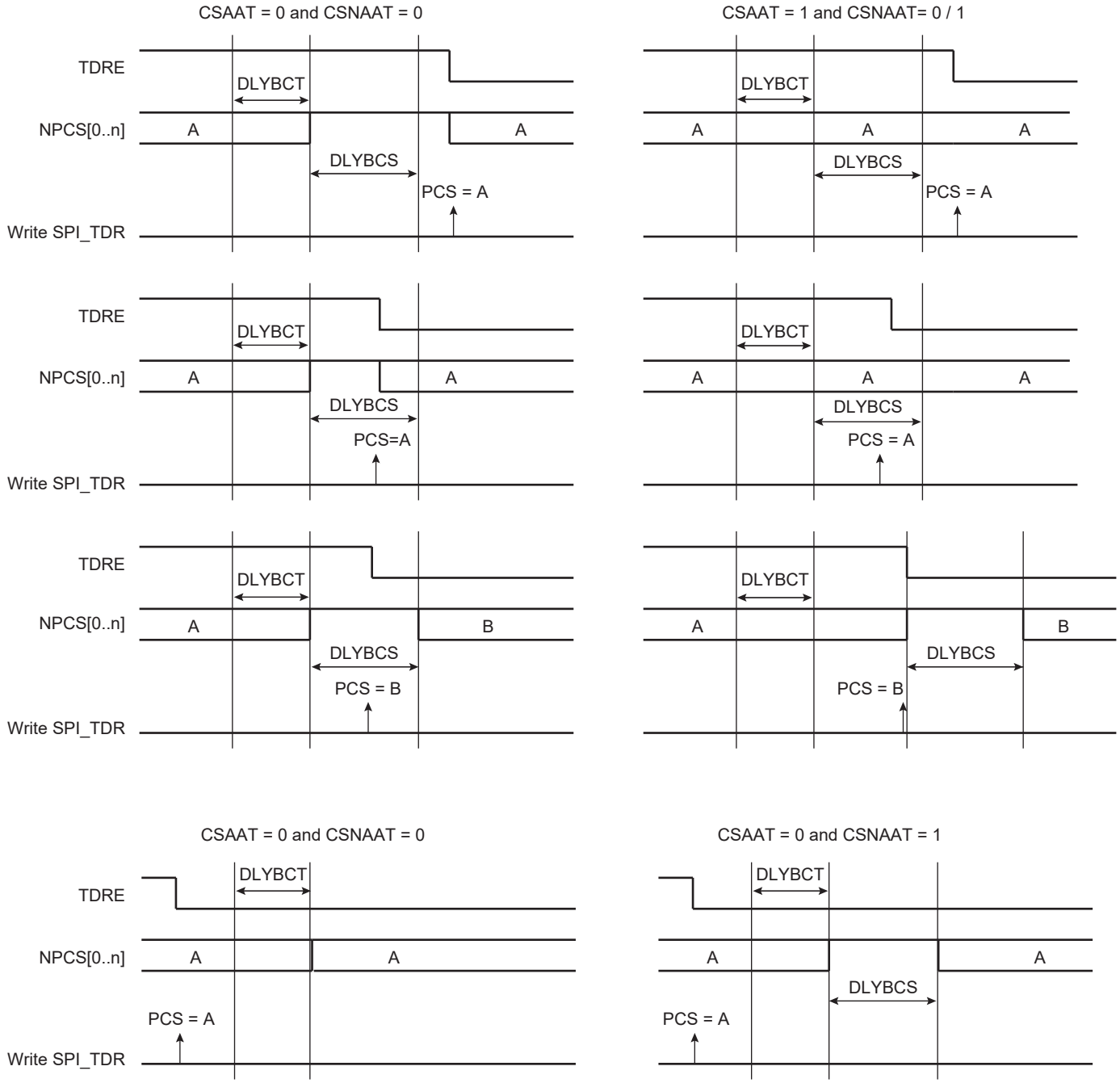
#### 46.7.3.9 Peripheral Deselection with DMA

DMA provides faster reloads of the SPI_TDR compared to software. However, depending on the system activity, it is not guaranteed that the SPI_TDR is written with the next data before the end of the current transfer. Consequently, data can be lost by the deassertion of the NPCS line for SPI slave peripherals requiring the chip select line to remain active between two transfers. The only way to guarantee a safe transfer in this case is the use of the CSAAT and LASTXFER bits.

When the CSAAT bit is configured to 0, the NPCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a chip select, the TDRE flag rises as soon as the content of the SPI_TDR is transferred into the internal shift register. When this flag is detected, the SPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the chip select is not deasserted between the two transfers. This can lead to difficulties to interface with some serial peripherals requiring the chip select to be deasserted after each transfer. To facilitate interfacing with such devices, the SPI_CSR can be programmed with the Chip Select Not Active After Transfer (CSNAAT) bit at 1. This allows the chip select lines to be deasserted systematically during a time “DLYBCS” (the value of the CSNAAT bit is processed only if the CSAAT bit is configured to 0 for the same chip select).

Figure 46-11 shows different peripheral deselection cases and the effect of the CSAAT and CSNAAT bits.

**Figure 46-11. Peripheral Deselection**




### 46.7.3.10 Mode Fault Detection

The SPI has the capability to operate in multimaster environment. Consequently, the NPCS0/NSS line must be monitored. If one of the masters on the SPI bus is currently transmitting, the NPCS0/NSS line is low and the SPI must not transmit any data. A mode fault is detected when the SPI is programmed in Master mode and a low level is driven by an external master on the NPCS0/NSS signal. In multimaster environment, NPCS0, MOSI, MISO and SPCK pins must be configured in open drain (through the PIO controller). When a mode fault is detected, the SPI_SR.MODF bit is set until SPI_SR is read and the SPI is automatically disabled until it is reenabled by setting the SPI_CR.SPIEN bit.

By default, the mode fault detection is enabled. The user can disable it by setting the SPI_MR.MODFDIS bit.

### 46.7.4 SPI Slave Mode

When operating in Slave mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits until NSS goes active before receiving the serial clock from an external master. When NSS falls, the clock is validated and the data is loaded in the SPI_RDR depending on the BITS field configured in SPI_CSR0. These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits in SPI_CSR0. Note that the fields BITS, CPOL and NCPHA of the other chip select registers (SPI_CSR1...SPI_CSR3) have no effect when the SPI is programmed in Slave mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

Note: For more information on the BITS field, see also the note below the SPI_CSRx bitmap (Section 46.8.12 “SPI Chip Select Register”).

When all bits are processed, the received data is transferred in the SPI_RDR and the RDRF bit rises. If the SPI_RDR has not been read before new data is received, the Overrun Error Status (OVRES) bit in the SPI_SR is set. As long as this flag is set, data is loaded in the SPI_RDR. The user must read SPI_SR to clear the OVRES bit.

When a transfer starts, the data shifted out is the data present in the Shift register. If no data has been written in the SPI_TDR, the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift register resets to 0.

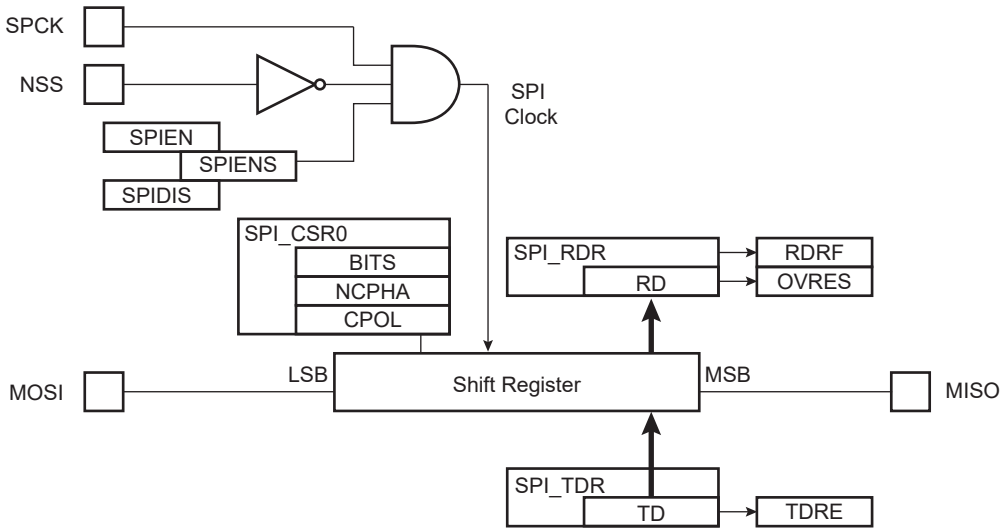
When a first data is written in the SPI_TDR, it is transferred immediately in the Shift register and the TDRE flag rises. If new data is written, it remains in the SPI_TDR until a transfer occurs, i.e., NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in the SPI_TDR is transferred in the Shift register and the TDRE flag rises. This enables frequent updates of critical variables with single transfers.

Then, new data is loaded in the Shift register from the SPI_TDR. If no character is ready to be transmitted, i.e., no character has been written in the SPI_TDR since the last load from the SPI_TDR to the Shift register, the SPI_TDR is retransmitted. In this case the Underrun Error Status Flag (UNDES) is set in the SPI_SR.

If NSS rises between two characters, it must be kept high for two MCK clock periods or more and the next SPCK capture edge must not occur less than four MCK periods after NSS rise.

Figure 46-12 shows a block diagram of the SPI when operating in Slave mode.

**Figure 46-12. Slave Mode Functional Block Diagram**



### 46.7.5 SPI Comparison Function on Received Character

The comparison is on ly relevant for SPI Slave mode (MSTR = 0 in SPI_MR).

The effect of a comparison match changes if the system is in Wait or Active mode.

In Wait mode, if asynchronous partial wakeup is enabled, a system wakeup is performed (see Section 46.7.6).

In Active mode, the CMP flag in SPI_SR is raised. It is set when the received character matches the conditions programmed in the SPI Comparison Register (SPI_CMPR). The CMP flag is set as soon as SPI_RDR is loaded with the new received character. The CMP flag is cleared by reading SPI_SR.

SPI_CMPR (see Section 46.8.15) can be programmed to provide different comparison methods. These are listed below:

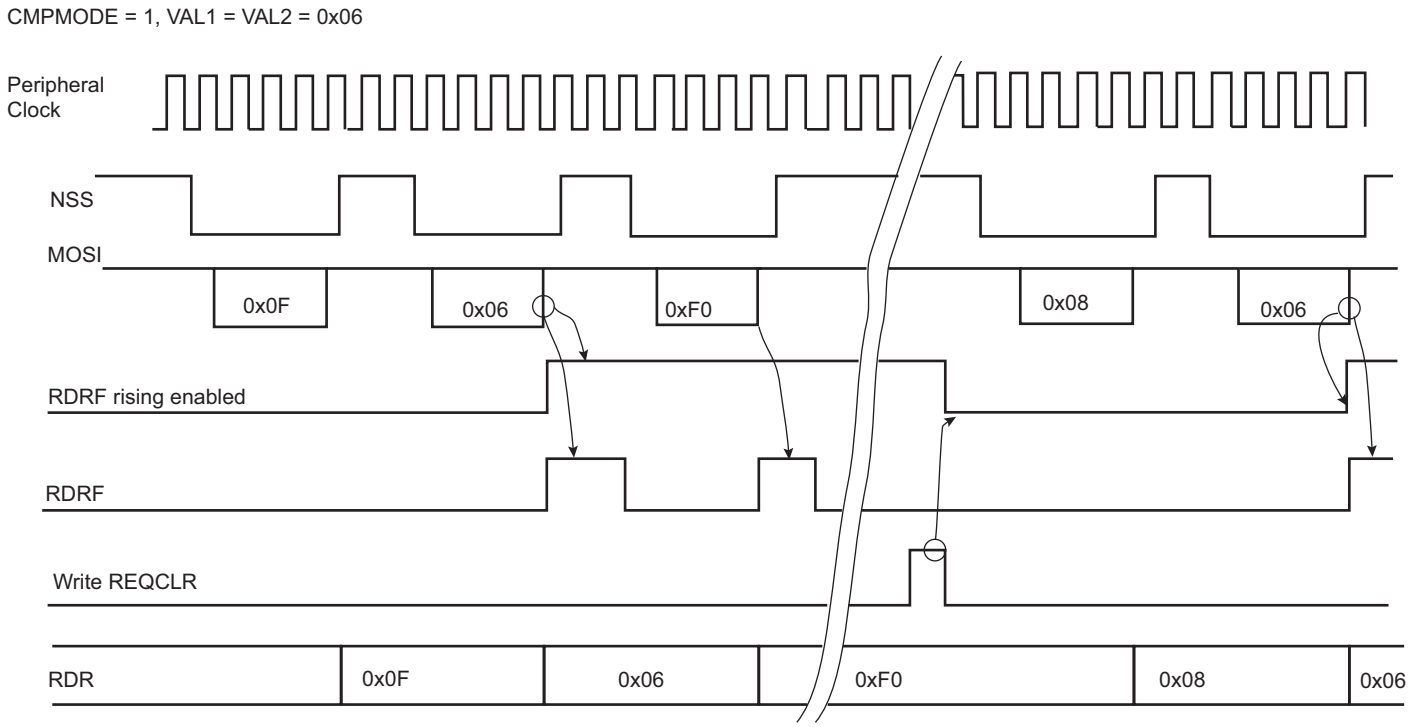
- If VAL1 equals VAL2, then the comparison is performed on a single value and the flag is set to 1 if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 sets the CMP flag.
- If VAL1 is strictly higher than VAL2, then the flag CMP is set to 1 if any received character equals VAL1 or VAL2.

When the CMPMODE bit is cleared in SPI_CMPR, all received data is loaded in SPI_RDR and the CMP flag provides the status of the comparison result.

By setting the CMPMODE bit, the comparison result triggers the start of the SPI_RDR loading (see Figure 46-13). The trigger condition exists as soon as the received character value matches the conditions defined by VAL1 and VAL2 in SPI_CMPR. The comparison trigger event is restarted by setting the REQCLR bit in SPI_CR if SleepWalking is disabled.

The value programmed in VAL1 and VAL2 fields must not exceed the maximum value of the received character (see BITS field in SPI_CSR0).

**Figure 46-13. Receive Data Register Management**



### 46.7.6 SPI Asynchronous and Partial Wakeup (SleepWalking)

This operating mode is a means of data preprocessing that qualifies an incoming event, thus allowing the SPI to decide whether or not to wake up the system. Asynchronous and partial wakeup is mainly used when the system is in Wait mode (see the PMC section for further details). It can also be enabled when the system is fully running.

Asynchronous and partial wakeup can be used only when SPI is configured in Slave mode (MSTR is cleared in SPI_MR).

The maximum SPI clock (SPCK) frequency that can be provided by the SPI master is bounded by the peripheral clock frequency. The SPCK frequency must be lower than or equal to the peripheral clock. The NSS line must be deasserted by the SPI master between two characters. The NSS deassertion duration time must be greater than or equal to six peripheral clock periods. The time between the assertion of NSS line (falling edge) and the first edge of the SPI clock must be higher than 5 µs.

The SPI_RDR must be read before enabling the asynchronous and partial wakeup.

When asynchronous and partial wakeup is enabled for the SPI (see the PMC section), the PMC decodes a clock request from the SPI. The request is generated as soon as there is a falling edge on the NSS line as this may indicate the beginning of a frame. If the system is in Wait mode (processor and peripheral clocks switched off), the PMC restarts the fast RC oscillator and provides the clock only to the SPI.

The SPI processes the received frame and compares the received character with VAL1 and VAL2 in SPI_CMPR (Section 46.8.15).

The SPI instructs the PMC to disable the peripheral clock if the received character value does not meet the conditions defined by VAL1 and VAL2 fields in SPI_CMPR (see Figure 46-15).

If the received character value meets the conditions, the SPI instructs the PMC to exit the system from Wait mode (see Figure 46-14).

The VAL1 and VAL2 fields can be programmed to provide different comparison methods and thus matching conditions.

- If VAL1 = VAL2, then the comparison is performed on a single value and the wakeup is triggered if the received character equals VAL1.
- If VAL1 is strictly lower than VAL2, then any value between VAL1 and VAL2 wakes up the system.
- If VAL1 is strictly higher than VAL2, the wakeup is triggered if any received character equals VAL1 or VAL2.
- If VAL1 = 0 and VAL2 = 65535, the wakeup is triggered as soon as a character is received.

If the processor and peripherals are running, the SPI can be configured in Asynchronous and Partial Wakeup mode by enabling the PMC_SLPWK_ER (see PMC section). When activity is detected on the receive line, the SPI requests the clock from the PMC and the comparison is performed. If there is a comparison match, the SPI continues to request the clock. If there is no match, the clock is switched off for the SPI only, until a new activity is detected.

The CMPMODE configuration has no effect when Asynchronous and Partial Wakeup mode is enabled for the SPI (see PMC_SLPWK_ER in PMC section).

When the system is in Active mode and the SPI enters Asynchronous and Partial Wakeup mode, the flag RDRF must be programmed as the unique source of the SPI interrupt.

When the system exits Wait mode as the result of a matching condition, the RDRF flag is used to determine if the SPI is the source for the exit from Wait mode.

**Figure 46-14. Asynchronous Wakeup Use Case Example**

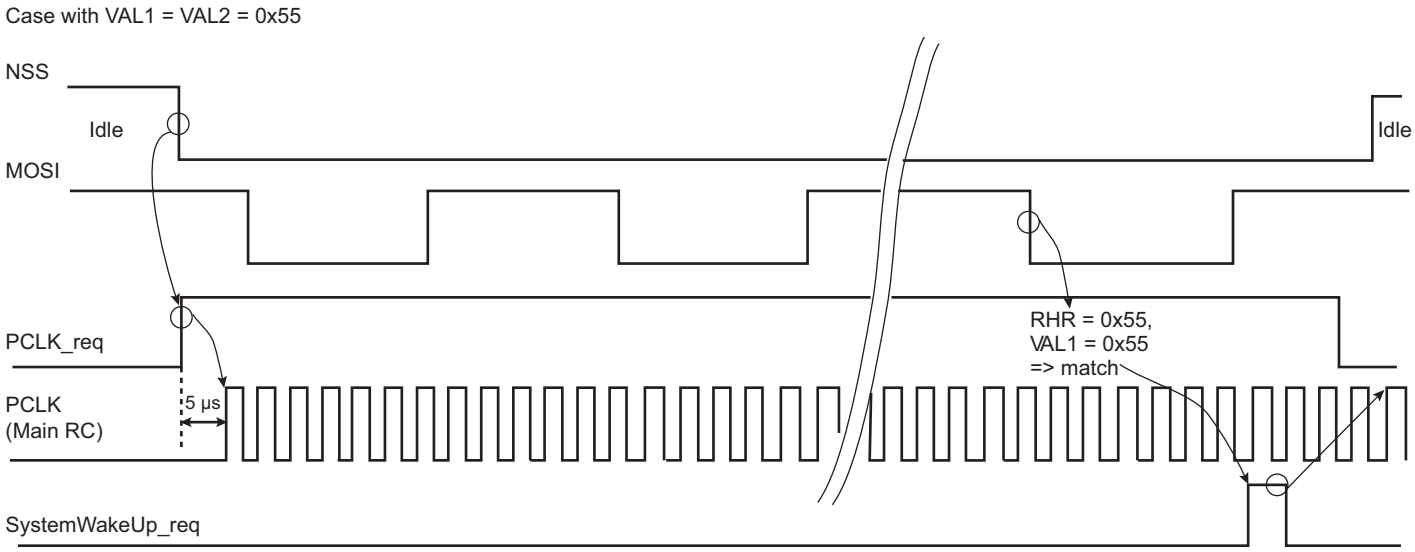
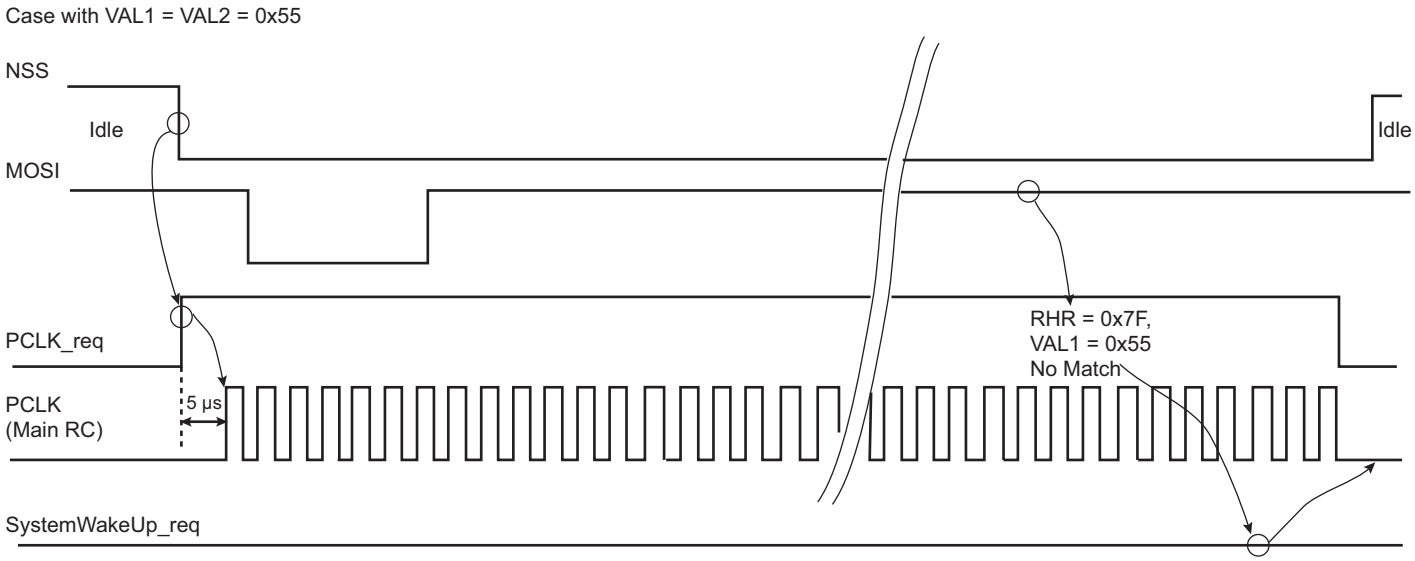



**Figure 46-15. Asynchronous Event Generating Only Partial Wakeup**

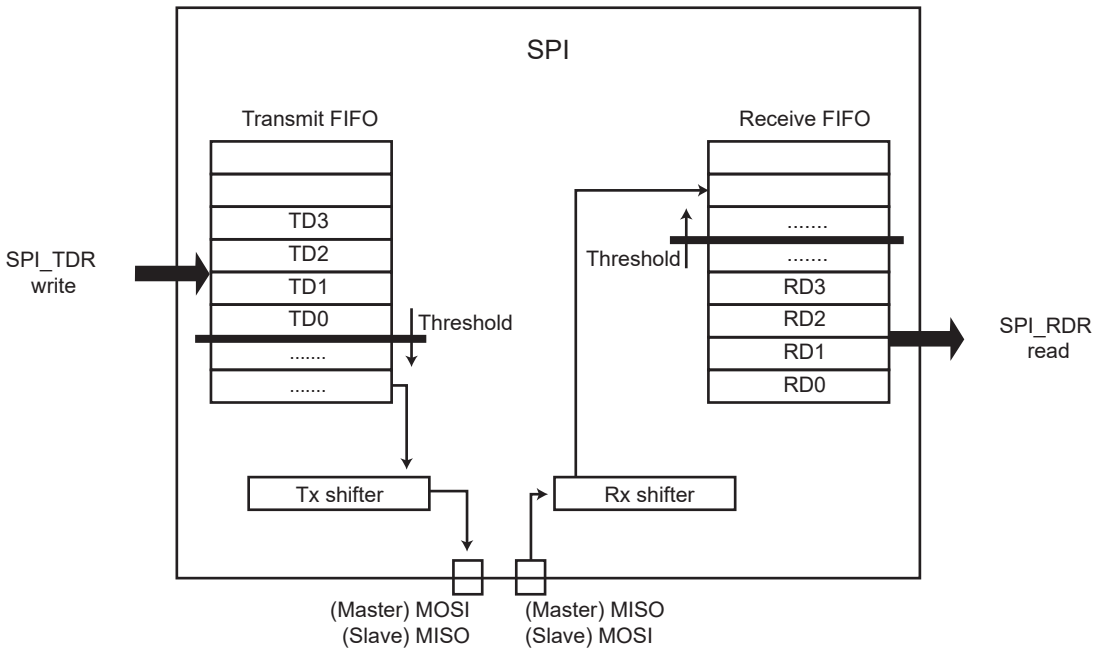


### 46.7.7 FIFOs

The SPI includes two FIFOs which can be enabled/disabled using the FIFOEN/FIFODIS bits in the SPI_CR. It is recommended to disable the SPI module before enabling or disabling the SPI FIFOs (TXDIS and RXDIS bit in SPI_CR).

Writing the FIFOEN bit to ‘1’ enables a 16-data Transmit FIFO and a 16-data Receive FIFO.

**Figure 46-16. FIFOs Block Diagram**

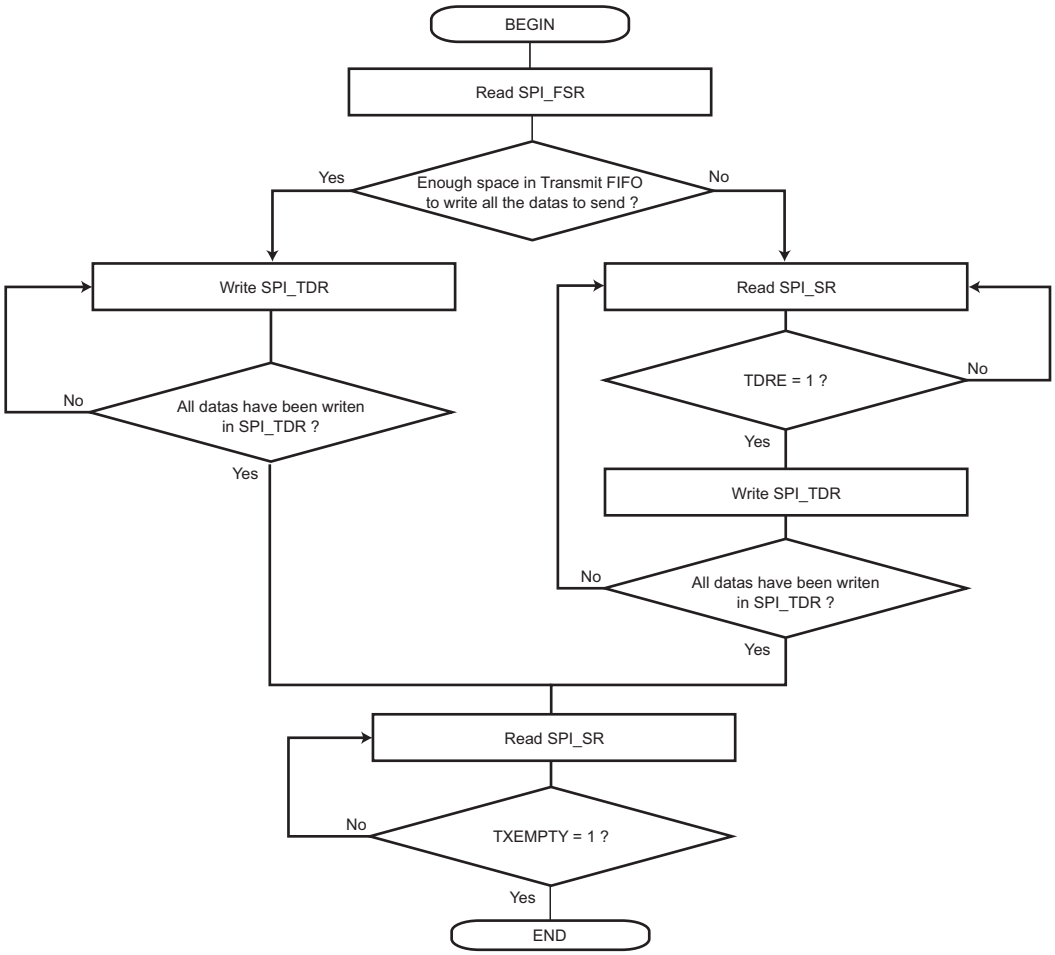


#### 46.7.7.1 Sending Data with FIFO Enabled

With the Transmit FIFO enabled, any write access to the SPI_TDR brings the written data to the Transmit FIFO. As a consequence, it is not mandatory any more to monitor the TDRE flag state to send multiple data without DMAC.

Knowing the number of data to send, and provided there is enough space in the Transmit FIFO, all the data to send can be written successively in the SPI_TDR without checking the TDRE flag between each access. The Transmit FIFO state can be checked reading the TXFL field in the SPI FIFO Level Register (SPI_FLR).

**Figure 46-17. Sending Data with FIFO Flowchart**

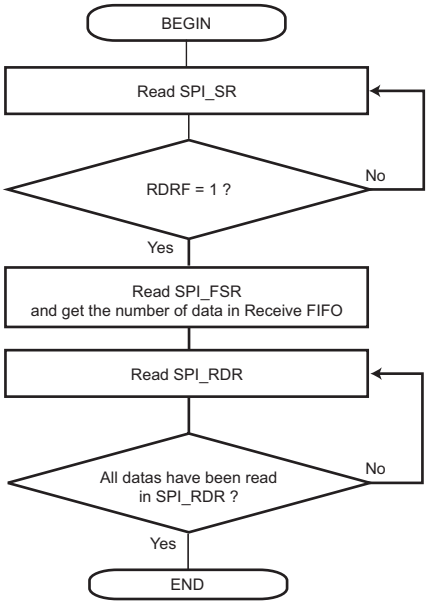


### 46.7.7.2 Receiving Data with FIFO Enabled

With Receive FIFO enabled, any read access on SPI_RDR pulls out a data from the Receive FIFO. As a consequence, it is not mandatory any more to monitor the RDRF flag when DMAC is not used and there are multiple data to read.

When data are present in the Receive FIFO (RDRF flag set to ‘1’), the exact number of data can be checked with the RXFL field in the SPI_FLR and all the data read successively in the SPI_RDR without checking the RDRF flag between each access.

**Figure 46-18. Receiving Data with FIFO Flowchart**




### 46.7.7.3 Clearing/Flushing FIFOs

Each FIFO can be cleared/flushed using the TXFCLR and RXFCLR bits in the SPI_CR.

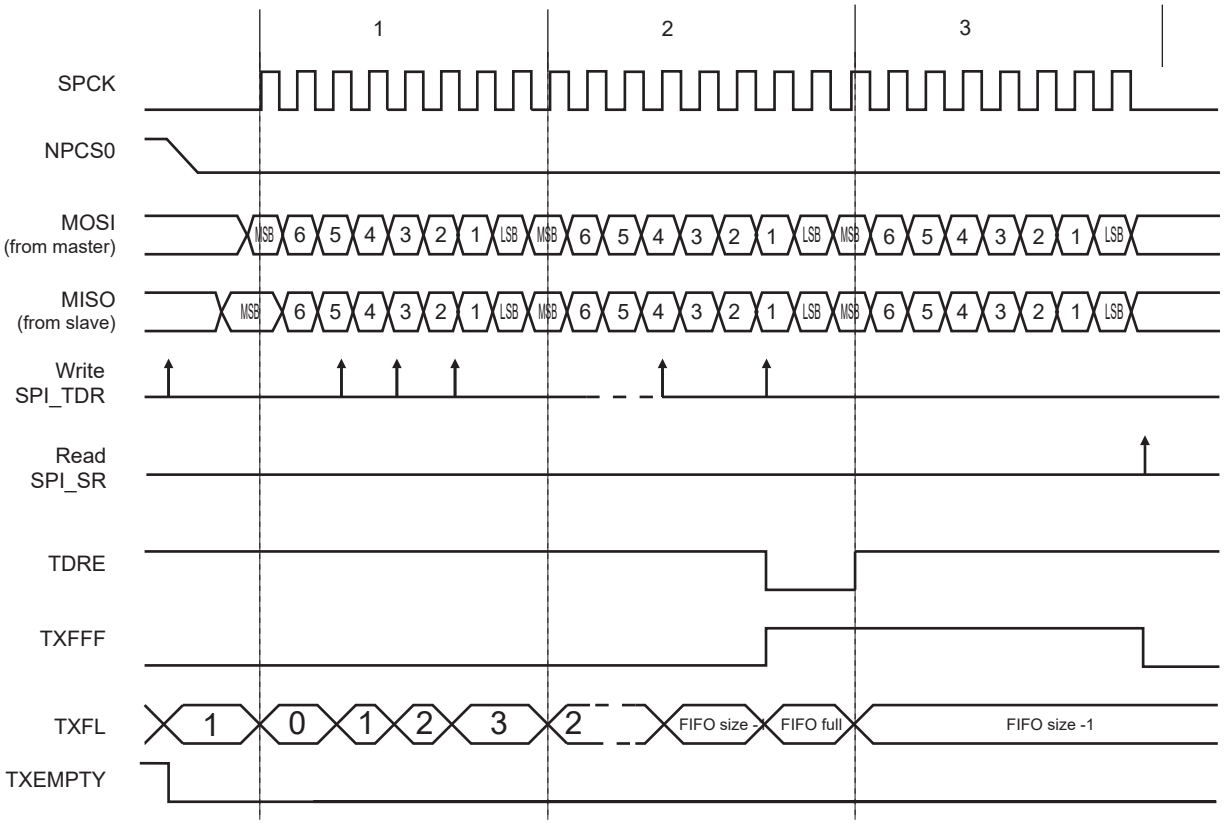
### 46.7.7.4 TDRE, RDRF and TXEMPTY Behavior

If FIFOs are enabled, the behavior of the TDRE, RDRF and TXEMPTY flags is slightly different.

With FIFO enabled, the TXEMPTY flag remains at ‘0’ state as long as there are characters in the Transmit FIFO or in the Transmit Shift Register. It is at ‘1’ state when there are no character in the Transmit FIFO and no character in the Transmit Shift Register.

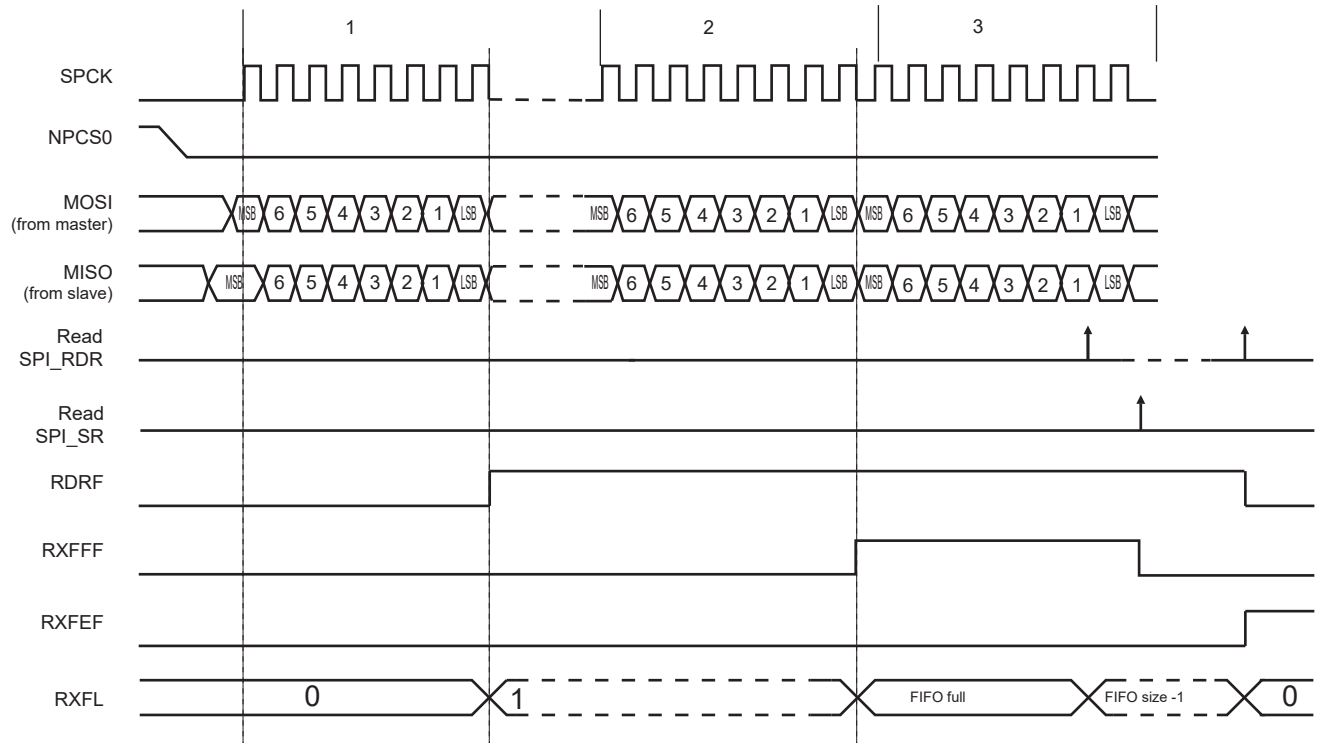
TDRE indicates if a data can be written in the Transmit FIFO. By default, the TDRE flag then stays at level ‘1’ as long as the Transmit FIFO is not full (TXRDYM = 0x0).

**Figure 46-19. TDRE in Single Data Mode and TXRDYM = 0x0**




RDRF indicates if an unread data is present in the Receive FIFO. By default, the RDRF flag is then at level ‘1’ as soon as one unread data is in the Receive FIFO (RXRDYM = 0x0).

**Figure 46-20. RDRF in Single Data Mode and RXRDYM = 0x0**




TDRE and RDRF behavior can be modified using the TXRDYM and RXRDYM fields in the SPI FIFO Mode Register (SPI_FMR). In Single Data mode, there is no need to modify the TDRE and RDRF behavior; however, it may be useful in Multiple Data mode.

### 46.7.7.5 Single Data Mode

If Variable Peripheral Select mode is used (PS bit set to '1' in SPI_MR), the Transmit FIFO operates in Single Data mode.

If Master mode is set (MSTR bit set to '1' in SPI_MR), the Receive FIFO operates in Single Data mode.

In this mode, only one data can be written/read per write/read access of SPI_TDR/SPI_RDR (see Section 46.8.3 "SPI Receive Data Register"). On the other hand, TXRDYM and RXRDYM fields should be configured with 0x0 value in this mode.

*DMAC*

TXRDYM and RXRDYM fields must be configured with 0x0 value when working with DMAC transfer in Single Data mode.

The same DMAC procedure applies as with FIFO disabled.

### 46.7.7.6 Multiple Data Mode

When the FIFOs do not operate in Single Data mode, they operate in Multiple Data mode.

In Multiple Data mode, it is possible to write up to two data in one SPI_TDR write access. It is possible, in this mode, to read up to four data in one SPI_RDR access if the BITS field is configured to 0 (8-bit data size) and up to two data if the BITS field value is other than 0 (more than 8-bit data size).

The number of data to write/read is defined by the size of the register access. If the access is a byte-size register access, only one data is written/read. If the access is a halfword size register access, then up to two data are read/only one data is written and finally, if the access is a word-size register access, then up to four data are read/up to two data are written.

Written/Read data are always right-aligned, as shown in Section 46.8.4 “SPI Receive Data Register (FIFO Multiple Data, 8-bit)”, Section 46.8.5 “SPI Receive Data Register (FIFO Multiple Data, 16-bit)” and Section 46.8.7 “SPI Transmit Data Register (FIFO Multiple Data, 8- to 16-bit)”.

For instance, if the Transmit FIFO is empty and there are six data to send, you can either:

- Perform six SPI_TDR-byte write accesses.
- Perform three SPI_TDR-halfword write accesses.

It goes the same with a Receive FIFO containing six data where you can either:

- Perform six SPI_RDR-byte read accesses.
- Perform three SPI_RDR-halfword read accesses.
- Perform one SPI_RDR-word read access and one SPI_RDR-halfword read access.

This mode allows to minimize the number of accesses by concatenating the data to send/read in one access.

*TDRE and RDRF Configuration*

In Multiple Data mode, the TXRDYM and RXRDYM fields in the SPI_FMR become useful.

As in Multiple Data mode, it is possible to write several data in the same access it might be useful to configure TDRE flag behavior to indicate if one or two data can be written in the FIFO depending on the access to perform on SPI_TDR.

If for instance two data are written each time in the SPI_TDR, it might be useful to configure the TXRDYM field to 0x1 value so that the TDRE flag is at ‘1’ only when at least two data can be written in the Transmit FIFO.

In the same way, if four data are read each time in the SPI_RDR, it might be useful to configure the RXRDYM field to 0x2 value so that the RDRF flag is at ‘1’ only when at least four unread data are in the Receive FIFO.

*DMAC*

If DMAC transfer is used, it is mandatory to configure TXRDYM/RXRDYM to the right value depending on the DMAC channel size (byte, halfword or word).

#### 46.7.7.7 FIFO Pointer Error

In some specific cases, it is possible to generate a FIFO pointer error.

- Transmit FIFO:

If the Transmit FIFO is full and a write access is performed on the SPI_TDR, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in SPI_SR.

In Multiple Data mode, if the number of data written in the SPI_TDR (according to the register access size) is bigger than the Transmit FIFO free space, it generates a Transmit FIFO pointer error and sets the TXFPTEF flag in SPI_SR.

- Receive FIFO:

In Multiple Data mode, if the number of data read in the SPI_RDR (according to the register access size) is bigger than the number of unread data in the Receive FIFO, it generates a Receive FIFO pointer error and sets the RXFPTEF flag in SPI_SR.

No pointer error should happen if the FIFO state is checked before writing/reading in SPI_TDR/SPI_RDR. The FIFO state can be checked either with TXRDY, RXRDY, TXFL or RXFL. When a pointer error occurs, other FIFO flags might not behave as expected; their state should be ignored.

If a pointer error occurs, a software reset must be performed through the SWRST bit in SPI_CR (configuration will be lost).

#### 46.7.7.8 FIFO Thresholds

Each Transmit and Receive FIFO includes a threshold feature used to set a flag and an interrupt when a FIFO threshold is crossed. Thresholds are defined as a number of data in the FIFO, and the FIFO state (TXFL or RXFL) represents the number of data currently in the FIFO.

- Transmit FIFO:

The Transmit FIFO threshold can be set using the TXFTHRES field in SPI_FMR. Each time the Transmit FIFO goes from the ‘above threshold’ to the ‘equal or below threshold’ state, the TXFTHF flag in SPI_SR is set. This enables the application to know that the Transmit FIFO reached the defined threshold and to refill it before it becomes empty.

- Receive FIFO:

The Receive FIFO threshold can be set using the RXFTHRES field in SPI_FMR. Each time the Receive FIFO goes from the ‘below threshold’ to the ‘equal or above threshold’ state, the RXFTHF flag in SPI_SR is set. This enables the application to know that the Receive FIFO reached the defined threshold and to read some data before it becomes full.

The TXFTHF and RXFTHF flags can be both configured to generate an interrupt using SPI_IER and SPI_IDR.

#### 46.7.7.9 FIFO Flags

FIFOs come with a set of flags which can be configured each to generate interrupt through SPI_IER and SPI_IDR.

FIFO flags state can be read in SPI_SR. They are cleared on SPI_SR read.

### 46.7.8 Register Write Protection

To prevent any single software error from corrupting SPI behavior, certain registers in the address space can be write-protected in the SPI Write Protection Mode Register (SPI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the SPI Write Protection Status Register (SPI_WPSR) is set and the WPVSRC field indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading SPI_WPSR.

The following registers are write-protected when WPEN is set in SPI_WPMR:

- SPI Mode Register
- SPI Chip Select Register

## 46.8 Serial Peripheral Interface (SPI) User Interface

In the “Offset” column of Table 46-5, ‘CS_number’ denotes the chip select number.

**Table 46-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | SPI_CR | Write-only | – |
| 0x04 | Mode Register | SPI_MR | Read/Write | 0x0 |
| 0x08 | Receive Data Register | SPI_RDR | Read-only | 0x0 |
| 0x0C | Transmit Data Register | SPI_TDR | Write-only | – |
| 0x10 | Status Register | SPI_SR | Read-only | 0x0 |
| 0x14 | Interrupt Enable Register | SPI_IER | Write-only | – |
| 0x18 | Interrupt Disable Register | SPI_IDR | Write-only | – |
| 0x1C | Interrupt Mask Register | SPI_IMR | Read-only | 0x0 |
| 0x20–0x2C | Reserved | – | – | – |
| 0x30 + (CS_number * 0x04) | Chip Select Register | SPI_CSR | Read/Write | 0x0 |
| 0x40 | FIFO Mode Register | SPI_FMR | Read/Write | 0x0 |
| 0x44 | FIFO Level Register | SPI_FLR | Read-only | 0x0 |
| 0x48 | Comparison Register | SPI_CMPR | Read-only | 0x0 |
| 0x4C–0xE0 | Reserved | – | – | – |
| 0xE4 | Write Protection Mode Register | SPI_WPMR | Read/Write | 0x0 |
| 0xE8 | Write Protection Status Register | SPI_WPSR | Read-only | 0x0 |
| 0xEC–0xF8 | Reserved | – | – | – |
| 0xFC | Reserved | – | – | – |

### 46.8.1 SPI Control Register

**Name:** SPI_CR

**Address:** 0xF8000000 (0), 0xFC000000 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FIFODIS | FIFOEN | – | – | – | – | – | LASTXFER |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | RXFCLR | TXFCLR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | REQCLR | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | – | – | – | – | – | SPIDIS | SPIEN |

- **SPIEN: SPI Enable**

0: No effect.

1: Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0: No effect.f

1: Disables the SPI.

All pins are set in Input mode after completion of the transmission in progress, if any.

If a transfer is in progress when SPIDIS is set, the SPI completes the transmission of the shifter register and does not start any new transfer, even if the SPI_THR is loaded.

Note: If both SPIEN and SPIDIS are equal to one when the SPI_CR is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0: No effect.

1: Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in Slave mode after software reset.

- **REQCLR: Request to Clear the Comparison Trigger**

SleepWalking enabled:

0: No effect.

1: Clears the potential clock request currently issued by SPI, thus the potential system wakeup is cancelled.

SleepWalking disabled:

0: No effect.

1: Restarts the comparison trigger to enable SPI_RDR loading.

- **TXFCLR: Transmit FIFO Clear**

0: No effect.

1: Clears the Transmit FIFO, Transmit FIFO will become empty.

- **RXFCLR: Receive FIFO Clear**

0: No effect.

1: Clears the Receive FIFO, Receive FIFO will become empty.

- **LASTXFER: Last Transfer**

0: No effect.

1: The current NPCS is deasserted after the character written in TD has been transferred. When SPI_CSRx.CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

Refer to Section 46.7.3.5 “Peripheral Selection” for more details.

- **FIFOEN: FIFO Enable**

0: No effect.

1: Enables the Transmit and Receive FIFOs

- **FIFODIS: FIFO Disable**

0: No effect.

1: Disables the Transmit and Receive FIFOs

### 46.8.2 SPI Mode Register

**Name:** SPI_MR

**Address:** 0xF8000004 (0), 0xFC000004 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCS | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | CMPMODE | – | – | – | LSBHALF |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LLB | – | WDRBT | MODFDIS | BRSRCCLK | PCSDEC | PS | MSTR |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

- **MSTR: Master/Slave Mode**

0: SPI is in Slave mode

1: SPI is in Master mode

- **PS: Peripheral Select**

0: Fixed Peripheral Select

1: Variable Peripheral Select

- **PCSDEC: Chip Select Decode**

0: The chip select lines are directly connected to a peripheral device.

1: The four NPCS chip select lines are connected to a 4-bit to 16-bit decoder.

When PCSDEC = 1, up to 15 chip select signals can be generated with the four NPCS lines using an external 4-bit to 16-bit decoder. The chip select registers define the characteristics of the 15 chip selects, with the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 3.

SPI_CSR1 defines peripheral chip select signals 4 to 7.

SPI_CSR2 defines peripheral chip select signals 8 to 11.

SPI_CSR3 defines peripheral chip select signals 12 to 14.

- **BRSRCCLK: Bit Rate Source Clock**

0 (PERIPH_CLK): The peripheral clock is the source clock for the bit rate generation.

1 (GCLK): PMC GCLK is the source clock for the bit rate generation, thus the bit rate can be independent of the core/peripheral clock.

Note: If bit BRSRCCLK = 1, the SCBR field in SPI_CSRx must be programmed with a value greater than 1.

- **MODFDIS: Mode Fault Detection**

0: Mode fault detection enabled

1: Mode fault detection disabled

- **WDRBT: Wait Data Read Before Transfer**

0: No Effect. In Master mode, a transfer can be initiated regardless of the SPI_RDR state.

1: In Master mode, a transfer can start only if the SPI_RDR is empty, i.e., does not contain any unread data. This mode prevents overrun error in reception.

- **LLB: Local Loopback Enable**

0: Local loopback path disabled.

1: Local loopback path enabled.

LLB controls the local loopback on the data shift register for testing in Master mode only (MISO is internally connected on MOSI).

- **LSBHALF: LSB Timing Selection**

0: To be used only if SPI slave LSB timing is 100% compliant with SPI standard (LSB duration is a full bit time). This value gives the better margin for SPI slave response delay (less than 1 SPCK clock cycle).

1: To be selected if the SPI slave LSB timing does not behave as the SPI standard (not triggered by NPCS deassertion in mode), the slave response delay is limited to less than 1/2 SPCK cycle.

- **CMPMODE: Comparison Mode**

| Value | Name | Description |
|---|---|---|
| 0 | FLAG_ONLY | Any character is received and comparison function drives CMP flag. |
| 1 | START_CONDITION | Comparison condition must be met to start reception of all incoming characters until REQCLR is set. |

- **PCS: Peripheral Chip Select**

This field is only used if fixed peripheral select is active (PS = 0).

If SPI_MR.PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don’t care)

If SPI_MR.PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay between the inactivation and the activation of NPCS. The DLYBCS time guarantees nonoverlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is lower than 6, six peripheral clock periods are inserted by default.

Otherwise, the following equations determine the delay:

If BRSRCCLK = 0:

$$\text{Delay Between Chip Selects} = \frac{\text{DLYBCS}}{f_{\text{peripheral clock}}}$$

If BRSRCCLK = 1:

$$\text{Delay Between Chip Selects} = \frac{\text{DLYBCS}}{f_{\text{GCLK}}}$$

### 46.8.3 SPI Receive Data Register

**Name:** SPI_RDR

**Address:** 0xF8000008 (0), 0xFC000008 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD | | | | | | | |

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

- **PCS: Peripheral Chip Select**

In Master mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits are read as zero.

Note: When using Variable Peripheral Select mode (PS = 1 in SPI_MR), it is mandatory to set the SPI_MR.WDRBT bit if the PCS field must be processed in SPI_RDR.

### 46.8.4 SPI Receive Data Register (FIFO Multiple Data, 8-bit)

**Name:** SPI_RDR (FIFO_MULTI_DATA_8)

**Address:** 0xF8000008 (0), 0xFC000008 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RD3 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RD2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in SPI_CR), refer to Section 46.7.7.6 “Multiple Data Mode”.

- **RDx: Receive Data**

First unread data in the Receive FIFO. Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

### 46.8.5 SPI Receive Data Register (FIFO Multiple Data, 16-bit)

**Name:** SPI_RDR (FIFO_MULTI_DATA_16)

**Address:** 0xF8000008 (0), 0xFC000008 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RD1 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RD1 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD0 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in SPI_CR), refer to Section 46.7.7.6 “Multiple Data Mode”.

- **RDx: Receive Data**

First unread data in the Receive FIFO. Data received by the SPI Interface is stored in this register in a right-justified format. Unused bits are read as zero.

### 46.8.6 SPI Transmit Data Register

**Name:** SPI_TDR

**Address:** 0xF800000C (0), 0xFC00000C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | LASTXFER |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | PCS | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TD | | | | | | | |

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

- **PCS: Peripheral Chip Select**

This field is only used if variable peripheral select is active (PS = 1).

If SPI_MR.PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don’t care)

If SPI_MR.PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **LASTXFER: Last Transfer**

0: No effect

1: The current NPCS is deasserted after the transfer of the character written in TD. When SPI_CSRx.CSAAT is set, the communication with the current serial peripheral can be closed by raising the corresponding NPCS line as soon as TD transfer is completed.

This field is only used if variable peripheral select is active (SPI_MR.PS = 1).

### 46.8.7 SPI Transmit Data Register (FIFO Multiple Data, 8- to 16-bit)

**Name:** SPI_TDR (FIFO_MULTI_DATA)

**Address:** 0xF800000C (0), 0xFC00000C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TD1 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TD1 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TD0 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TD0 | | | | | | | |

Note: If FIFO is enabled (FIFOEN bit in SPI_CR), refer to Section 46.7.7.6 “Multiple Data Mode”.

- **TDx: Transmit Data**

Next data to write in the Transmit FIFO. Information to be transmitted must be written to the transmit data register in a right-justified format.

### 46.8.8 SPI Status Register

**Name:** SPI_SR

**Address:** 0xF8000010 (0), 0xFC000010 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | SPIENS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

- **RDRF: Receive Data Register Full (cleared by reading SPI_RDR)**

0: No data has been received since the last read of SPI_RDR.

1: Data has been received and the received data has been transferred from the shift register to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty (cleared by writing SPI_TDR)**

0: Data has been written to SPI_TDR and not yet transferred to the shift register.

1: The last data written in the SPI_TDR has been transferred to the shift register.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to 1.

- **MODF: Mode Fault Error (cleared on read)**

0: No mode fault has been detected since the last read of SPI_SR.

1: A mode fault occurred since the last read of SPI_SR.

- **OVRES: Overrun Error Status (cleared on read)**

0: No overrun has been detected since the last read of SPI_SR.

1: An overrun has occurred since the last read of SPI_SR.

An overrun occurs when SPI_RDR is loaded at least twice from the shift register since the last read of the SPI_RDR.

- **NSSR: NSS Rising (cleared on read)**

0: No rising edge detected on NSS pin since the last read of SPI_SR.

1: A rising edge occurred on NSS pin since the last read of SPI_SR.

- **TXEMPTY: Transmission Registers Empty (cleared by writing SPI_TDR)**

0: As soon as data is written in SPI_TDR.

1: SPI_TDR and internal shift register are empty. If a transfer delay has been defined, TXEMPTY is set after the end of this delay.

- **UNDES: Underrun Error Status (Slave mode only) (cleared on read)**

0: No underrun has been detected since the last read of SPI_SR.

1: A transfer starts whereas no data has been loaded in SPI_TDR.

- **CMP: Comparison Status (cleared on read)**

0: No received character matched the comparison criteria programmed in VAL1 and VAL2 fields in SPI_CMPR since the last read of SPI_SR.

1: A received character matched the comparison criteria since the last read of SPI_SR.

- **SPIENS: SPI Enable Status**

0: SPI is disabled.

1: SPI is enabled.

- **TXFEF: Transmit FIFO Empty Flag (cleared on read)**

0: Transmit FIFO is not empty.

1: Transmit FIFO has been emptied since the last read of SPI_SR.

- **TXFFF: Transmit FIFO Full Flag (cleared on read)**

0: Transmit FIFO is not full or TXFF flag has been cleared.

1: Transmit FIFO has been filled since the last read of SPI_SR.

- **TXFTHF: Transmit FIFO Threshold Flag (cleared on read)**

0: Number of DATA in Transmit FIFO is above TXFTHRES threshold.

1: Number of DATA in Transmit FIFO has reached TXFTHRES threshold since the last read of SPI_SR.

- **RXFEF: Receive FIFO Empty Flag**

0: Receive FIFO is not empty or RXFE flag has been cleared.

1: Receive FIFO has become empty (coming from “not empty” state to “empty” state).

- **RXFFF: Receive FIFO Full Flag**

0: Receive FIFO is not empty or RXFE flag has been cleared.

1: Receive FIFO has become full (coming from “not full” state to “full” state).

- **RXFTHF: Receive FIFO Threshold Flag**

0: Number of unread DATA in Receive FIFO is below RXFTHRES threshold or RXFTH flag has been cleared.

1: Number of unread DATA in Receive FIFO has reached RXFTHRES threshold (coming from “below threshold” state to “equal or above threshold” state).

- **TXFPTEF: Transmit FIFO Pointer Error Flag**

0: No Transmit FIFO pointer occurred

1: Transmit FIFO pointer error occurred. Transceiver must be reset

See Section 46.7.7.7 “FIFO Pointer Error” for details.

- **RXFPTEF: Receive FIFO Pointer Error Flag**

0: No Receive FIFO pointer occurred

1: Receive FIFO pointer error occurred. Receiver must be reset

See Section 46.7.7.7 “FIFO Pointer Error” for details.

### 46.8.9 SPI Interrupt Enable Register

**Name:** SPI_IER

**Address:** 0xF8000014 (0), 0xFC000014 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **NSSR: NSS Rising Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **UNDES: Underrun Error Interrupt Enable**
- **CMP: Comparison Interrupt Enable**
- **TXFEF: TXFEF Interrupt Enable**
- **TXFFF: TXFFF Interrupt Enable**
- **TXFTHF: TXFTHF Interrupt Enable**
- **RXFEF: RXFEF Interrupt Enable**
- **RXFFF: RXFFF Interrupt Enable**
- **RXFTHF: RXFTHF Interrupt Enable**
- **TXFPTEF: TXFPTEF Interrupt Enable**
- **RXFPTEF: RXFPTEF Interrupt Enable**

### 46.8.10 SPI Interrupt Disable Register

**Name:** SPI_IDR

**Address:** 0xF8000018 (0), 0xFC000018 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **NSSR: NSS Rising Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **UNDES: Underrun Error Interrupt Disable**
- **CMP: Comparison Interrupt Disable**
- **TXFEF: TXFEF Interrupt Disable**
- **TXFFF: TXFFF Interrupt Disable**
- **TXFTHF: TXFTHF Interrupt Disable**
- **RXFEF: RXFEF Interrupt Disable**
- **RXFFF: RXFFF Interrupt Disable**
- **RXFTHF: RXFTHF Interrupt Disable**
- **TXFPTEF: TXFPTEF Interrupt Disable**
- **RXFPTEF: RXFPTEF Interrupt Disable**

### 46.8.11 SPI Interrupt Mask Register

**Name:** SPI_IMR

**Address:** 0xF800001C (0), 0xFC00001C (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| RXFPTEF | TXFPTEF | RXFTHF | RXFFF | RXFEF | TXFTHF | TXFFF | TXFEF |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | CMP | UNDES | TXEMPTY | NSSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | MODF | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **NSSR: NSS Rising Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **UNDES: Underrun Error Interrupt Mask**
- **CMP: Comparison Interrupt Mask**
- **TXFEF: TXFEF Interrupt Mask**
- **TXFFF: TXFFF Interrupt Mask**
- **TXFTHF: TXFTHF Interrupt Mask**
- **RXFEF: RXFEF Interrupt Mask**
- **RXFFF: RXFFF Interrupt Mask**
- **RXFTHF: RXFTHF Interrupt Mask**
- **TXFPTEF: TXFPTEF Interrupt Mask**
- **RXFPTEF: RXFPTEF Interrupt Mask**

### 46.8.12 SPI Chip Select Register

**Name:** SPI_CSRx [x=0..3]

**Address:** 0xF8000030 (0), 0xFC000030 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYBCT | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DLYBS | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SCBR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BITS | | | | CSAAT | CSNAAT | NCPHA | CPOL |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

Note: SPI_CSRx must be written even if the user wants to use the default reset values. The BITS field is not updated with the translated value unless the register is written.

- **CPOL: Clock Polarity**

0: The inactive state value of SPCK is logic level zero.

1: The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0: Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1: Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSNAAT: Chip Select Not Active After Transfer (Ignored if CSAAT = 1)**

0: The Peripheral Chip Select Line does not rise between two transfers if the SPI_TDR is reloaded before the end of the first transfer and if the two transfers occur on the same chip select.

1: The Peripheral Chip Select Line rises systematically after each transfer performed on the same slave. It remains inactive after the end of transfer for a minimal duration of:

If SPI_MR.BRSRCCLK = 0:

$$\frac{DLYBCS}{f_{\text{peripheral clock}}}$$

If SPI_MR.BRSRCCLK = 1:

$$\frac{DLYBCS}{f_{\text{GCLK}}}$$

If field DLYBCS is lower than 6, a minimum of six periods is introduced.

- **CSAAT: Chip Select Active After Transfer**

0: The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1: The Peripheral Chip Select Line does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

(See the note below the SPI_CSR bitmap.)

The BITS field determines the number of data bits transferred. Reserved values should not be used.

| Value | Name | Description |
|---|---|---|
| 0 | 8_BIT | 8 bits for transfer |
| 1 | 9_BIT | 9 bits for transfer |
| 2 | 10_BIT | 10 bits for transfer |
| 3 | 11_BIT | 11 bits for transfer |
| 4 | 12_BIT | 12 bits for transfer |
| 5 | 13_BIT | 13 bits for transfer |
| 6 | 14_BIT | 14 bits for transfer |
| 7 | 15_BIT | 15 bits for transfer |
| 8 | 16_BIT | 16 bits for transfer |
| 9 | – | Reserved |
| 10 | – | Reserved |
| 11 | – | Reserved |
| 12 | – | Reserved |
| 13 | – | Reserved |
| 14 | – | Reserved |
| 15 | – | Reserved |

- **SCBR: Serial Clock Bit Rate**

In Master mode, the SPI Interface uses a modulus counter to derive the SPCK bit rate from the clock defined by the SPI_MR.BRSRCCLK bit. The bit rate is selected by writing a value from1 to 255 in the SCBR field. The following equations determine the SPCK bit rate:

If SPI_MR.BRSRCCLK = 0: SCBR = $f_{peripheral\ clock}$ / SPCK Bit Rate

If SPI_MR.BRSRCCLK = 1: SCBR = $f_{GCLK}$ / SPCK Bit Rate

Programming the SCBR field to 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

If BRSRCCLK = 1 in SPI_MR, SCBR must be programmed with a value greater than 1.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

Note: If one of the SCBR fields in SPI_CSRx is set to 1, the other SCBR fields in SPI_CSRx must be set to 1 as well, if they are used to process transfers. If they are not used to transfer data, they can be set at any value.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS falling edge (activation) to the first valid SPCK transition.

When DLYBS = 0, the delay is half the SPCK clock period.

Otherwise, the following equations determine the delay:

If SPI_MR.BRSRCCLK = 0: DLYBS = Delay Before SPCK × $f_{peripheral\ clock}$

If SPI_MR.BRSRCCLK = 1: DLYBS = Delay Before SPCK × $f_{GCLK}$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT = 0, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equations determine the delay:

If SPI_MR.BRSRCCLK = 0: DLYBCT = Delay Between Consecutive Transfers × $f_{peripheral\ clock}$ / 32

If SPI_MR.BRSRCCLK = 1: DLYBCT = Delay Between Consecutive Transfers × $f_{GCLK}$ / 32

### 46.8.13 SPI FIFO Mode Register

**Name:** SPI_FMR

**Address:** 0xF8000040 (0), 0xFC000040 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | RXFTHRES | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | TXFTHRES | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | RXRDYM | | – | – | TXRDYM | |

- **TXRDYM: Transmitter Data Register Empty Mode**

If FIFOs are enabled, the TDRE flag (in SPI_SR) will behave as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | TDRE will be at level ‘1’ when at least one data can be written in the Transmit FIFO. |
| 0x1 | TWO_DATA | TDRE will be at level ‘1’ when at least two data can be written in the Transmit FIFO. |
| 0x2 | FOUR_DATA | TDRE will be at level ‘1’ when at least four data can be written in the Transmit FIFO. |

- **RXRDYM: Receiver Data Register Full Mode**

If FIFOs are enabled, the RDRF flag (in SPI_SR) will behave as follows.

| Value | Name | Description |
|---|---|---|
| 0x0 | ONE_DATA | RDRF will be at level ‘1’ when at least one unread data is in the Receive FIFO. |
| 0x1 | TWO_DATA | RDRF will be at level ‘1’ when at least two unread data are in the Receive FIFO. |
| 0x2 | FOUR_DATA | RDRF will be at level ‘1’ when at least four unread data are in the Receive FIFO. |

- **TXFTHRES: Transmit FIFO Threshold**

0–16: Defines the Transmit FIFO threshold value (number of data). TXFTH flag in SPI_SR will be set when Transmit FIFO goes from “above” threshold state to “equal or below” threshold state.

- **RXFTHRES: Receive FIFO Threshold**

0–16: Defines the Receive FIFO threshold value (number of data). RXFTH flag in SPI_SR will be set when Receive FIFO goes from “below” threshold state to “equal or above” threshold state.

### 46.8.14 SPI FIFO Level Register

**Name:** SPI_FLR

**Address:** 0xF8000044 (0), 0xFC000044 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | RXFL | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TXFL | | | | | |

- **TXFL: Transmit FIFO Level**

0: There is no data in the Transmit FIFO

1–16: Indicates the number of DATA in the Transmit FIFO

- **RXFL: Receive FIFO Level**

0: There is no unread data in the Receive FIFO

1–16: Indicates the number of unread DATA in the Receive FIFO

### 46.8.15 SPI Comparison Register

**Name:** SPI_CMPR

**Address:** 0xF8000048 (0), 0xFC000048 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| VAL2 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| VAL2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| VAL1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VAL1 | | | | | | | |

This register can only be written if the WPEN bit is cleared in the SPI Write Protection Mode Register.

- **VAL1: First Comparison Value for Received Character**

0–65535: The received character must be higher or equal to the value of VAL1 and lower or equal to VAL2 to set CMP flag in SPI_SR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the SPI requests a system wakeup if the condition is met.

- **VAL2: Second Comparison Value for Received Character**

0–65535: The received character must be lower or equal to the value of VAL2 and higher or equal to VAL1 to set CMP flag in SPI_CSR. If asynchronous partial wakeup (SleepWalking) is enabled in PMC_SLPWK_ER, the SPI requests a system wakeup if condition is met.

### 46.8.16 SPI Write Protection Mode Register

**Name:** SPI_WPMR

**Address:** 0xF80000E4 (0), 0xFC0000E4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

1: Enables the write protection if WPKEY corresponds to 0x535049 (“SPI” in ASCII)

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x535049 | PASSWD | Writing any other value in this field aborts the write operation of the WPEN bit.<br>Always reads as 0. |

See Section 46.7.8 “Register Write Protection” for the list of registers that can be write-protected.

### 46.8.17 SPI Write Protection Status Register

**Name:** SPI_WPSR

**Address:** 0xF80000E8 (0), 0xFC0000E8 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of SPI_WPSR.

1: A write protection violation has occurred since the last read of SPI_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

# 47. Quad Serial Peripheral Interface (QSPI)

## 47.1 Description

The Quad Serial Peripheral Interface (QSPI) is a synchronous serial data link that provides communication with external devices in Master mode.

The QSPI can be used in SPI mode to interface to serial peripherals such as ADCs, DACs, LCD controllers, CAN controllers and sensors, or in Serial Memory mode to interface to serial Flash memories.

The QSPI allows the system to execute code directly from a serial Flash memory (XIP) without code shadowing to RAM. The serial Flash memory mapping is seen in the system as other memories such as ROM, SRAM, DRAM, embedded Flash memory, etc.

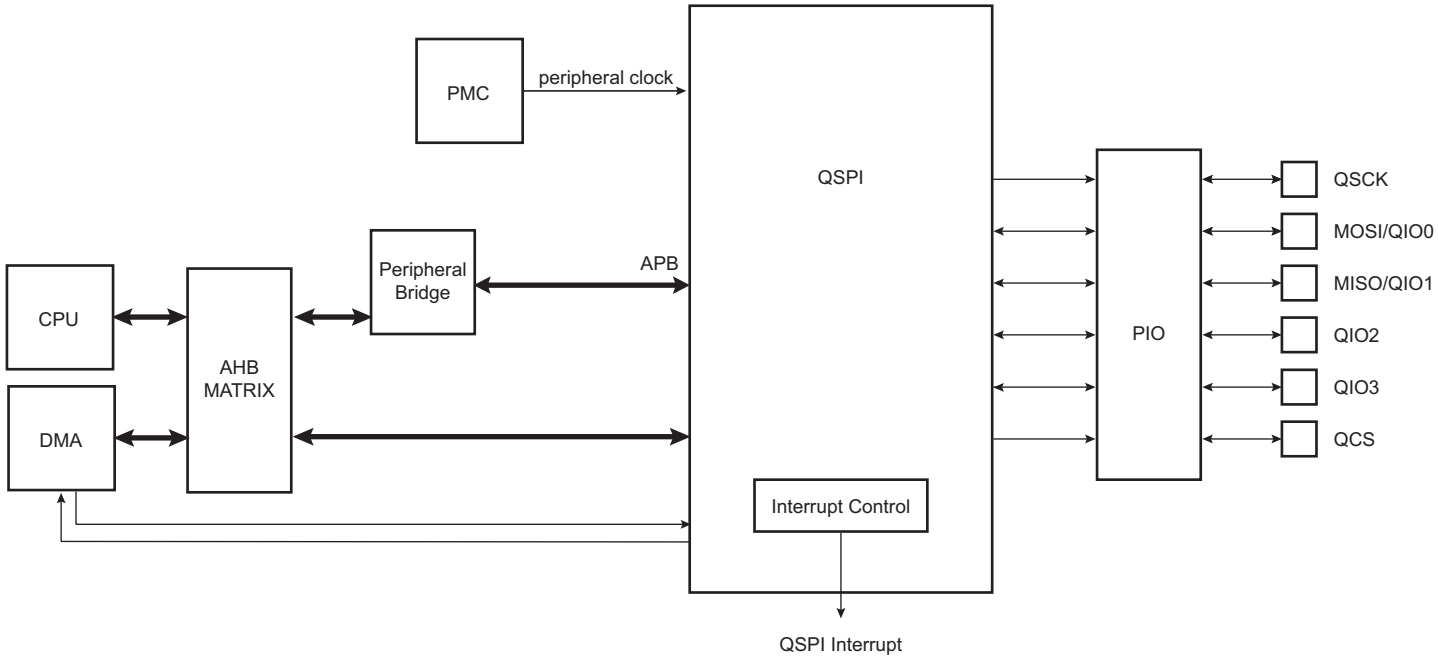
With the support of the Quad SPI protocol, the QSPI allows the system to use high-performance serial Flash memories which are small and inexpensive, in place of larger and more expensive parallel Flash memories.

## 47.2 Embedded Characteristics

- Master SPI Interface
  - Programmable Clock Phase and Clock Polarity
  - Programmable Transfer Delays Between Consecutive Transfers, Between Clock and Data, Between Deactivation and Activation of Chip Select
- SPI Mode
  - Interface to Serial Peripherals such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - 8-bit/16-bit/32-bit Programmable Data Length
- Serial Memory Mode
  - Interface to Serial Flash Memories Operating in Single-bit SPI, Dual SPI and Quad SPI
  - Interface to Serial Flash Memories Operating in Single Data Rate or Double Data Rate Modes
  - Supports “Execute In Place” (XIP)— Code Execution by the System Directly from a Serial Flash Memory
  - Flexible Instruction Register for Compatibility with All Serial Flash Memories
  - 32-bit Address Mode (default is 24-bit address) to Support Serial Flash Memories Larger than 128 Mbit
  - Continuous Read Mode
  - Scrambling/Unscrambling “On-The-Fly”
- Connection to DMA Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
- Register Write Protection

## 47.3 Block Diagram

**Figure 47-1. Block Diagram**




## 47.4 Signal Description

**Table 47-1. Signal Description**

| Pin Name | Pin Description | Type |
|---|---|---|
| QSCK | Serial Clock | Output |
| MOSI (QIO0) (1) (2) | Data Output (Data Input Output 0) | Output (Input/Output) |
| MISO (QIO1) (1) (2) | Data Input (Data Input Output 1) | Input (Input/Output) |
| QIO2 (3) | Data Input Output 2 | Input/Output |
| QIO3 (3) | Data Input Output 3 | Input/Output |
| QCS | Peripheral Chip Select | Output |

Notes:
1. MOSI and MISO are used for single-bit SPI operation.
2. QIO0–QIO1 are used for Dual SPI operation.
3. QIO0–QIO3 are used for Quad SPI operation.

## 47.5 Product Dependencies

### 47.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the QSPI pins to their peripheral functions.

**Table 47-2. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| QSPI0 | QSPI0_CS | PA1 | B |
| QSPI0 | QSPI0_CS | PA15 | C |
| QSPI0 | QSPI0_CS | PA23 | F |
| QSPI0 | QSPI0_IO0 | PA2 | B |
| QSPI0 | QSPI0_IO0 | PA16 | C |
| QSPI0 | QSPI0_IO0 | PA24 | F |
| QSPI0 | QSPI0_IO1 | PA3 | B |
| QSPI0 | QSPI0_IO1 | PA17 | C |
| QSPI0 | QSPI0_IO1 | PA25 | F |
| QSPI0 | QSPI0_IO2 | PA4 | B |
| QSPI0 | QSPI0_IO2 | PA18 | C |
| QSPI0 | QSPI0_IO2 | PA26 | F |
| QSPI0 | QSPI0_IO3 | PA5 | B |
| QSPI0 | QSPI0_IO3 | PA19 | C |
| QSPI0 | QSPI0_IO3 | PA27 | F |
| QSPI0 | QSPI0_SCK | PA0 | B |
| QSPI0 | QSPI0_SCK | PA14 | C |
| QSPI0 | QSPI0_SCK | PA22 | F |
| QSPI1 | QSPI1_CS | PA11 | B |
| QSPI1 | QSPI1_CS | PB6 | D |
| QSPI1 | QSPI1_CS | PB15 | E |
| QSPI1 | QSPI1_IO0 | PA7 | B |
| QSPI1 | QSPI1_IO0 | PB7 | D |
| QSPI1 | QSPI1_IO0 | PB16 | E |
| QSPI1 | QSPI1_IO1 | PA8 | B |
| QSPI1 | QSPI1_IO1 | PB8 | D |
| QSPI1 | QSPI1_IO1 | PB17 | E |
| QSPI1 | QSPI1_IO2 | PA9 | B |
| QSPI1 | QSPI1_IO2 | PB9 | D |
| QSPI1 | QSPI1_IO2 | PB18 | E |
| QSPI1 | QSPI1_IO3 | PA10 | B |
| QSPI1 | QSPI1_IO3 | PB10 | D |

Table 47-2. I/O Lines (Continued)

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| QSPI1 | QSPI1_IO3 | PB19 | E |
| QSPI1 | QSPI1_SCK | PA6 | B |
| QSPI1 | QSPI1_SCK | PB5 | D |
| QSPI1 | QSPI1_SCK | PB14 | E |

### 47.5.2 Power Management

The QSPI may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the QSPI clock.

### 47.5.3 Interrupt Sources

The QSPI has an interrupt line connected to the Interrupt Controller. Handling the QSPI interrupt requires programming the interrupt controller before configuring the QSPI.

Table 47-3. Peripheral IDs

| Instance | ID |
|---|---|
| QSPI0 | 52 |
| QSPI1 | 53 |

### 47.5.4 Direct Memory Access Controller (DMA)

The QSPI can be used in conjunction with the Direct Memory Access Controller (DMA) in order to reduce processor overhead. For a full description of the DMA, refer to the section “DMA Controller (XDMAC)”.

## 47.6 Functional Description

### 47.6.1 Serial Clock Baud Rate

The QSPI baud rate clock is generated by dividing the peripheral clock by a value between 1 and 256.

### 47.6.2 Serial Clock Phase and Polarity

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the QSPI Serial Clock register (QSPI_SCR). The CPHA bit in the QSPI_SCR programs the clock phase. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, the interfaced slave must use the same parameter values to communicate.

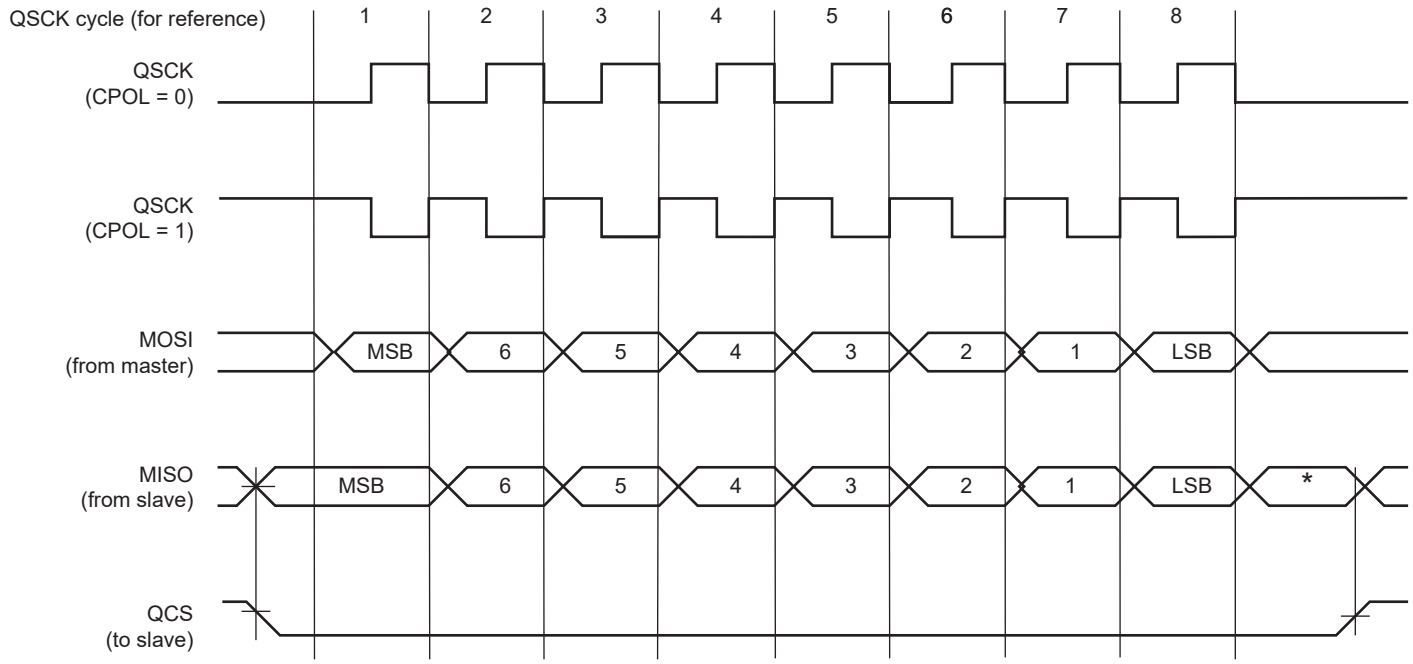
Table 47-4 shows the four modes and corresponding parameter settings.

Table 47-4. QSPI Bus Clock Modes

| QSPI Clock Mode | QSPI_SCR.CPOL | QSPI_SCR.CPHA | Shift QSCK Edge | Capture QSCK Edge | QSCK Inactive Level |
|---|---|---|---|---|---|
| 0 | 0 | 0 | Falling | Rising | Low |
| 1 | 0 | 1 | Rising | Falling | Low |
| 2 | 1 | 0 | Rising | Falling | High |
| 3 | 1 | 1 | Falling | Rising | High |

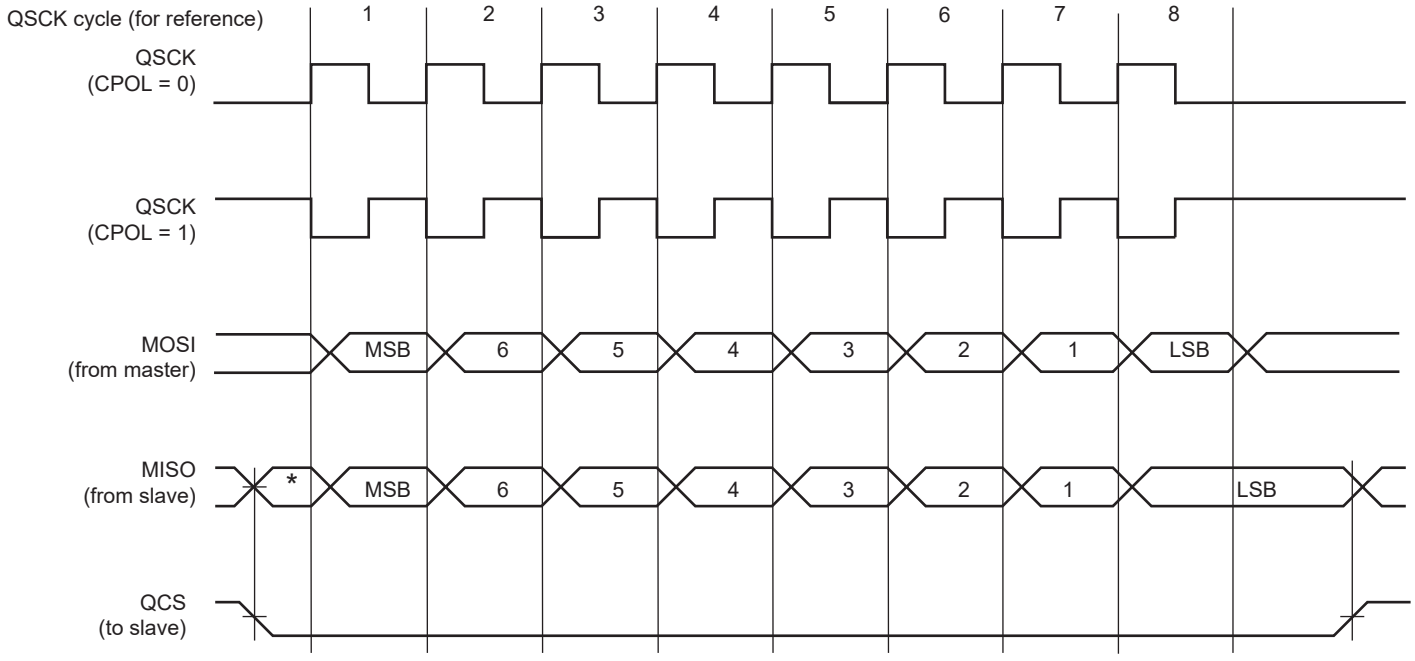
Figure 47-2 and Figure 47-3 show examples of data transfers.

**Figure 47-2. QSPI Transfer Format (QSPI_SCR.CPHA = 0, 8 bits per transfer)**




* Not defined, but normally MSB of previous character received.

**Figure 47-3. QSPI Transfer Format (QSPI_SCR.CPHA = 1, 8 bits per transfer)**




* Not defined but normally LSB of previous character transmitted.

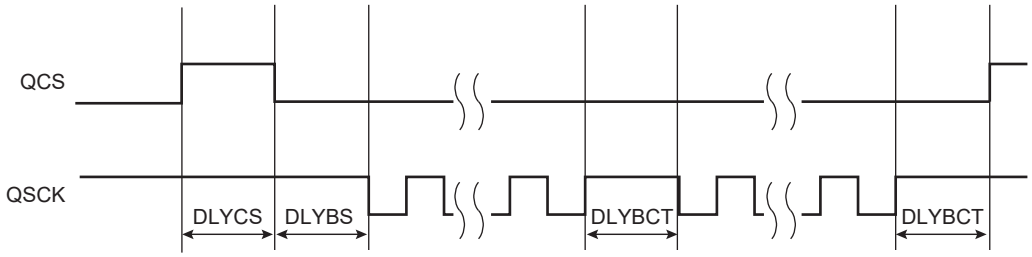
### 47.6.3 Transfer Delays

Figure 47-4 shows several consecutive transfers while the chip select is active. Three delays can be programmed to modify the transfer waveforms:

- The delay between the deactivation and the activation of QCS, programmed by writing QSPI_MR.DLYCS. Allows to adjust the minimum time of QCS at high level.
- The delay before QSCK, programmed by writing QSPI_SR.DLYBS. Allows the start of QSCK to be delayed after the chip select has been asserted.
- The delay between consecutive transfers, programmed by writing QSPI_MR.DLYBCT. Allows insertion of a delay between two consecutive transfers. In Serial Memory mode, this delay is not programmable and DLYBCT is ignored. In this mode, DLYBCT must be written to ‘0’.

These delays allow the QSPI to be adapted to the interfaced peripherals and their speed and bus release time.

**Figure 47-4. Programmable Delays**




### 47.6.4 QSPI SPI Mode

In SPI mode, the QSPI acts as a regular SPI Master.

To activate this mode, QSPI_MR.SMM must be written to ‘0’ in QSPI_MR.

#### 47.6.4.1 SPI Mode Operations

The QSPI in standard SPI mode operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave connected to the SPI bus. The QSPI drives the chip select line to the slave (QCS) and the serial clock signal (QSCK).

The QSPI features two holding registers, the Transmit Data register (QSPI_TDR) and the Receive Data register (QSPI_RDR), and a single internal shift register. The holding registers maintain the data flow at a constant rate.

After enabling the QSPI, a data transfer begins when the processor writes to the QSPI_TDR. The written data is immediately transferred to the internal shift register and transfer on the SPI bus starts. While the data in the internal shift register is shifted on the MOSI line, the MISO line is sampled and shifted to the internal shift register. Receiving data cannot occur without transmitting data. If receiving mode is not needed, for example when communicating with a slave receiver only (such as an LCD), the receive status flags in the Status register (QSPI_SR) can be discarded.

If new data is written in QSPI_TDR during the transfer, it is retained there until the current transfer is completed. Then, the received data is transferred from the internal shift register to the QSPI_RDR, the data in QSPI_TDR is loaded in the internal shift register and a new transfer starts.

The transfer of a data written in QSPI_TDR in the internal shift register is indicated by the Transmit Data Register Empty (TDRE) bit in the QSPI_SR. When new data is written in QSPI_TDR, this bit is cleared. QSPI_SR.TDRE is used to trigger the Transmit DMA channel.

The end of transfer is indicated by the TXEMPTY flag in the QSPI_SR. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, QSPI_SR.TXEMPTY is set after the completion of this delay. The peripheral clock can be switched off at this time.

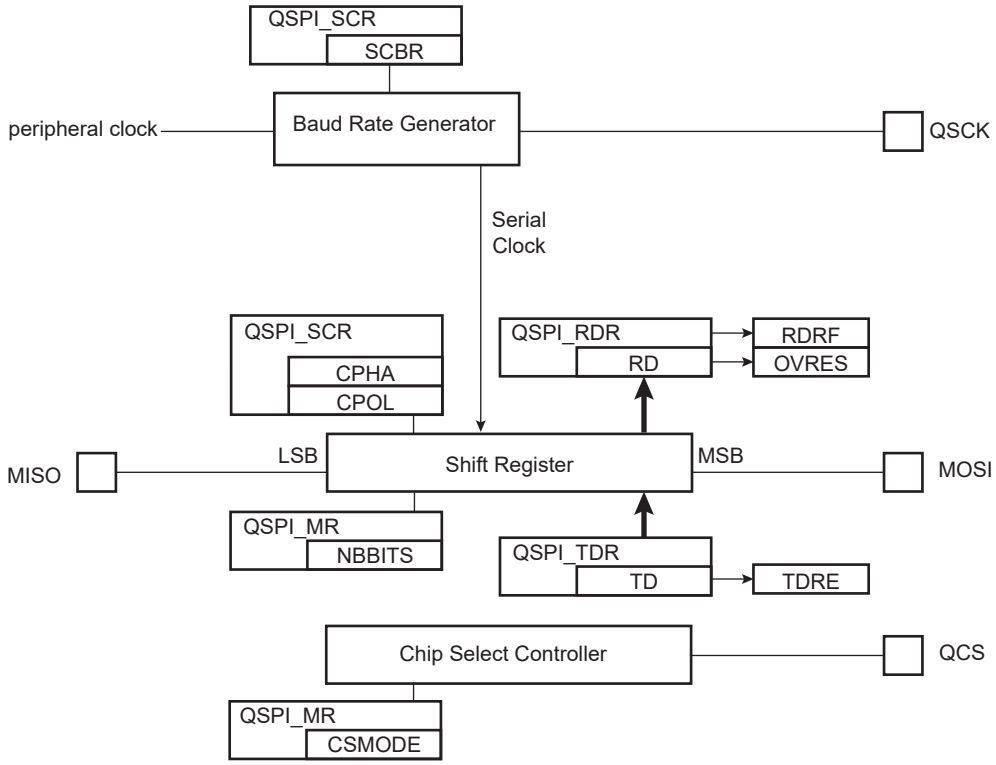
The transfer of received data from the internal shift register in QSPI_RDR is indicated by the Receive Data Register Full (RDRF) bit in the QSPI_SR. When the received data is read, QSPI_SR.RDRF bit is cleared.

If the QSPI_RDR has not been read before new data is received, the Overrun Error Status (OVRES) bit in QSPI_SR is set. As long as this flag is set, data is loaded in QSPI_RDR. The user must read the QSPI_SR to clear the OVRES bit.

Figure 47-5 shows a block diagram of the SPI when operating in Master mode. Figure 47-6 shows a flow chart describing how transfers are handled.

#### 47.6.4.2 SPI Mode Block Diagram

**Figure 47-5. SPI Mode Block Diagram**



### 47.6.4.3 SPI Mode Flow Diagram

**Figure 47-6. SPI Mode Flow Diagram**

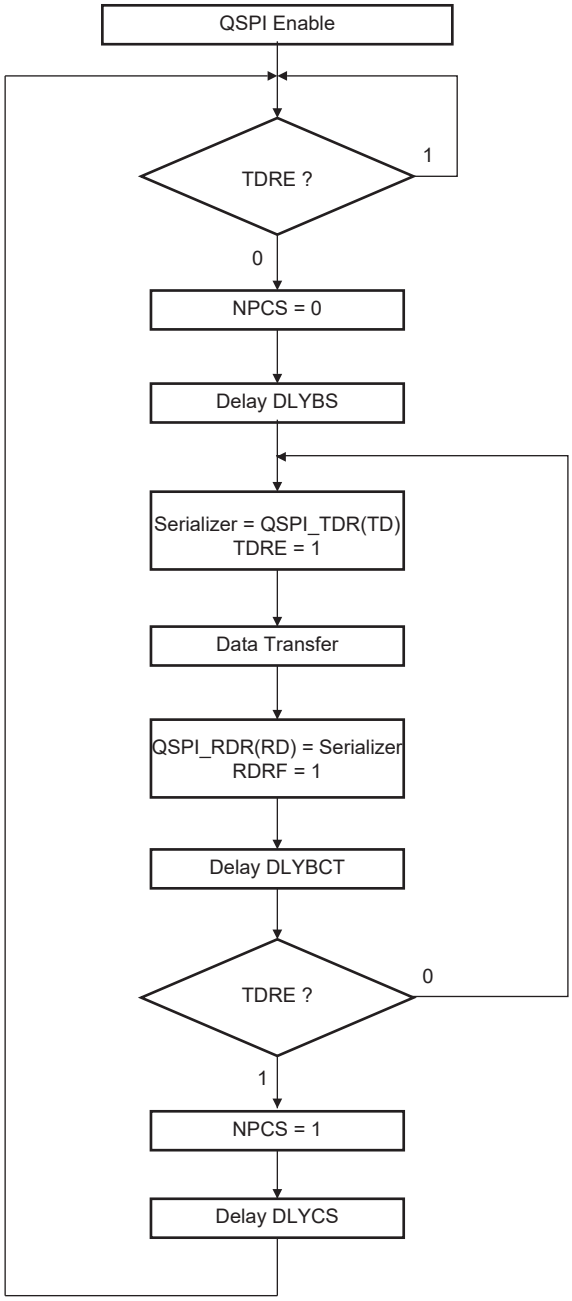
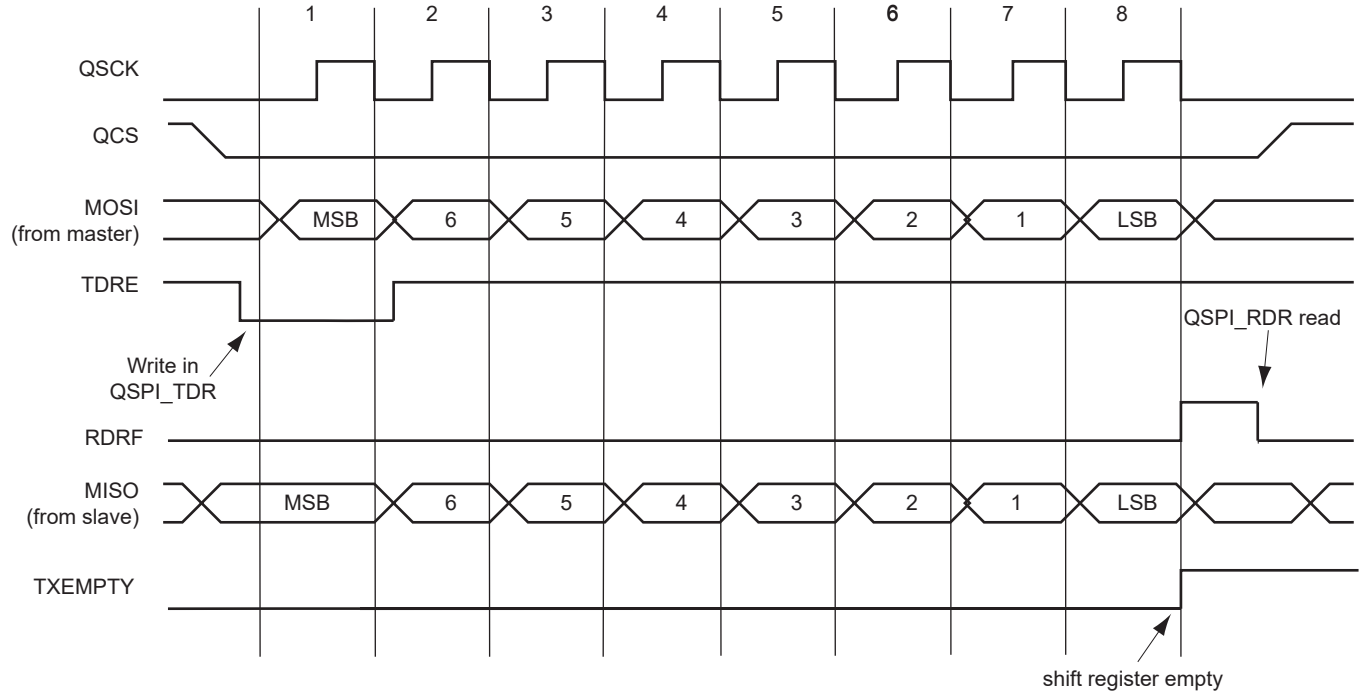



Figure 47-7 shows Transmit Data Register Empty (TDRE), Receive Data Register Full (RDRF) and Transmission Register Empty (TXEMPTY) status flags behavior within the QSPI_SR during an 8-bit data transfer in Fixed mode, without DMA.

**Figure 47-7. Status Register Flags Behavior**




#### 47.6.4.4 Peripheral Deselection without DMA

During a transfer of more than one data on a Chip Select without the DMA, the QSPI_TDR is loaded by the processor and the flag TDRE rises as soon as the content of the QSPI_TDR is transferred into the internal shift register. When this flag is detected high, the QSPI_TDR can be reloaded. If this reload by the processor occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. Depending on the application software handling the QSPI_SR flags (by interrupt or polling method) or servicing other interrupts or other tasks, the processor may not reload the QSPI_TDR in time to keep the chip select active (low). A null Delay Between Consecutive Transfer (DLYBCT) value in the QSPI_MR gives even less time for the processor to reload the QSPI_TDR. With some SPI slave peripherals, requiring the chip select line to remain active (low) during a full set of transfers may lead to communication errors.

To facilitate interfacing with such devices, QSPI_MR.CSMODE may be configured to ‘1’. This allows the chip select lines to remain in their current state (low = active) until the end of transfer is indicated by the Last Transfer (LASTXFER) bit in the Control register (QSPI_CR). Even if the QSPI_TDR is not reloaded, the chip select remains active. To have the chip select line rise at the end of the last data transfer, QSPI_CR.LASTXFER must be written to ‘1’ at the same time or after writing the last data to transmit into the QSPI_TDR.

#### 47.6.4.5 Peripheral Deselection with DMA

When the DMA Controller is used, the Chip Select line remains low during the transfer since the TDRE flag is managed by the DMA itself. Reloading the QSPI_TDR by the DMA is done as soon as the TDRE flag is set. In this case, writing QSPI_MR.CSMODE to ‘1’ may not be needed. However, when other DMA channels connected to other peripherals are also in use, the QSPI DMA could be delayed by another DMA with a higher priority on the bus. Having DMA buffers in slower memories like Flash memory or SDRAM compared to fast internal SRAM, may lengthen the reload time of the QSPI_TDR by the DMA as well. This means that the QSPI_TDR might not be reloaded in time to keep the chip select line low. In this case, the chip select line may toggle between data transfer and according to some SPI Slave devices, the communication might get lost. It may be necessary to configure QSPI_MR.CSMODE to ‘1’.

When QSPI_MR.CSMODE is configured to ‘0’, the QCS does not rise in all cases between two transfers on the same peripheral. During a transfer on a Chip Select, the flag TDRE rises as soon as the content of the QSPI_TDR is transferred into the internal shifter. When this flag is detected, the QSPI_TDR can be reloaded. If this reload occurs before the end of the current transfer and if the next transfer is performed on the same chip select as the current transfer, the Chip Select is not de-asserted between the two transfers. This might lead to difficulties for interfacing with some serial peripherals requiring the chip select to be de-asserted after each transfer. To facilitate interfacing with such devices, the QSPI_MR may be configured with QSPI_MR.CSMODE at ‘2’.

### 47.6.5 QSPI Serial Memory Mode

In Serial Memory mode, the QSPI acts as a serial Flash memory controller. The QSPI can be used to read data from the serial Flash memory allowing the CPU to execute code from it (XIP execute in place). The QSPI can also be used to control the serial Flash memory (Program, Erase, Lock, etc.) by sending specific commands. In this mode, the QSPI is compatible with single-bit SPI, Dual SPI and Quad SPI protocols.

To activate this mode, QSPI_MR.SMM must be written to ‘1’.

In Serial Memory mode, data cannot be transferred by the QSPI_TDR and the QSPI_RDR, but by writing or reading the QSPI memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).

#### 47.6.5.1 Instruction Frame

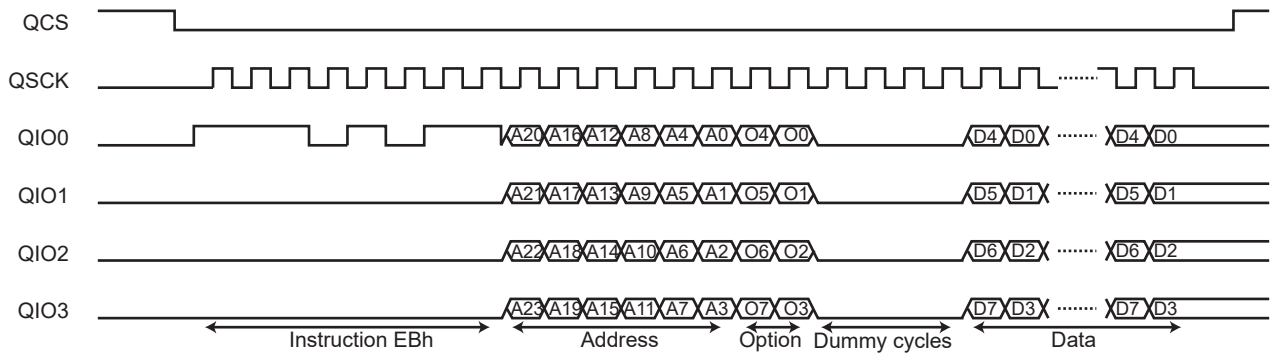
In order to control serial Flash memories, the QSPI is able to send instructions via the SPI bus (ex: READ, PROGRAM, ERASE, LOCK, etc.). Because the instruction set implemented in serial Flash memories is memory vendor dependant, the QSPI includes a complete Instruction Frame register (QSPI_IFR), which makes it very flexible and compatible with all serial Flash memories.

An instruction frame includes:

- An instruction code (size: 8 bits). The instruction is optional in some cases (see Section 47.6.5.4).
- An address (size: 24 bits or 32 bits). The address is optional but is required by instructions such as READ, PROGRAM, ERASE, LOCK. By default the address is 24 bits long, but it can be 32 bits long to support serial Flash memories larger than 128 Mbit (16 Mbyte).
- An option code (size: 1/2/4/8 bits). The option code is not required, but it is useful to activate the XIP mode or the Continuous Read mode (see Section 47.6.5.4) for READ instructions, in some serial Flash memory devices. These modes improve the data read latency.
- Dummy cycles. Dummy cycles are optional but required by some READ instructions.
- Data bytes are optional. Data bytes are present for data transfer instructions such as READ or PROGRAM.

The instruction code, the address/option and the data can be sent with Single-bit SPI, Dual SPI or Quad SPI protocols.

**Figure 47-8. Instruction Frame**



#### 47.6.5.2 Instruction Frame Transmission

To send an instruction frame, the user must first configure the address to send by writing the field ADDR in the Instruction Address register (QSPI_IAR). This step is required if the instruction frame includes an address and no data. When data is present, the address of the instruction is defined by the address of the data accesses in the QSPI memory space, not by QSPI_IAR.

If the instruction frame includes the instruction code and/or the option code, the user must configure the instruction code and/or the option code to send by writing the fields INST and OPT in the Instruction Code register (QSPI_ICR).

Then, the user must write QSPI_IFR to configure the instruction frame depending on which instruction must be sent. If the instruction frame does not include data, writing in this register triggers the send of the instruction frame in the QSPI. If the instruction frame includes data, the send of the instruction frame is triggered by the first data access in the QSPI memory space.

The instruction frame is configured by the following bits and fields of QSPI_IFR:

- WIDTH field—used to configure which data lanes are used to send the instruction code, the address, the option code and to transfer the data. It is possible to use two unidirectional data lanes (MISO-MOSI Single-bit SPI), two bidirectional data lanes (QIO0-QIO1 Dual SPI) or four bidirectional data lanes (QIO0–QIO3 Quad SPI).
- INSTEN bit—used to enable the send of an instruction code.
- ADDREN bit—used to enable the send of an address after the instruction code.
- OPTEN bit—used to enable the send of an option code after the address.
- DATAEN bit—used to enable the transfer of data (READ or PROGRAM instruction).
- OPTL field—used to configure the option code length. The value written in OPTL must be consistent with the value written in the field WIDTH. For example: OPTL = 0 (1-bit option code) is not consistent with WIDTH = 6 (option code sent with QuadSPI protocol, thus the minimum length of the option code is 4 bits).
- ADDRL bit—used to configure the address length.
- TFRTYP field—used to define which type of data transfer must be performed.
- DDREN bit—used to configure the double data rate mode, instruction code is still transmitted in single data rate mode.
- NBDUM field—used to configure the number of dummy cycles when reading data from the serial Flash memory. Between the address/option and the data, with some instructions, dummy cycles are inserted by the serial Flash memory.

Refer to Section 47.7.12 “QSPI Instruction Frame Register”.

If data transfer is enabled, the user can access the serial memory by reading or writing the QSPI memory space:

- To read in the serial memory, but not a memory data, for example a JEDEC-ID or the QSPI_SR, QSPI_IFR.TFRTYP must be written to ‘0’.
- To read in the serial memory, and particularly a memory data, TFRTYP must be written to ‘1’.
- To write in the serial memory, but not a memory data, for example writing the configuration or the QSPI_SR, TFRTYP must be written to ‘2’.
- If the user wants to write in the serial memory in particular to program a memory data, TFRTYP must be written to ‘3’ .

If QSPI_IFR.TFRTYP has a value other than ‘1’ and QSPI_MR.SMRM = 0, the address sent in the instruction frame is the address of the first system bus accesses. The addresses of the next accesses are not used by the QSPI. At each system bus access, an SPI transfer is performed with the same size. For example, a halfword system bus access leads to a 16-bit SPI transfer, and a byte system bus access leads to an 8-bit SPI transfer.
If SMRM = 1 and TFRTYP= (0 or 2), accesses are made via the QSPI registers and the address sent in the instruction frame is the address defined in QSPI_IAR.

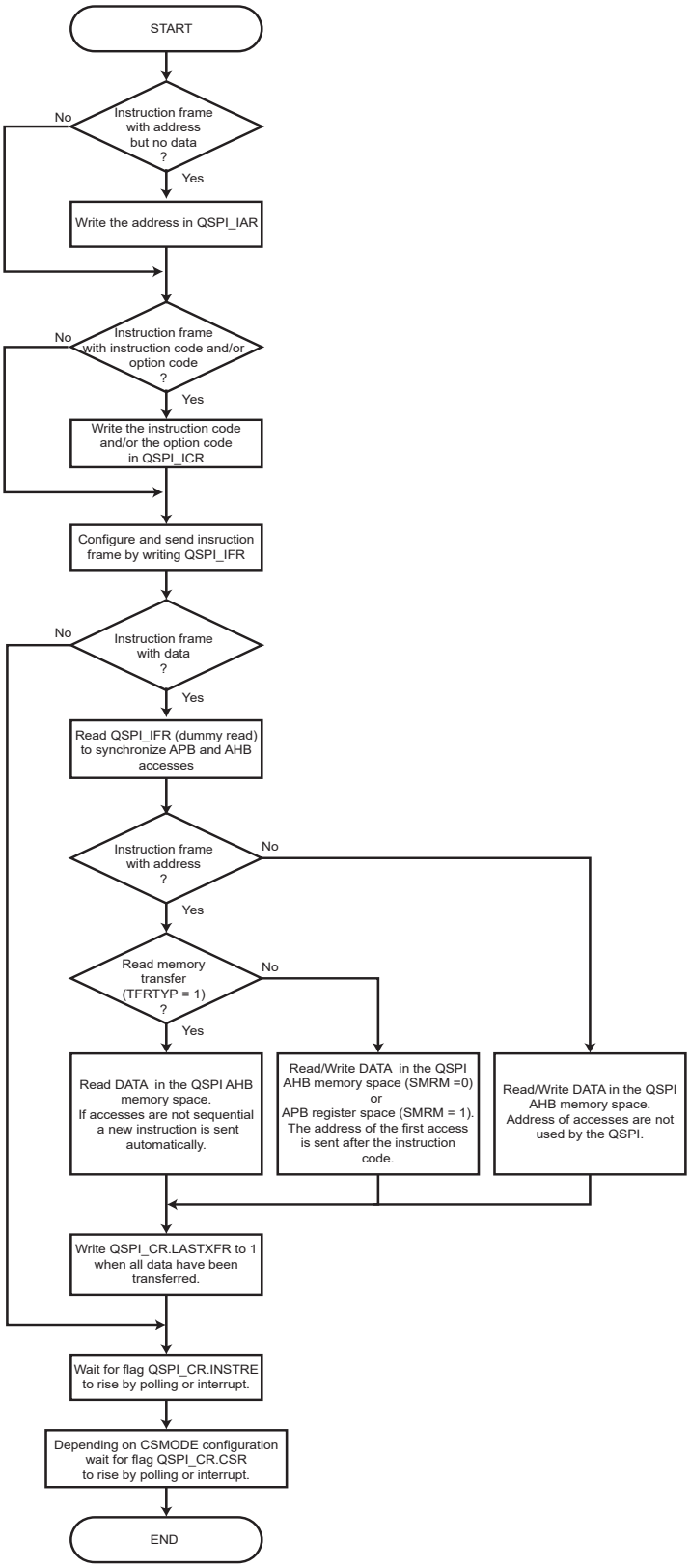
Each time QSPI_IFR is written (in case of read access), or each time QSPI_TDR is written (in case of write transfer), an SPI transfer is performed with a byte size. Another byte is read each time QSPI_RDR is read (flag RDRF shows when a data can be read in QSPI_RDR) or written each time QSPI_TDR is written (flag TDRE shows when a new data can be written). The SPI transfer ends by writing QSPI_CR.LASTXFER.

If TFRTYP = 1, the address of the first instruction frame is the one of the first read access in the QSPI memory space. Each time the read accesses become non-sequential (addresses are not consecutive), a new instruction frame is sent with the last system bus access address. In this way, the system can read data at a random location in the serial memory. The size of the SPI transfers may differ from the size of the system bus read accesses.

When data transfer is not enabled, the end of the instruction frame is indicated when QSPI_SR.INSTRE rises. (The QSPI_SR.CSR flag indicates when chip select rises. A delay between these flags may exist in case of high clock division or a high DLYBCT value). When data transfer is enabled, the user must indicate when the data transfer is completed in the QSPI memory space by setting QSPI_CR.LASTXFR. The end of the instruction frame is indicated when QSPI_SR.INSTRE rises.

Figure 47-9 illustrates instruction transmission management.

**Figure 47-9. Instruction Transmission Flow Diagram**



#### 47.6.5.3 Read Memory Transfer

The user can access the data of the serial memory by sending an instruction with QSPI_IFR.DATAEN = 1 and QSPI_IFR.TFRTYP = 1.

In this mode, the QSPI is able to read data at random address into the serial Flash memory, allowing the CPU to execute code directly from it (XIP execute-in-place).

In order to fetch data, the user must first configure the instruction frame by writing the QSPI_IFR. Then data can be read at any address in the QSPI address space mapping. The address of the system bus read accesses match the address of the data inside the serial Flash memory.

When Fetch mode is enabled, several instruction frames can be sent before writing QSPI_CR.LASTXFR. Each time the system bus read accesses become non- sequential (addresses are not consecutive), a new instruction frame is sent with the corresponding address.

#### 47.6.5.4 Continuous Read Mode

The QSPI is compatible with the Continuous Read mode which is implemented in some serial Flash memories.

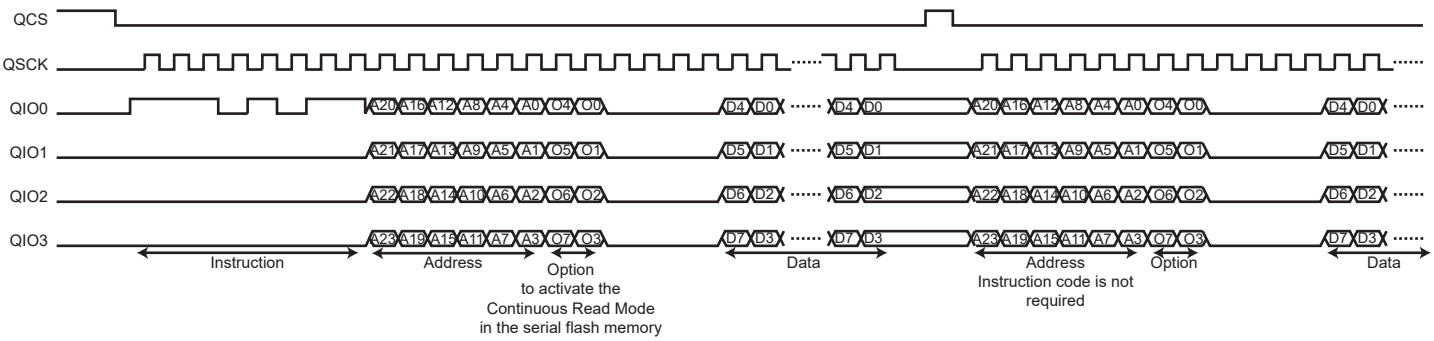
In Continuous Read mode, the instruction overhead is reduced by excluding the instruction code from the instruction frame. When the Continuous Read mode is activated in a serial Flash memory by a specific option code, the instruction code is stored in the memory. For the next instruction frames, the instruction code is not required as the memory uses the stored one.

In the QSPI, Continuous Read mode is used when reading data from the memory (QSPI_IFR.TFRTYP = 1). The addresses of the system bus read accesses are often non-sequential and this leads to many instruction frames that have the same instruction code. By disabling the send of the instruction code, the Continuous Read mode reduces the access time of the data.

To be functional, this mode must be enabled in both the QSPI and the serial Flash memory. The Continuous Read mode is enabled in the QSPI by writing CRM to ‘1’ in the QSPI_IFR (TFRTYP must equal 1). The Continuous Read mode is enabled in the serial Flash memory by sending a specific option code.

**CAUTION**: If the Continuous Read mode is not supported by the serial Flash memory or disabled, CRM bit must not be written to ‘1’, otherwise data read out the serial Flash memory is unpredictable.

**Figure 47-10. Continuous Read Mode**



### 47.6.5.5 Instruction Frame Transmission Examples

All waveforms in the following examples describe SPI transfers in SPI Clock mode 0 (QSPI_SCR.CPOL = 0 and QSPI_SCR.CPHA = 0; see Section 47.6.2 “Serial Clock Phase and Polarity”).

All system bus accesses described below refer to the system bus address phase. System bus wait cycles and system bus data phases are not shown.

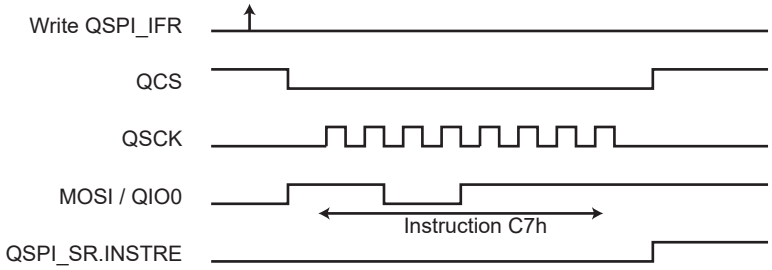
Example 1:

Instruction in Single-bit SPI, without address, without option, without data.

Command: CHIP ERASE (C7h).

- Write 0x0000_00C7 in QSPI_ICR.
- Write 0x0000_0010 in QSPI_IFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-11. Instruction Transmission Waveform 1**



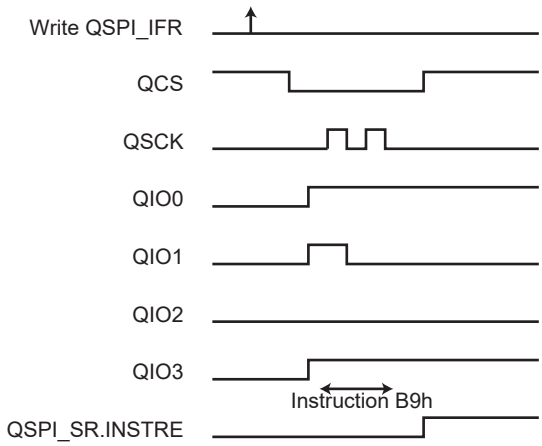

Example 2:

Instruction in Quad SPI, without address, without option, without data.

Command: POWER DOWN (B9h)

- Write 0x0000_00B9 in QSPI_ICR.
- Write 0x0000_0016 in QSPI_IFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-12. Instruction Transmission Waveform 2**



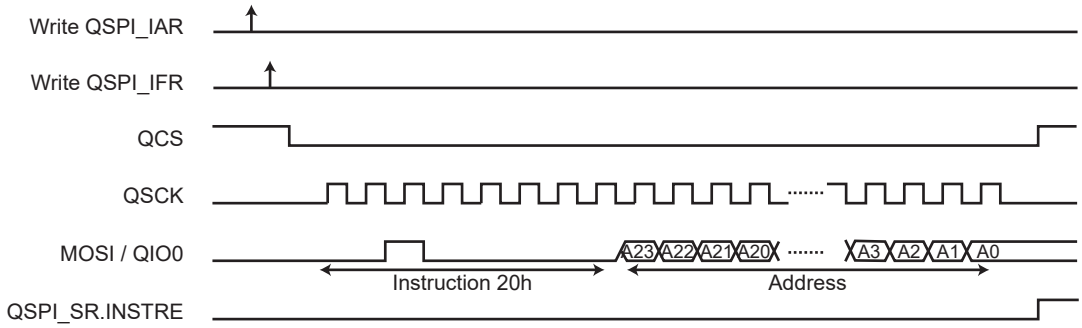
Example 3:

Instruction in Single-bit SPI, with address in Single-bit SPI, without option, without data.

Command: BLOCK ERASE (20h)

- Write the address (of the block to erase) in QSPI_AR.
- Write 0x0000_0020 in QSPI_ICR.
- Write 0x0000_0030 in QSPI_IFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-13. Instruction Transmission Waveform 3**



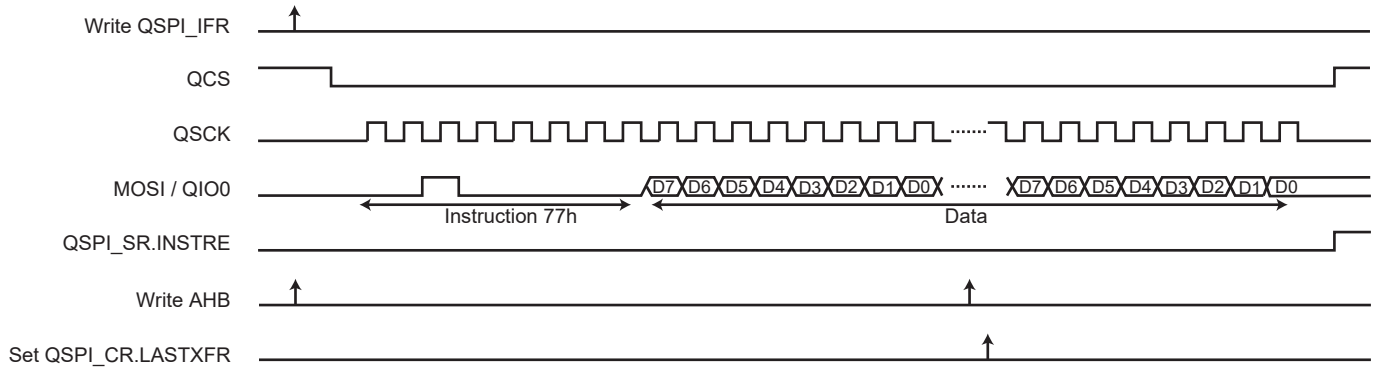

Example 4:

Instruction in Single-bit SPI, without address, without option, with data write in Single-bit SPI.

Command: SET BURST (77h)

- Write 0x0000_0077 in QSPI_ICR.
- Write 0x0000_2090 in QSPI_IFR.
- Read QSPI_IFR (dummy read) to synchronize system bus accesses.
- Write data in the system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  The address of system bus write accesses is not used.
- Write a ‘1’ to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-14. Instruction Transmission Waveform 4**



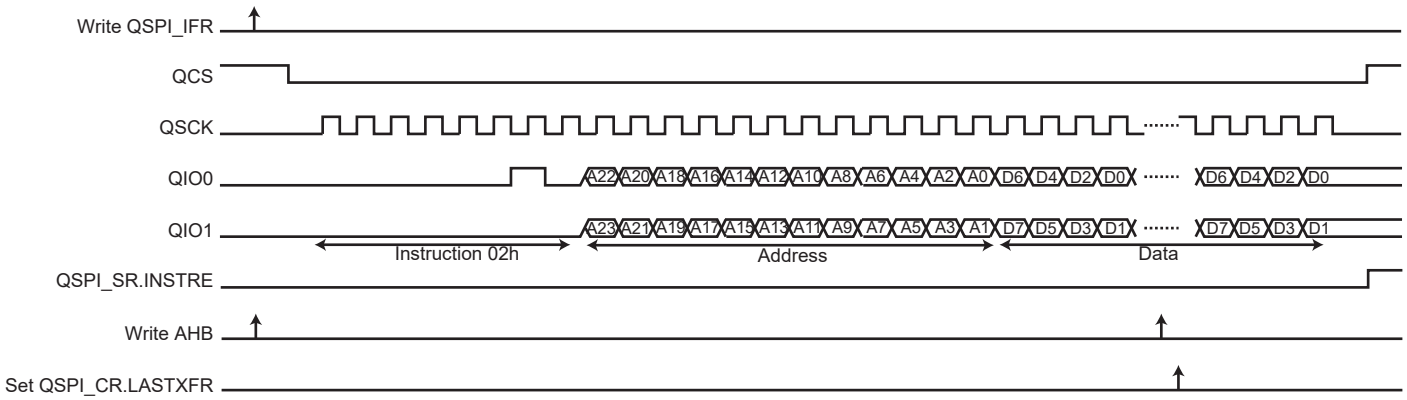
Example 5:

Instruction in Single-bit SPI, with address in Dual SPI, without option, with data write in Dual SPI.

Command: BYTE/PAGE PROGRAM (02h)

- Write 0x0000_0002 in QSPI_ICR.
- Write 0x0000_30B3 in QSPI_IFR.
- Read QSPI_IFR (dummy read) to synchronize system bus accesses.
- Write data in the QSPI system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  The address of the first system bus write access is sent in the instruction frame.
  The address of the next system bus write accesses is not used.
- Write a '1' to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-15. Instruction Transmission Waveform 5**



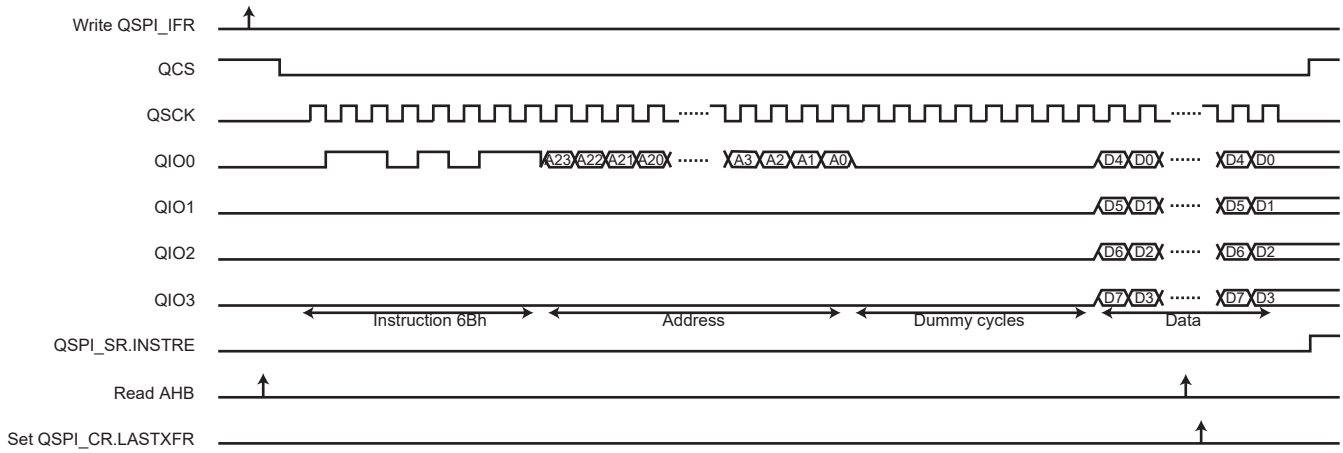
Example 6:

Instruction in Single-bit SPI, with address in Single-bit SPI, without option, with data read in Quad SPI, with eight dummy cycles.

Command: QUAD_OUTPUT READ ARRAY (6Bh)

- Write 0x0000_006B in QSPI_ICR.
- Write 0x0008_10B2 in QSPI_IFR.
- Read QSPI_IR (dummy read) to synchronize system bus accesses.
- Read data in the QSPI system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  The address of the first system bus read access is sent in the instruction frame.
  The address of the next system bus read accesses is not used.
- Write a '1' to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-16. Instruction Transmission Waveform 6**



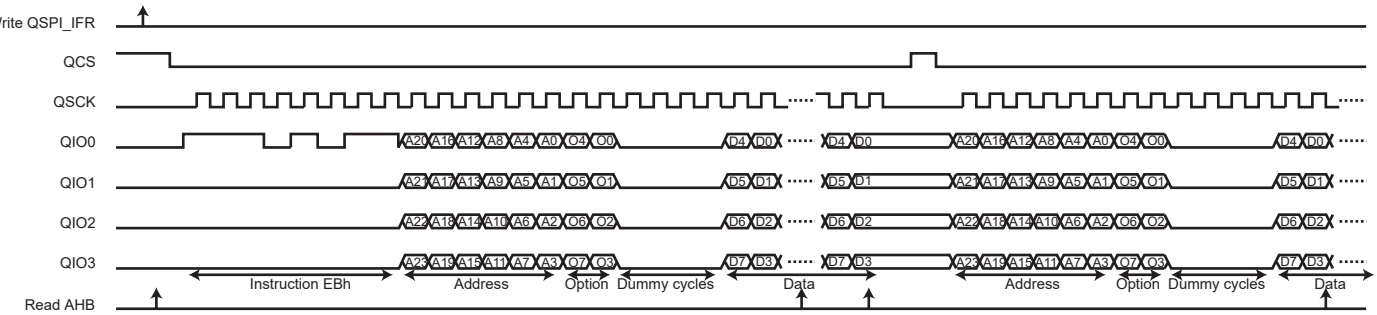
Example 7:

Instruction in Single-bit SPI, with address and option in Quad SPI, with data read in Quad SPI, with four dummy cycles, with fetch and continuous read.

Command: FAST READ QUAD I/O (EBh) - 8-BIT OPTION (0x30h)

- Write 0x0030_00EB in QSPI_ICR.
- Write 0x0004_33F4 in QSPI_IFR.
- Read QSPI_IFR (dummy read) to synchronize system bus accesses.
- Read data in the QSPI system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  Fetch is enabled, the address of the system bus read accesses is always used.
- Write a '1' to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-17. Instruction Transmission Waveform 7**



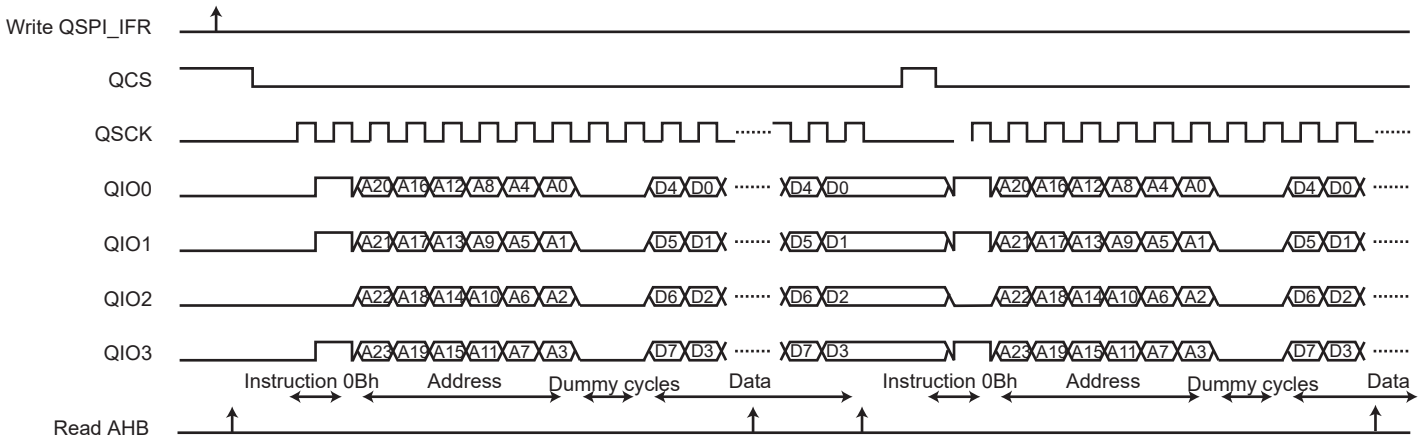
Example 8:

Instruction in Quad SPI, with address in Quad SPI, without option, with data read in Quad SPI, with two dummy cycles, with fetch.

Command: HIGH-SPEED READ (0Bh)

- Write 0x0000_000B in QSPI_ICR.
- Write 0x0002_20B6 in QSPI_IFR.
- Read QSPI_IFR (dummy read) to synchronize system bus accesses.
- Read data in the QSPI system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  Fetch is enabled, the address of the system bus read accesses is always used.
- Write a '1' to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-18. Instruction Transmission Waveform 8**



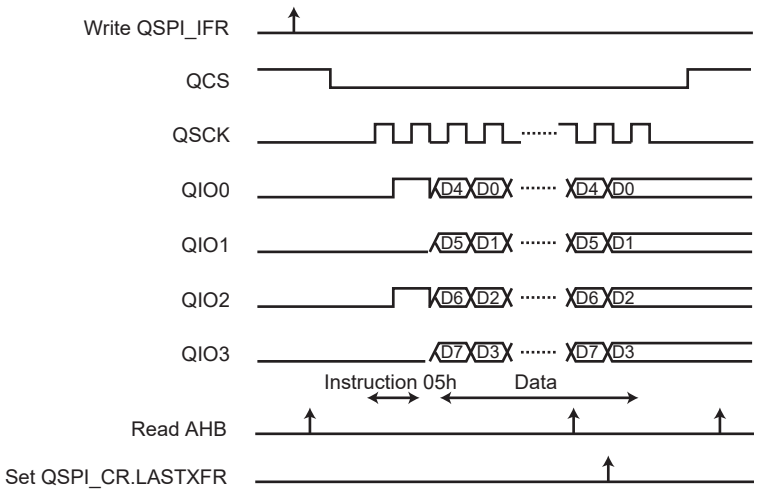
Example 9:

Instruction in Quad SPI, without address, without option, with data read in Quad SPI, without dummy cycles, without fetch.

Command: HIGH-SPEED READ (05h)

- Write 0x0000_0005 in QSPI_ICR.
- Write 0x0000_0096 in QSPI_IFR.
- Read QSPI_IFR (dummy read) to synchronize system bus accesses.
- Read data in the QSPI system bus memory space (0x9000_00000-0x9800_00000/0XD000_0000--0XD800_0000).
  Fetch is disabled.
- Write a '1' to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-19. Instruction Transmission Waveform 9**



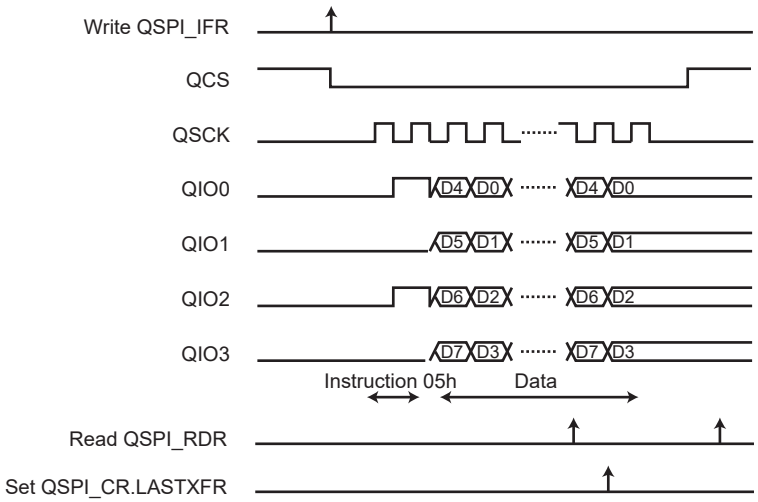
Example 10:

Instruction in Quad SPI, without address, without option, with data read in Quad SPI, without dummy cycles, without fetch, read launched through APB interface.

Command: HIGH-SPEED READ (05h)

- Set SMRM to ‘1’ in QSPI_MR
- Write 0x0000_0005 in QSPI_ICR.
- Write 0x0100_0096 in QSPI_IFR (will start the transfer).
- Wait flag RDRF and Read data in the QSPI_RDR register
  Fetch is disabled.
- Write a ‘1’ to QSPI_CR.LASTXFR.
- Wait for QSPI_SR.INSTRE to rise.

**Figure 47-20. Instruction Transmission Waveform 10**



### 47.6.6 Scrambling/Unscrambling Function

The scrambling/unscrambling function cannot be performed on devices other than memories. Data is scrambled when written to memory and unscrambled when data is read.

The external data lines can be scrambled in order to prevent intellectual property data located in off-chip memories from being easily recovered by analyzing data at the package pin level of either the microcontroller or the QSPI slave device (e.g., memory).

The scrambling/unscrambling function can be enabled by writing a '1' to the SCREN bit in the QSPI Scrambling Mode Register (QSPI_SMR).

The scrambling and unscrambling are performed on-the-fly without impacting the throughput.

The scrambling method depends on the user-configurable user scrambling key (field USRK) in the QSPI Scrambling Key Register (QSPI_SKR). QSPI_SKR is only accessible in Write mode.

If QSPI_SMR.RVDIS is written to '0', the scrambling/unscrambling algorithm includes the user scrambling key plus a random value depending on device processing characteristics. Data scrambled by a given microcontroller cannot be unscrambled by another.

If QSPI_SMR.RVDIS is written to '1', the scrambling/unscrambling algorithm includes only the user scrambling key. No random value is part of the key.

The user scrambling key or the seed for key generation must be securely stored in a reliable non-volatile memory in order to recover data from the off-chip memory. Any data scrambled with a given key cannot be recovered if the key is lost.

### 47.6.7 Register Write Protection

To prevent any single software error from corrupting QSPI behavior, certain registers in the address space can be write-protected by setting the WPEN bit in the QSPI Write Protection Mode Register (QSPI_WPMR).

If a write access to a write-protected register is detected, the WPVS flag in the QSPI Write Protection Status Register (QSPI_WPSR) is set and the field WPVSRC indicates the register in which the write access has been attempted.

The WPVS bit is automatically cleared after reading the QSPI_WPSR.

The following registers can be write-protected when WPEN is set in QSPI_WPMR:

- QSPI Mode Register
- QSPI Serial Clock Register
- QSPI Scrambling Mode Register
- QSPI Scrambling Key Register

## 47.7 Quad Serial Peripheral Interface (QSPI) User Interface

**Table 47-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Register | QSPI_CR | Write-only | – |
| 0x04 | Mode Register | QSPI_MR | Read/Write | 0x0 |
| 0x08 | Receive Data Register | QSPI_RDR | Read-only | 0x0 |
| 0x0C | Transmit Data Register | QSPI_TDR | Write-only | – |
| 0x10 | Status Register | QSPI_SR | Read-only | 0x0 |
| 0x14 | Interrupt Enable Register | QSPI_IER | Write-only | – |
| 0x18 | Interrupt Disable Register | QSPI_IDR | Write-only | – |
| 0x1C | Interrupt Mask Register | QSPI_IMR | Read-only | 0x0 |
| 0x20 | Serial Clock Register | QSPI_SCR | Read/Write | 0x0 |
| 0x30 | Instruction Address Register | QSPI_IAR | Read/Write | 0x0 |
| 0x34 | Instruction Code Register | QSPI_ICR | Read/Write | 0x0 |
| 0x38 | Instruction Frame Register | QSPI_IFR | Read/Write | 0x0 |
| 0x3C | Reserved | – | – | – |
| 0x40 | Scrambling Mode Register | QSPI_SMR | Read/Write | 0x0 |
| 0x44 | Scrambling Key Register | QSPI_SKR | Write-only | – |
| 0x48–0xE0 | Reserved | – | – | – |
| 0xE4 | Write Protection Mode Register | QSPI_WPMR | Read/Write | 0x0 |
| 0xE8 | Write Protection Status Register | QSPI_WPSR | Read-only | 0x0 |
| 0xEC–0xFC | Reserved | – | – | – |

### 47.7.1 QSPI Control Register

**Name:** QSPI_CR

**Address:** 0xF0020000 (0), 0xF0024000 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | LASTXFER |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWRST | – | – | – | – | – | QSPIDIS | QSPIEN |

- **QSPIEN: QSPI Enable**

0: No effect.

1: Enables the QSPI to transfer and receive data.

- **QSPIDIS: QSPI Disable**

0: No effect.

1: Disables the QSPI.

As soon as QSPIDIS is set, the QSPI finishes its transfer.

All pins are set in Input mode and no data is received or transmitted.

If a transfer is in progress, the transfer is finished before the QSPI is disabled.

If both QSPIEN and QSPIDIS are equal to one when QSPI_CR is written, the QSPI is disabled.

- **SWRST: QSPI Software Reset**

0: No effect.

1: Reset the QSPI. A software-triggered hardware reset of the QSPI interface is performed.

DMA channels are not affected by software reset.

- **LASTXFER: Last Transfer**

0: No effect.

1: The chip select is deasserted after the character written in QSPI_TDR.TD has been transferred.

### 47.7.2 QSPI Mode Register

**Name:** QSPI_MR

**Address:** 0xF0020004 (0), 0xF0024004 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DLYCS | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DLYBCT | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | NBBITS | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | CSMODE | | SMRM | WDRBT | LLB | SMM |

This register can only be written if bit WPEN is cleared in the QSPI Write Protection Mode Register.

- **SMM: Serial Memory Mode**

0 (SPI): The QSPI is in SPI mode.

1 (MEMORY): The QSPI is in Serial Memory mode.

- **LLB: Local Loopback Enable**

0 (DISABLED): Local loopback path disabled.

1 (ENABLED): Local loopback path enabled.

LLB controls the local loopback on the data serializer for testing in SPI mode only. (MISO is internally connected on MOSI).

- **WDRBT: Wait Data Read Before Transfer**

0 (DISABLED): No effect. In SPI mode, a transfer can be initiated whatever the state of the QSPI_RDR is.

1 (ENABLED): In SPI mode, a transfer can start only if the QSPI_RDR is empty, i.e., does not contain any unread data. This mode prevents overrun error in reception.

- **SMRM: Serial Memory Register Mode**

0: Serial Memory registers are written via AHB access. See Section 47.6.5.2 for details.

1: Serial Memory registers are written via APB access. See Section 47.6.5.2 for details.

- **CSMODE: Chip Select Mode**

The CSMODE field determines how the chip select is deasserted

Note: This field is forced to LASTXFER when SMM is written to ‘1’.

| Value | Name | Description |
|---|---|---|
| 0 | NOT_RELOADED | The chip select is deasserted if QSPI_TDR.TD has not been reloaded before the end of the current transfer. |
| 1 | LASTXFER | The chip select is deasserted when the bit LASTXFER is written to ‘1’ and the character written in QSPI_TDR.TD has been transferred. |
| 2 | SYSTEMATICALLY | The chip select is deasserted systematically after each transfer. |

- **NBBITS: Number Of Bits Per Transfer**

| Value | Name | Description |
|---|---|---|
| 0 | 8_BIT | 8 bits for transfer |
| 8 | 16_BIT | 16 bits for transfer |

- **DLYCS: Minimum Inactive QCS Delay**

This field defines the minimum delay between the deactivation and the activation of QCS. The DLYCS time guarantees the slave minimum deselect time.

If DLYCS written to ‘0’, one peripheral clock period is inserted by default.

Otherwise, the following equation determines the delay:

DLYCS = Minimum inactive $\times f_{\text{peripheral clock}}$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT is written to ‘0’, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers. In Serial Memory mode (SMM = 1), DLYBCT must be written to ‘0’ and no delay is inserted.

Otherwise, the following equation determines the delay:

DLYBCT = (Delay Between Consecutive Transfers $\times f_{\text{peripheral clock}}$) / 32

### 47.7.3 QSPI Receive Data Register

**Name:** QSPI_RDR

**Address:** 0xF0020008 (0), 0xF0024008 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RD | | | | | | | |

- **RD: Receive Data**

Data received by the QSPI is stored in this register right-justified. Unused bits read zero.

### 47.7.4 QSPI Transmit Data Register

**Name:** QSPI_TDR

**Address:** 0xF002000C (0), 0xF002400C (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TD | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TD | | | | | | | |

- **TD: Transmit Data**

Data to be transmitted by the QSPI is stored in this register. Information to be transmitted must be written to the Transmit Data register in a right-justified format.

### 47.7.5 QSPI Status Register

**Name:** QSPI_SR

**Address:** 0xF0020010 (0), 0xF0024010 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | QSPIENS |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | INSTRE | CSS | CSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | TXEMPTY | TDRE | RDRF |

- **RDRF: Receive Data Register Full (cleared by reading QSPI_RDR)**

0: No data has been received since the last read of QSPI_RDR.

1: Data has been received and the received data has been transferred from the serializer to QSPI_RDR since the last read of QSPI_RDR.

- **TDRE: Transmit Data Register Empty (cleared by writing QSPI_TDR)**

0: Data has been written to QSPI_TDR and not yet transferred to the serializer.

1: The last data written in the QSPI_TDR has been transferred to the serializer.

TDRE equals zero when the QSPI is disabled or at reset. The QSPI enable command sets this bit to one.

- **TXEMPTY: Transmission Registers Empty (cleared by writing QSPI_TDR)**

0: As soon as data is written in QSPI_TDR.

1: QSPI_TDR and the internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion of such delay.

- **OVRES: Overrun Error Status (cleared on read)**

0: No overrun has been detected since the last read of QSPI_SR.

1: At least one overrun error has occurred since the last read of QSPI_SR.

An overrun occurs when QSPI_RDR is loaded at least twice from the serializer since the last read of the QSPI_RDR.

- **CSR: Chip Select Rise (cleared on read)**

0: No chip select rise has been detected since the last read of QSPI_SR.

1: At least one chip select rise has been detected since the last read of QSPI_SR.

- **CSS: Chip Select Status**

0: The chip select is asserted.

1: The chip select is not asserted.

- **INSTRE: Instruction End Status (cleared on read)**

0: No instruction end has been detected since the last read of QSPI_SR.

1: At least one instruction end has been detected since the last read of QSPI_SR.

- **QSPIENS: QSPI Enable Status**

0: QSPI is disabled.

1: QSPI is enabled.

### 47.7.6 QSPI Interrupt Enable Register

**Name:** QSPI_IER

**Address:** 0xF0020014 (0), 0xF0024014 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | INSTRE | CSS | CSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | TXEMPTY | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Enables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: Transmit Data Register Empty Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **CSR: Chip Select Rise Interrupt Enable**
- **CSS: Chip Select Status Interrupt Enable**
- **INSTRE: Instruction End Interrupt Enable**

### 47.7.7 QSPI Interrupt Disable Register

**Name:** QSPI_IDR

**Address:** 0xF0020018 (0), 0xF0024018 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | INSTRE | CSS | CSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | TXEMPTY | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: No effect.

1: Disables the corresponding interrupt.

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: Transmit Data Register Empty Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **CSR: Chip Select Rise Interrupt Disable**
- **CSS: Chip Select Status Interrupt Disable**
- **INSTRE: Instruction End Interrupt Disable**

### 47.7.8 QSPI Interrupt Mask Register

**Name:** QSPI_IMR

**Address:** 0xF002001C (0), 0xF002401C (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | INSTRE | CSS | CSR |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | OVRES | TXEMPTY | TDRE | RDRF |

The following configuration values are valid for all listed bit names of this register:

0: The corresponding interrupt is not enabled.

1: The corresponding interrupt is enabled.

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: Transmit Data Register Empty Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **CSR: Chip Select Rise Interrupt Mask**
- **CSS: Chip Select Status Interrupt Mask**
- **INSTRE: Instruction End Interrupt Mask**

### 47.7.9 QSPI Serial Clock Register

**Name:** QSPI_SCR

**Address:** 0xF0020020 (0), 0xF0024020 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DLYBS | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| SCBR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | CPHA | CPOL |

This register can only be written if bit WPEN is cleared in the QSPI Write Protection Mode Register.

- **CPOL: Clock Polarity**

0: The inactive state value of QSCK is logic level zero.

1: The inactive state value of QSCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (QSCK). It is used with CPHA to produce the required clock/data relationship between master and slave devices.

- **CPHA: Clock Phase**

0: Data is captured on the leading edge of QSCK and changed on the following edge of QSCK.

1: Data is changed on the leading edge of QSCK and captured on the following edge of QSCK.

CPHA determines which edge of QSCK causes data to change and which edge causes data to be captured. CPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **SCBR: Serial Clock Baud Rate**

The QSPI uses a modulus counter to derive the QSCK baud rate from the peripheral clock. The baud rate is selected by writing a value from 0 to 255 in the SCBR field. The following equation determines the QSCK baud rate:

SCBR = ($f_{peripheral\ clock}$ / QSCK Baud rate) - 1

- **DLYBS: Delay Before QSCK**

This field defines the delay from QCS valid to the first valid QSCK transition.

When DLYBS equals zero, the QCS valid to QSCK transition is 1/2 the QSCK clock period.

Otherwise, the following equation determines the delay:

DLYBS = Delay Before QSCK $\times$ $f_{peripheral\ clock}$

### 47.7.10 QSPI Instruction Address Register

**Name:** QSPI_IAR

**Address:** 0xF0020030 (0), 0xF0024030 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ADDR | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| ADDR | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ADDR | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADDR | | | | | | | |

- **ADDR: Address**

Address to send to the serial Flash memory in the instruction frame.

### 47.7.11 QSPI Instruction Code Register

**Name:** QSPI_ICR

**Address:** 0xF0020034 (0), 0xF0024034 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| OPT | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| INST | | | | | | | |

- **INST: Instruction Code**

Instruction code to send to the serial Flash memory.

- **OPT: Option Code**

Option code to send to the serial Flash memory.

### 47.7.12 QSPI Instruction Frame Register

**Name:** QSPI_IFR

**Address:** 0xF0020038 (0), 0xF0024038 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | NBDUM | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | CRM | TFRTYP | | – | ADDRL | OPTL | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DATAEN | OPTEN | ADDREN | INSTEN | – | WIDTH | | |

- **WIDTH: Width of Instruction Code, Address, Option Code and Data**

| Value | Name | Description |
|---|---|---|
| 0 | SINGLE_BIT_SPI | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Single-bit SPI |
| 1 | DUAL_OUTPUT | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Dual SPI |
| 2 | QUAD_OUTPUT | Instruction: Single-bit SPI / Address-Option: Single-bit SPI / Data: Quad SPI |
| 3 | DUAL_IO | Instruction: Single-bit SPI / Address-Option: Dual SPI / Data: Dual SPI |
| 4 | QUAD_IO | Instruction: Single-bit SPI / Address-Option: Quad SPI / Data: Quad SPI |
| 5 | DUAL_CMD | Instruction: Dual SPI / Address-Option: Dual SPI / Data: Dual SPI |
| 6 | QUAD_CMD | Instruction: Quad SPI / Address-Option: Quad SPI / Data: Quad SPI |

- **INSTEN: Instruction Enable**

0: The instruction is not sent to the serial Flash memory.

1: The instruction is sent to the serial Flash memory.

- **ADDREN: Address Enable**

0: The transfer address is not sent to the serial Flash memory.

1: The transfer address is sent to the serial Flash memory.

- **OPTEN: Option Enable**

0: The option is not sent to the serial Flash memory.

1: The option is sent to the serial Flash memory.

- **DATAEN: Data Enable**

0: No data is sent/received to/from the serial Flash memory.

1: Data is sent/received to/from the serial Flash memory.

- **OPTL: Option Code Length**

The OPTL field determines the length of the option code. The value written in OPTL must be coherent with value written in the field WIDTH. For example: OPTL = 0 (1-bit option code) is not coherent with WIDTH = 6 (option code sent with Quad-SPI protocol, thus the minimum length of the option code is 4 bits).

| Value | Name | Description |
|---|---|---|
| 0 | OPTION_1BIT | The option code is 1 bit long. |
| 1 | OPTION_2BIT | The option code is 2 bits long. |
| 2 | OPTION_4BIT | The option code is 4 bits long. |
| 3 | OPTION_8BIT | The option code is 8 bits long. |

- **ADDRL: Address Length**

The ADDRL bit determines the length of the address.

0 (24_BIT): The address is 24 bits long.

1 (32_BIT): The address is 32 bits long.

- **TFRTYP: Data Transfer Type**

| Value | Name | Description |
|---|---|---|
| 0 | TRSFR_READ | Read transfer from the serial memory.<br>Scrambling is not performed.<br>Read at random location (fetch) in the serial Flash memory is not possible. |
| 1 | TRSFR_READ_MEMORY | Read data transfer from the serial memory.<br>If enabled, scrambling is performed.<br>Read at random location (fetch) in the serial Flash memory is possible. |
| 2 | TRSFR_WRITE | Write transfer into the serial memory.<br>Scrambling is not performed. |
| 3 | TRSFR_WRITE_MEMORY | Write data transfer into the serial memory.<br>If enabled, scrambling is performed. |

- **CRM: Continuous Read Mode**

0 (DISABLED): The Continuous Read mode is disabled.

1 (ENABLED): The Continuous Read mode is enabled.

- **NBDUM: Number Of Dummy Cycles**

The NBDUM field defines the number of dummy cycles required by the serial Flash memory before data transfer.

### 47.7.13 QSPI Scrambling Mode Register

**Name:** QSPI_SMR

**Address:** 0xF0020040 (0), 0xF0024040 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | RVDIS | SCREN |

This register can only be written if bit WPEN is cleared in the QSPI Write Protection Mode Register.

- **SCREN: Scrambling/Unscrambling Enable**

0 (DISABLED): The scrambling/unscrambling is disabled.

1 (ENABLED): The scrambling/unscrambling is enabled.

- **RVDIS: Scrambling/Unscrambling Random Value Disable**

0: The scrambling/unscrambling algorithm includes the user scrambling key plus a random value that may differ between devices.

1: The scrambling/unscrambling algorithm includes only the user scrambling key.

### 47.7.14 QSPI Scrambling Key Register

**Name:** QSPI_SKR

**Address:** 0xF0020044 (0), 0xF0024044 (1)

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| USRK | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| USRK | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| USRK | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| USRK | | | | | | | |

This register can only be written if bit WPEN is cleared in the QSPI Write Protection Mode Register.

- **USRK: User Scrambling Key**

### 47.7.15 QSPI Write Protection Mode Register

**Name:** QSPI_WPMR

**Address:** 0xF00200E4 (0), 0xF00240E4 (1)

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WPKEY | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| WPKEY | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPKEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPEN |

- **WPEN: Write Protection Enable**

0: Disables the write protection if WPKEY corresponds to 0x515350 (QSP in ASCII)

1: Enables the write protection if WPKEY corresponds to 0x515350 (QSP in ASCII)

See Section 47.6.7 “Register Write Protection” for the list of registers that can be protected.

- **WPKEY: Write Protection Key**

| Value | Name | Description |
|---|---|---|
| 0x515350 | PASSWD | Writing any other value in this field aborts the write operation of the WPEN bit. Always reads as 0. |

### 47.7.16 QSPI Write Protection Status Register

**Name:** QSPI_WPSR

**Address:** 0xF00200E8 (0), 0xF00240E8 (1)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WPVSRC | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | WPVS |

- **WPVS: Write Protection Violation Status**

0: No write protection violation has occurred since the last read of the QSPI_WPSR.

1: A write protection violation has occurred since the last read of the QSPI_WPSR. If this violation is an unauthorized attempt to write a protected register, the associated violation is reported into field WPVSRC.

- **WPVSRC: Write Protection Violation Source**

When WPVS = 1, WPVSRC indicates the register address offset at which a write access has been attempted.

# 48. Secure Digital Multimedia Card Controller (SDMMC)

## 48.1 Description

The Secure Digital Multimedia Card Controller (SDMMC) supports the embedded MultiMedia Card (e.MMC) Specification V4.51, the SD Memory Card Specification V3.0, and the SDIO V3.0 specification. It is compliant with the SD Host Controller Standard V3.0 specification.

The SDMMC includes the register set defined in the "SD Host Controller Simplified Specification V3.00" and additional registers to manage e.MMC devices, sampling tuning procedure, PAD calibration and enhanced features.

The SDMMC is clocked by three asynchronous clocks (see Section 48.5 "Block Diagram") and requires the PMC to be configured first.

## 48.2 Embedded Characteristics

- Compatible with SD Host Controller Standard Specification Version 3.00
- Compatible with MultiMedia Card Specification Version 4.51
- Compatible with SD Memory Card Specification Version 3.00
- Compatible with SDIO Specification Version 3.00
- Support for 1-bit/ 4-bit SD/SDIO Devices
- Support for 1-bit/4-bit/8-bit e.MMC Devices
- Support for SD/SDIO Default Speed (Maximum SDCLK Frequency = 25 MHz)
- Support for SD/SDIO High Speed (Maximum SDCLK Frequency = 50 MHz)
- Support for SD/SDIO UHS-I SDR12 (Maximum SDCLK Frequency = 25 MHz)
- Support for SD/SDIO UHS-I SDR25 (Maximum SDCLK Frequency = 50 MHz)
- Support for SD/SDIO UHS-I SDR50 (Maximum SDCLK Frequency = 100 MHz)
- Support for SD/SDIO UHS-I SDR104 (Maximum SDCLK Frequency = 120 MHz)
- Support for SD/SDIO UHS-I DDR50 (Maximum SDCLK Frequency = 50 MHz)
- Support for SDSC, SDHC and SDXC
- Support for e.MMC Default Speed (Maximum SDCLK Frequency = 26 MHz)
- Support for e.MMC High Speed (Maximum SDCLK Frequency = 52 MHz)
- Support for e.MMC High Speed DDR (Maximum SDCLK Frequency = 52 MHz)
- Support for e.MMC HS200 (Maximum SDCLK Frequency = 120 MHz)
- e.MMC Boot Operation Mode Support
- Support for Block Size from 1 to 512 bytes
- Support for Stream, Block and Multi-block Data Read and Write
  - Advanced DMA and SDMA Capability
- Internal 1024-byte Dual Port RAM
- Support for both synchronous and asynchronous abort
- Supports for SDIO Card Interrupt

## 48.3 Embedded Features for SDMMC0 and SDMMC1

The device embeds two SDMMC interfaces; both do not support the same features.

SDMMC0 supports all the features listed in Section 48.2 “Embedded Characteristics”.

SDMMC1 is compatible with SD Memory Card Specification Version 3.00 and does **not** support the following features:

- 8-bit e.MMC Devices
- SD/SDIO UHS-I SDR12 (Maximum SDCLK Frequency = 25 MHz)
- SD/SDIO UHS-I SDR25 (Maximum SDCLK Frequency = 50 MHz)
- SD/SDIO UHS-I SDR50 (Maximum SDCLK Frequency = 100 MHz)
- SD/SDIO UHS-I SDR104 (Maximum SDCLK Frequency = 120 MHz)
- SD/SDIO UHS-I DDR50 (Maximum SDCLK Frequency = 50 MHz)
- e.MMC HS200 (Maximum SDCLK Frequency = 120 MHz)

In addition, the SDMMC1 pin interface does not embed SDMMC_1V8SEL and the SDMMC_DAT is limited to SDMMC_DAT[3..0].

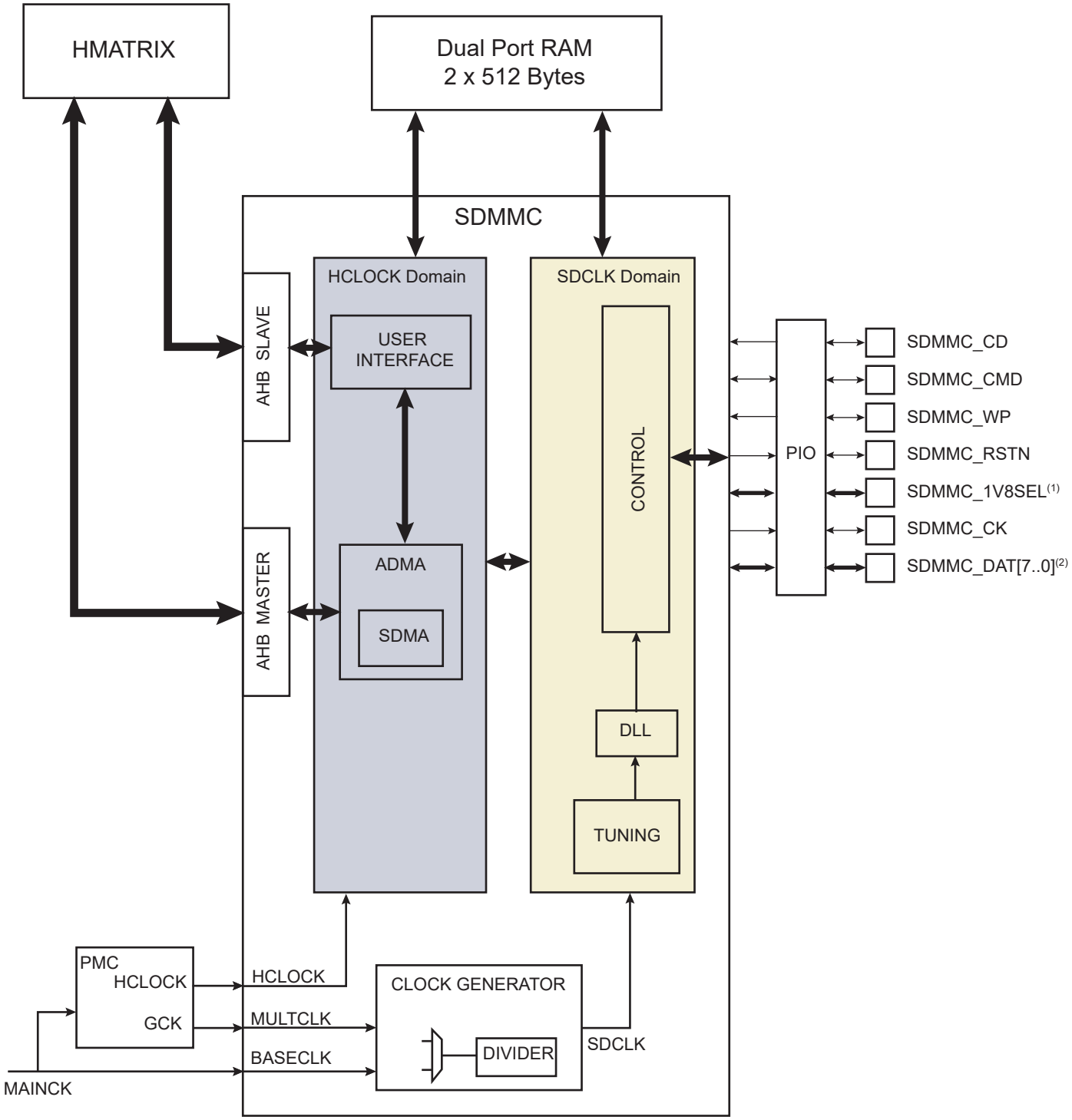
## 48.4 Reference Documents

**Table 48-1. Reference Documents**

| Name | Link |
|---|---|
| SD Host Controller Simplified Specification V3.00 | https://www.sdcard.org |
| SDIO Simplified Specification V3.00 | |
| Physical Layer Simplified Specification V3.01 | |
| Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51 | http://www.jedec.org |

## 48.5 Block Diagram

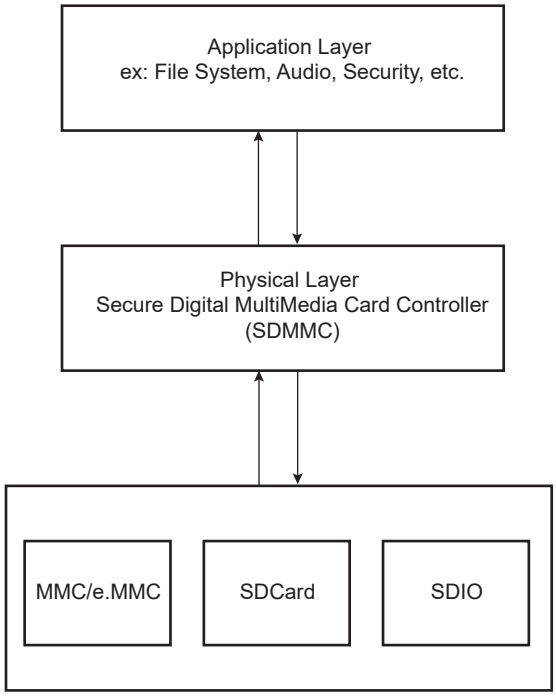
**Figure 48-1. Block Diagram**




Notes: 1. SDMMC_1V8SEL not available in SDMMC1
2. Limited to SDMMC_DAT[3..0] in SDMMC1

## 48.6 Application Block Diagram

**Figure 48-2. Application Block Diagram**




## 48.7 Pin Name List

**Table 48-2. I/O Lines Description for 8-bit Configuration**

| Pin Name[1] | Pin Description | Type |
|---|---|---|
| SDMMC_CD | SDcard / SDIO / e.MMC Card Detect | Input |
| SDMMC_CMD | SDcard / SDIO / e.MMC Command/Response Line | I/O |
| SDMMC_WP | SDcard Connector Write Protect Signal | Input |
| SDMMC_RSTN | e.MMC Reset Signal | Output |
| SDMMC_1V8SEL | SDcard Signal Voltage Selection | Output |
| SDMMC_CK | SDcard / SDIO / e.MMC Clock Signal | Output |
| SDMMC_DAT[7..0] | SDcard / SDIO / e.MMC Data Lines | I/O |

Notes: 1. When several SDMMCs are embedded in a product, SDMMC_CK refers to SDMMCx_CK, SDMMC_CMD to SDMMCx_CMD, SDMMC_DATy to SDMMCx_DATy, SDMMC_WP to SDMMCx_WP, SDMMC_1V8SEL to SDMMCx_1V8SEL, SDMMC_CD to SDMMCx_CD and SDMMC_RSTN to SDMMCx_RSTN.

## 48.8 Product Dependencies

### 48.8.1 I/O Lines

The pins used for interfacing the Secure Digital MultiMedia Card (SDMMC) Controller are multiplexed with PIO lines. The programmer must first program the PIO controller to assign the peripheral functions to SDMMC pins.

### 48.8.2 Power Management

The SDMMC is clocked through the Power Management Controller (PMC), so the programmer must first configure the PMC to enable the SDMMC clocks.

### 48.8.3 Interrupt Sources

The SDMMC has an interrupt line connected to the interrupt controller.

Handling the SDMMC interrupt requires programming the interrupt controller before configuring the SDMMC.

## 48.9 SD/SDIO Operating Mode

The SDMMC is fully compliant with the “SD Host Controller Simplified Specification V3.00” for SD/SDIO devices. Refer to this specification for SDMMC configuration.

Refer to “Physical Layer Simplified Specification V3.01” and “SDIO Simplified Specification V3.00” for SD/SDIO management.

## 48.10 e.MMC Operating Mode

The SDMMC supports e.MMC devices management. As the “SD Host Controller Simplified Specification V3.00” does not apply to e.MMC devices, some registers have been added to those described in this specification in order to manage e.MMC devices. Most of the registers described in the “SD Host Controller Simplified Specification V3.00” must be used for e.MMC management, but e.MMC-specific features are managed using SDMMC_MC1R and SDMMC_MC2R.

### 48.10.1 Boot Operation Mode

In Boot Operation mode, the processor can read boot data from the e.MMC device by keeping the CMD line low after power-on before issuing the CMD1. The data can be read from either one of the boot partitions or the user area according to BOOT_PARTITION_ENABLE in the Extended CSD register (see “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”).

#### 48.10.1.1 Boot Procedure, Processor Mode

1. Configure SDMMC:
    1. Set the data bus width using SDMMC_HC1R.DW and SDMMC_HC1R.EXTDW according to BOOT_BUS_WIDTH in the Extended CSD Register (see “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”).
    2. Select the speed mode (using SDMMC_HC1R.HSEN or SDMMC_MC1R.DDR) according to BOOT_MODE in the Extended CSD Register.
    3. Set the SDCLK frequency according to the selected speed mode.
    4. If the Boot Acknowledge is sent by the e.MMC device (BOOT_ACK = 1 in the Extended CSD Register), set the Boot Acknowledge Enable to 1 (SDMMC_MC1R.BOOTA = 1).
    5. Enable interrupt on Boot Acknowledge Received (SDMMC_NISTER.BOOTAR = 1 and SDMMC_NISIER.BOOTAR = 1).
    6. Set the e.MMC Command Type to BOOT (SDMMC_MC1R.CMDTYP = 3)

7. Set SDMMC_TMR to read multiple blocks for the e.MMC device (SDMMC_TMR.MSBSEL = 1 and SDMMC_TMR.DTDSEL = 1).
8. Select the Non-DMA transfer (SDMMC_TMR.DMAEN = 0).
9. Optional: select the Auto CMD method (using SDMMC_TMR.ACMDEN).
10. Set the block size to 512 bytes (SDMMC_BSR.BLKSIZE = 512).
11. Set the required number of read blocks (using SDMMC_BCR.BLKCNT). SDMMC_TMR.BCEN must be set to 1.

2. Write SDMMC_CR = 20(hexa) to set the e.MMC in Boot Operation mode.
3. Wait for interrupt on Boot Acknowledge Received (BOOTAR).
4. The user can copy the boot data sequentially as soon as the BRDRDY flag is asserted.
5. When the data transfer is completed, the boot operation must be terminated by setting SDMMC_MC2R.ABOOT to 1.

#### 48.10.1.2 Boot Procedure, SDMA Mode

1. Configure SDMMC:
    1. Set the data bus width using SDMMC_HC1R.DW and SDMMC_HC1R.EXTDW according to BOOT_BUS_WIDTH in the Extended CSD Register (see "Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51").
    2. Select the speed mode (SDMMC_HC1R.HSEN or SDMMC_MC1R.DDR) according to BOOT_MODE in the Extended CSD Register .
    3. Set the SDCLK frequency according to the selected speed mode.
    4. If the Boot Acknowledge is sent by the e.MMC device (BOOT_ACK = 1 in the Extended CSD Register), set the Boot Acknowledge Enable to 1 (SDMMC_MC1R.BOOTA = 1).
    5. Enable interrupt on Boot Acknowledge Received (SDMMC_NISTER.BOOTAR = 1 and SDMMC_NISIER.BOOTAR = 1).
    6. Set the e.MMC Command Type to BOOT (SDMMC_MC1R.CMDTYP = 3).
    7. Set SDMMC_TMR to read multiple blocks for the e.MMC device (SDMMC_TMR.MSBSEL = 1 and SDMMC_TMR.TDSEL = 1).
    8. Select the SDMA transfer (SDMMC_TMR.DMAEN = 1 and SDMMC_HC1R.DMASEL = 0).
    9. Write the SDMA system address where the boot data will be copied (SDMMC_SSAR.ADDR).
    10. Optional: select the Auto CMD method (SDMMC_TMR.ACMDEN).
    Note: Auto CMD23 cannot be used with SDMA.
    11. Set the block size to 512 bytes (SDMMC_BSR.BLKSIZE = 512).
    12. Set the required number of read blocks (SDMMC_BCR.BLKCNT). SDMMC_TMR.BCEN must be set to 1.
2. Write SDMMC_CR = 20(hexa) to set the e.MMC in Boot Operation mode.
3. Wait for interrupt on Boot Acknowledge Received (BOOTAR).
4. The user can copy the boot data sequentially as soon as the BRDRDY flag is asserted.
5. When the data transfer is completed, the boot operation must be terminated by setting SDMMC_MC2R.ABOOT to 1.

#### 48.10.1.3 Boot Procedure, ADMA Mode

1. Configure SDMMC:
    1. Set the data bus width using SDMMC_HC1R.DW and SDMMC_HC1R.EXTDW according to BOOT_BUS_WIDTH in the Extended CSD Register (see "Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51").
    2. Select the speed mode (SDMMC_HC1R.HSEN or SDMMC_MC1R.DDR) according to BOOT_MODE in the Extended CSD Register .
    3. Set the SDCLK frequency according to the selected speed mode.

4. If the Boot Acknowledge is sent by the e.MMC device (BOOT_ACK = 1 in the Extended CSD Register), set the Boot Acknowledge Enable to 1 (SDMMC_MC1R.BOOTA = 1).
5. Enable interrupt on Boot Acknowledge Received (SDMMC_NISTER.BOOTAR = 1 and SDMMC_NISIER.BOOTAR = 1).
6. Set the e.MMC Command Type to BOOT (SDMMC_MC1R.CMDTYP = 3).
7. Set SDMMC_TMR to read multiple blocks for the e.MMC device (SDMMC_TMR.MSBSEL = 1 and SDMMC_TMR.DTDSEL = 1).
8. Select the ADMA transfer (SDMMC_TMR.DMAEN = 1 and SDMMC_HC1R.DMASEL = 2 or 3).
9. Write the address of the descriptor table in the ADMA system address (SDMMC_ASARx [0..1].ADMASA).
10. Optional: select the Auto CMD method (SDMMC_TMR.ACMDEN).
    Note: Auto CMD23 cannot be used with SDMA.
11. Set the block size to 512 bytes (SDMMC_BSR.BLKSIZE = 512).
12. Set the required number of read blocks (SDMMC_BCR.BLKCNT). SDMMC_TMR.BCEN must be set to 1.

2. Write SDMMC_CR = 20(hexa) to set the e.MMC in Boot Operation Mode.
3. Wait for interrupt on Boot Acknowledge Received (BOOTAR).
4. The user can copy the boot data sequentially as soon as the BRDRDY flag is asserted.
5. When the data transfer is completed, the boot operation must be terminated by setting SDMMC_MC2R.ABOOT to 1.

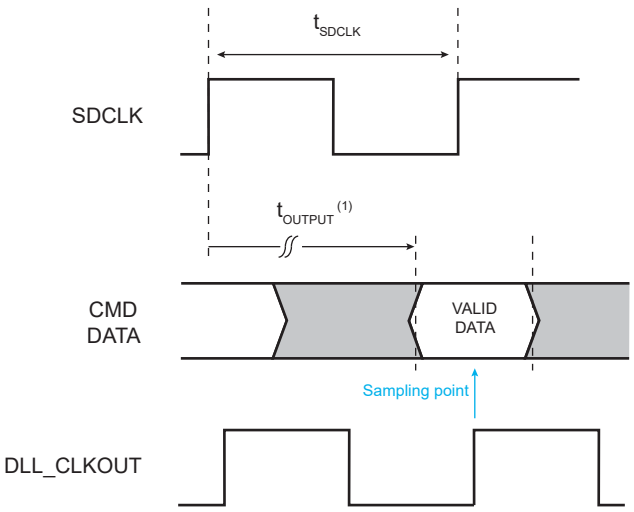
## 48.11 SDR104 / HS200 Tuning

### 48.11.1 DLL and Sampling Point

In SD/SDIO SDR104 mode (VS18EN = 1 and UHSMS = 3 in SDMMC_HC2R) or e.MMC HS200 mode (HS200EN = $B_{(hexa)}$), a tuning procedure must be performed first in order to adjust the sampling point for read transactions. For more details regarding the basic tuning procedure, refer to “Sampling Clock Tuning Procedure” in the “SD Host Controller Simplified Specification V3.00”.

As the position of data and command coming from the device varies, a DLL is used to generate an accurate sampling point (DLL_CLKOUT)(see Figure 48-3).

**Figure 48-3. DLL Sampling Point**

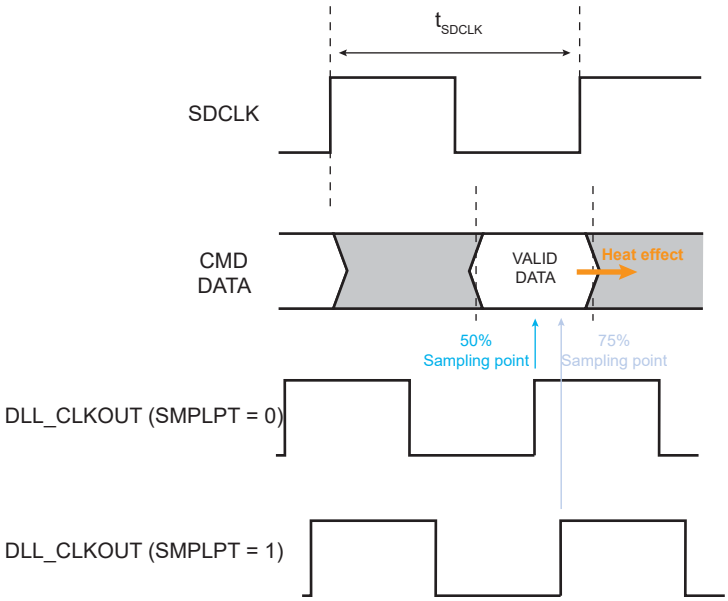



Note: 1. $t_{OUTPUT}$ varies from 0 to 2 x $t_{SDCLK}$

The minimum SDLCK frequency is 100 MHz when SD/SDIO SDR104 or e.MMC HS200 is selected.

The sampling point can be selected to be located at 50% or 75% of the data window to anticipate the effect of the temperature rise. If the SMPLPT bit is cleared in SDMMC_TUNCR, the sampling point is centered (50% of the data window). If the SMPLPT bit is set to 1 in SDMMC_TUNCR, the sampling point is set at 75% of the data window (see Figure 48-4).

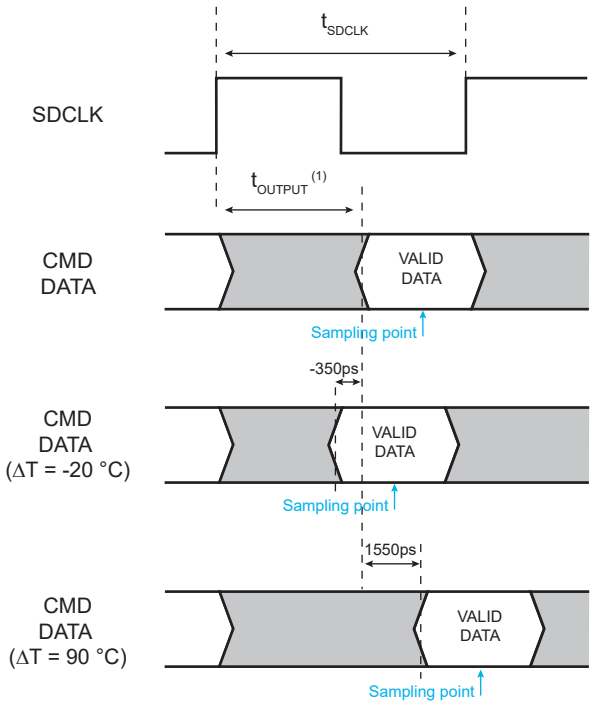
**Figure 48-4. SDR104/HS200 Sampling Point Selection**



### 48.11.2 Retuning Method

Once the data window sampling point has been tuned following the tuning procedure, the data window can be shifted by temperature drift. Thus, the tuning procedure must be applied periodically to adjust the sampling point position. The SDMMC implements a retuning timer which periodically instructs the software to restart the tuning procedure.

**Figure 48-5. Temperature Effect on Data Window**




Note: 1. $t_{OUTPUT}$ varies from 0 to 2 x $t_{SDCLK}$

#### 48.11.2.1 SDMMC Tuning Sequence

The SDMMC tuning sequence must only be done when SD/SDIO SDR104 or e.MMC HS200 is selected and for a 100-MHz SDCLK frequency or higher.

1. Enable the retuning timer (TMREN = 1 in SDMMC_RTC1R).
2. Configure the retuning period by setting TCVAL in SDMMC_RTCVR.
3. Set ‘Retuning Timer Event’ (TEVT) to 1 in SDMMC_RTISTER so that the TEVT status flag in SDMMC_RTISTR rises each time the retuning timer counter period elapses.
4. Set TEVT to 1 in SDMMC_RTISIER to generate an interrupt on the TEVT status flag assertion (optional).
5. Execute the tuning procedure as defined in “Sampling Clock Tuning Procedure” in the “SD Host Controller Simplified Specification V3.00”.
6. Start the retuning timer count (write RLD to 1 in SDMMC_RTC2R). At this step, data can be read by the SDMMC.
7. Each time TEVT is set to 1 in SDMMC_RTISTR:
    1. Execute the tuning procedure as defined in “Sampling Clock Tuning Procedure” in the “SD Host Controller Simplified Specification V3.00” before issuing the next command.
    2. Re-start the retuning timer count (write RLD to 1 in SDMMC_RTC2R).
    3. Resume data reading from the device.

When several instances of SDMMC are implemented in a product, the TEVT status flag of each SDMMC instance can be checked by reading SDMMC_RTSSR.

## 48.12 I/O Calibration

The need for output impedance calibration arises with higher data rates. As the data rate increases, some transmission line effects can occur and lead to the generation of undershoots and overshoots, hence degrading the signal quality.

To avoid these transmission problems, an I/O calibration cell is used to adjust the output impedance to the driven I/Os.

The I/O calibration sequence is mandatory when one of the SD/SDIO UHS-I modes (VS18EN = 1 in SDMMC_HC2R) or e.MMC HS200 (HS200EN = $B_{(hexa)}$) is selected. It must be performed periodically to prevent the output impedance drift. Once the calibration is finished, the I/O calibration cell provides two four-bit control words (CALP[3:0] and CALN[3:0] in SDMMC_CALCR) to tune the output impedance, and thus reach the best transmission performances.

The I/O calibration sequence can be started manually by writing the EN bit to 1 in SDMMC_CALCR. The EN bit is cleared automatically at the end of the calibration.

The I/O calibration sequence can also be performed automatically if the TUNDIS bit is cleared in SDMMC_CALCR. In such case, the calibration starts automatically at the beginning of the tuning procedure when writing EXTUN to 1 in SDMMC_HC2R.

The I/O calibration cell requires a startup time defined by the CNTVAL field in SDMMC_CALCR. Thus, CNTVAL must be configured prior to start the calibration sequence. If ALWYSON is set to 1 in SDMMC_CALCR, the startup time is only required for the first calibration sequence as the analog circuitry is not shut down at the end of the calibration. In order to reduce the power consumption, the analog circuitry can be shut down at the end of the calibration sequence by clearing the ALWYSON bit. In this case, the startup time is performed each time a calibration sequence is started.

## 48.13 Secure Digital MultiMedia Card Controller (SDMMC) User Interface

**Table 48-3. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | SDMA System Address / Argument 2 Register | SDMMC_SSAR | Read/Write | 0x0 |
| 0x04 | Block Size Register | SDMMC_BSR | Read/Write | 0x0 |
| 0x06 | Block Count Register | SDMMC_BCR | Read/Write | 0x0 |
| 0x08 | Argument 1 Register | SDMMC_ARG1R | Read/Write | 0x0 |
| 0x0C | Transfer Mode Register | SDMMC_TMR | Read/Write | 0x0 |
| 0x0E | Command Register | SDMMC_CR | Read/Write | 0x0 |
| 0x10 | Response Register 0 | SDMMC_RR0 | Read-only | 0x0 |
| 0x14 | Response Register 1 | SDMMC_RR1 | Read-only | 0x0 |
| 0x18 | Response Register 2 | SDMMC_RR2 | Read-only | 0x0 |
| 0x1C | Response Register 3 | SDMMC_RR3 | Read-only | 0x0 |
| 0x20 | Buffer Data Port Register | SDMMC_BDPR | Read/Write | –(1) |
| 0x24 | Present State Register | SDMMC_PSR | Read-only | 0x00F8_0000 |
| 0x28 | Host Control 1 Register | SDMMC_HC1R | Read/Write | 0x0 |
| 0x29 | Power Control Register | SDMMC_PCR | Read/Write | 0x0E |
| 0x2A | Block Gap Control Register | SDMMC_BGCR | Read/Write | 0x0 |
| 0x2B | Wakeup Control Register | SDMMC_WCR | Read/Write | 0x0 |
| 0x2C | Clock Control Register | SDMMC_CCR | Read/Write | 0x0 |
| 0x2E | Timeout Control Register | SDMMC_TCR | Read/Write | 0x0 |
| 0x2F | Software Reset Register | SDMMC_SRR | Read/Write | 0x0 |
| 0x30 | Normal Interrupt Status Register | SDMMC_NISTR | Read/Write | 0x0 |
| 0x32 | Error Interrupt Status Register | SDMMC_EISTR | Read/Write | 0x0 |
| 0x34 | Normal Interrupt Status Enable Register | SDMMC_NISTER | Read/Write | 0x0 |
| 0x36 | Error Interrupt Status Enable Register | SDMMC_EISTER | Read/Write | 0x0 |
| 0x38 | Normal Interrupt Signal Enable Register | SDMMC_NISIER | Read/Write | 0x0 |
| 0x3A | Error Interrupt Signal Enable Register | SDMMC_EISIER | Read/Write | 0x0 |
| 0x3C | Auto CMD Error Status Register | SDMMC_ACESR | Read-only | 0x0 |
| 0x3E | Host Control 2 Register | SDMMC_HC2R | Read/Write | 0x0 |
| 0x40 | Capabilities 0 Register | SDMMC_CA0R | Read-only | 0x27EC_0C8C(2)<br>0x27E8_0C8C(3) |
| 0x44 | Capabilities 1 Register | SDMMC_CA1R | Read/Write | 0x0020_0F77(2)<br>0x0020_0070(3) |
| 0x48 | Maximum Current Capabilities Register | SDMMC_MCCAR | Read/Write | 0x0 |
| 0x4C | Reserved | – | – | – |
| 0x50 | Force Event Register for Auto CMD Error Status | SDMMC_FERACES | Write-only | – |
| 0x52 | Force Event Register for Error Interrupt Status | SDMMC_FEREIS | Write-only | – |
| 0x54 | ADMA Error Status Register | SDMMC_AESR | Read-only | 0x0 |

**Table 48-3. Register Mapping (Continued)**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x58 | ADMA System Address Register 0 | SDMMC_ASAR0 | Read/Write | 0x0 |
| 0x5C | Reserved | – | – | – |
| 0x60 | Preset Value Register 0 (for initialization) | SDMMC_PVR0 | Read/Write | 0x0 |
| 0x62 | Preset Value Register 1 (for Default Speed) | SDMMC_PVR1 | Read/Write | 0x0 |
| 0x64 | Preset Value Register 2 (for High Speed) | SDMMC_PVR2 | Read/Write | 0x0 |
| 0x66 | Preset Value Register 3 (for SDR12) | SDMMC_PVR3 | Read/Write | 0x0 |
| 0x68 | Preset Value Register 4 (for SDR25) | SDMMC_PVR4 | Read/Write | 0x0 |
| 0x6A | Preset Value Register 5 (for SDR50) | SDMMC_PVR5 | Read/Write | 0x0 |
| 0x6C | Preset Value Register 6 (for SDR104) | SDMMC_PVR6 | Read/Write | 0x0 |
| 0x6E | Preset Value Register 7 (for DDR50) | SDMMC_PVR7 | Read/Write | 0x0 |
| 0x70–0xF8 | Reserved | – | – | – |
| 0xFC | Slot Interrupt Status Register | SDMMC_SISR | Read-only | 0x0 |
| 0xFE | Host Controller Version Register | SDMMC_HCVR | Read-only | 0x1502 |
| 0x100–0x1FC | Reserved | – | – | – |
| 0x200 | Additional Present State Register | SDMMC_APSR | Read-only | 0x0000_000F(2)<br>0x0000_0000(3) |
| 0x204 | MMC Control 1 Register | SDMMC_MC1R | Read/Write | 0x0 |
| 0x205 | MMC Control 2 Register | SDMMC_MC2R | Write-only | – |
| 0x208 | AHB Control Register | SDMMC_ACR | Read/Write | 0x0 |
| 0x20C | Clock Control 2 Register | SDMMC_CC2R | Read/Write | 0x0 |
| 0x210 | Retuning Timer Control 1 Register | SDMMC_RTC1R | Read/Write | 0x0 |
| 0x211 | Retuning Timer Control 2 Register | SDMMC_RTC2R | Write-only | – |
| 0x214 | Retuning Timer Counter Value Register | SDMMC_RTCVR | Read/Write | 0x0 |
| 0x218 | Retuning Timer Interrupt Status Enable Register | SDMMC_RTISTER | Read/Write | 0x0 |
| 0x219 | Retuning Timer Interrupt Signal Enable Register | SDMMC_RTISIER | Read/Write | 0x0 |
| 0x21C | Retuning Timer Interrupt Status Register | SDMMC_RTISTR | Read/Write | 0x0 |
| 0x21D | Retuning Timer Status Slots Register | SDMMC_RTSSR | Read-only | 0x0 |
| 0x220 | Tuning Control Register | SDMMC_TUNCR | Read/Write | 0x0 |
| 0x224–0x22C | Reserved | – | – | – |
| 0x230 | Capabilities Control Register | SDMMC_CACR | Read/Write | 0x0 |
| 0x234–0x23C | Reserved | – | – | – |
| 0x240 | Calibration Control Register | SDMMC_CALCR | Read/Write | 0x00005000 |
| 0x244–0x2FC | Reserved | – | – | – |

Notes: 1. Unpredictable value read from the dual port RAM
2. Reset value for SDMMC0 instance
3. Reset value for SDMMC1 instance

### 48.13.1 SDMMC SDMA System Address / Argument 2 Register

**Name:** SDMMC_SSAR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ADDR / ARG2 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| ADDR / ARG2 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ADDR / ARG2 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADDR / ARG2 | | | | | | | |

This register contains the physical system memory address used for SDMA transfers or the second argument for Auto CMD23.

- **ADDR: SDMA System Address**

This field is the system memory address for a SDMA transfer. When the SDMMC stops an SDMA transfer, this field points to the system address of the next contiguous data position. This field can be accessed only if no transaction is executing (i.e., after a transaction has stopped). Read operations during transfers may return an invalid value. An interrupt can be generated to instruct the software to update this field. Writing the next system address of the next data position restarts the SDMA transfer.

- **ARG2: Argument 2**

This field is used with Auto CMD23 to set a 32-bit block count value to the CMD23 argument while executing Auto CMD23. If Auto CMD23 is used with ADMA, the full 32-bit block count value can be used. If Auto CMD23 is used without ADMA, the available block count value is limited by SDMMC_BCR. In this case, 65535 blocks is the maximum value.

### 48.13.2 SDMMC Block Size Register

**Name:** SDMMC_BSR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | BOUNDARY | | | – | – | BLKSIZE | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BLKSIZE | | | | | | | |

- **BLKSIZE: Transfer Block Size**

This field specifies the block size of data transfers for CMD17, CMD18, CMD24, CMD25 and CMD53. Values ranging from 1 to 512 can be set. It can be accessed only if no transaction is executing (i.e., after a transaction has stopped). Read operations during transfers may return an invalid value, and write operations are ignored.

- **BOUNDARY: SDMA Buffer Boundary**

This field specifies the size of the contiguous buffer in the system memory. The SDMA transfer waits at every boundary specified by this field and the SDMMC generates the DMA Interrupt to instruct the software to update SDMMC_SSAR. If this field is set to 0 (buffer size = 4 Kbytes), the lowest 12 bits of SDMMC_SSAR.ADDRESS point to data in the contiguous buffer, and the upper 20 bits point to the location of the buffer in the system memory. This function is active when the DMAEN bit is set in SDMMC_TMR.

| Value | Name | Description |
|---|---|---|
| 0 | 4K | 4-Kbyte boundary |
| 1 | 8K | 8-Kbyte boundary |
| 2 | 16K | 16-Kbyte boundary |
| 3 | 32K | 32-Kbyte boundary |
| 4 | 64K | 64-Kbyte boundary |
| 5 | 128K | 128-Kbyte boundary |
| 6 | 256k | 256-Kbyte boundary |
| 7 | 512K | 512-Kbyte boundary |

### 48.13.3 SDMMC Block Count Register

**Name:** SDMMC_BCR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| BLKCNT | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BLKCNT | | | | | | | |

- **BLKCNT: Block Count for Current Transfer**

This field is used only if SDMMC_TMR.BCEN (Block Count Enable) is set to 1 and is valid only for multiple block transfers. BLKCNT is the number of blocks to be transferred and it must be set to a value between 1 and the maximum block count. The SDMMC decrements the block count after each block transfer and stops when the count reaches 0. When this field is set to 0, no data block is transferred.

This register should be accessed only when no transaction is executing (i.e., after transactions are stopped). During data transfer, read operations on this register may return an invalid value and write operations are ignored.

When a suspend command is completed, the number of blocks yet to be transferred can be determined by reading this register. Before issuing a resume command, the previously saved block count is restored.

### 48.13.4 SDMMC Argument 1 Register

**Name:** SDMMC_ARG1R

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ARG1 | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| ARG1 | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ARG1 | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARG1 | | | | | | | |

- **ARG1: Argument 1**

This register contains the SD command argument which is specified as the bit 39-8 of Command-Format in the “Physical Layer Simplified Specification V3.01” or “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”.

### 48.13.5 SDMMC Transfer Mode Register

**Name:** SDMMC_TMR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | MSBSEL | DTDSEL | ACMDEN | | BCEN | DMAEN |

This register is used to control data transfers. The user shall set this register before issuing a command which transfers data (refer to bit DPSEL in SDMMC_CR), or before issuing a Resume command. The user must save the value of this register when the data transfer is suspended (as a result of a Suspend command) and restore it before issuing a Resume command. To prevent data loss, this register cannot be written while data transactions are in progress. Writes to this register are ignored when bit SDMMC_PSR.CMDINHD is 1.

**Table 48-4. Determining the Transfer Type**

| MSBSEL | BCEN | BLKCNT (SDMMC_BCR) | Function |
|---|---|---|---|
| 0 | Don’t care | Don’t care | Single Transfer |
| 1 | 0 | Don’t care | Infinite Transfer |
| 1 | 1 | Not Zero | Multiple Transfer |
| 1 | 1 | Zero | Stop Multiple Transfer |

- **DMAEN: DMA Enable**

This bit enables the DMA functionality described in section “Supporting DMA” in “SD Host Controller Simplified Specification V3.00”. DMA can be enabled only if it is supported as indicated by the bit SDMMC_CA0R.ADMA2SUP. One of the DMA modes can be selected using the field SDMMC_HC1R.DMASEL. If DMA is not supported, this bit is meaningless and then always reads 0. When this bit is set to 1, a DMA operation begins when the user writes to the upper byte of SDMMC_CR.

0 (DISABLED): DMA functionality is disabled.

1 (ENABLED): DMA functionality is enabled.

- **BCEN: Block Count Enable**

This bit is used to enable SDMMC_BCR, which is only relevant for multiple block transfers. When this bit is 0, SDMMC_BCR is disabled, which is useful when executing an infinite transfer (refer to Table 48-4). If an ADMA2 transfer is more than 65535 blocks, this bit is set to 0 and the data transfer length is designated by the Descriptor Table.

0 (DISABLED): Block count is disabled.

1 (ENABLED): Block count is enabled.

- **ACMDEN: Auto Command Enable**

Two methods can be used to stop Multiple-block read and write operation:

- Auto CMD12: when the ACMDEN field is set to 1, the SDMMC issues CMD12 automatically when the last block transfer is completed. An Auto CMD12 error is indicated to SDMMC_ACESR. Auto CMD12 is not enabled if the command does not require CMD12.
- Auto CMD23: when the ACMDEN field is set to 2, the SDMMC issues a CMD23 automatically before issuing a command specified in SDMMC_CR.

The following conditions are required to use Auto CMD23:

– A memory card that supports CMD23 (SCR[33] = 1)

– If DMA is used, it must be ADMA (SDMA not supported).

– Only CMD18 or CMD25 is issued.

Note: The SDMMC does not check the command index.

Auto CMD23 can be used with or without ADMA. By writing SDMMC_CR, the SDMMC issues a CMD23 first and then issues a command specified by the SDMMC_CR.CMDIDX field. If CMD23 response errors are detected, the second command is not issued. A CMD23 error is indicated in SDMMC_ACESR. The CMD23 argument (32-bit block count value) is set in SDMMC_SSAR.

This field determines the use of auto command functions.

| Value | Name | Description |
|---|---|---|
| 0 | DISABLED | Auto Command Disabled |
| 1 | CMD12 | Auto CMD12 Enabled |
| 2 | CMD23 | Auto CMD23 Enabled |
| 3 | – | Reserved |

- **DTDSEL: Data Transfer Direction Selection**

This bit defines the direction of the DAT lines data transfers. Set this bit to 1 to transfer data from the device (SD Card/SDIO/e.MMC) to the SDMMC, and to 0 for all other commands.

0 (WRITE): Writes data from the SDMMC to the device.

1 (READ): Reads data from the device to the SDMMC.

- **MSBSEL: Multi/Single Block Selection**

This bit is set to 1 when issuing multiple-block transfer commands using DAT line(s). For any other commands, set this bit to 0. If this bit is 0, it is not necessary to set SDMMC_BCR (refer to Table 48-4).

### 48.13.6 SDMMC Command Register

**Name:** SDMMC_CR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | CMDIDX | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CMDTYP | | DPSEL | CMDICEN | CMDCCEN | – | RESPTYP | |

- **RESPTYP: Response Type**

This field is set according to the response type expected for the command index (CMDIDX).

| Value | Name | Description |
|---|---|---|
| 0 | NORESP | No Response |
| 1 | RL136 | Response Length 136 |
| 2 | RL48 | Response Length 48 |
| 3 | RL48BUSY | Response Length 48 with Busy |

- **CMDCCEN: Command CRC Check Enable**

If this bit is set to 1, the SDMMC checks the CRC field in the response. If an error is detected, it is reported as a Command CRC Error (CMDCRC) in SDMMC_EISTR. If this bit is set to 0, the CRC field is not checked. The position of the CRC field is determined according to the length of the response.

0 (DISABLED): The Command CRC Check is disabled.

1 (ENABLED): The Command CRC Check is enabled.

- **CMDICEN: Command Index Check Enable**

If this bit is set to 1, the SDMMC checks the Index field in the response to see if it has the same value as the command index. If it has not, it is reported as a Command Index Error (CMDIDX) in SDMMC_EISTR. If this bit is set to 0, the Index field of the response is not checked.

0 (DISABLED): The Command Index Check is disabled.

1 (ENABLED): The Command Index Check is enabled.

- **DPSEL: Data Present Select**

This bit is set to 1 to indicate that data is present and shall be transferred using the DAT lines. It is set to 0 for the following:

  – Commands using only CMD line (Ex. CMD52)

  – Commands with no data transfer but using Busy signal on DAT[0] line (Ex. CMD38)

  – Resume command

0: No data present

1: Data present

- **CMDTYP: Command Type**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | Other commands |
| 1 | SUSPEND | CMD52 to write “Bus Suspend” in the Card Common Control Registers (CCCR) (for SDIO only) |
| 2 | RESUME | CMD52 to write “Function Select” in the Card Common Control Registers (CCCR) (for SDIO only) |
| 3 | ABORT | CMD12, CMD52 to write “I/O Abort” in the Card Common Control Registers (CCCR) (for SDIO only) |

- **CMDIDX: Command Index**

This bit shall be set to the command number (CMD0-63, ACMD0-63) that is specified in bits 45-40 of the Command-Format in the “Physical Layer Simplified Specification V3.01”, “SDIO Simplified Specification V3.00”, and “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”.

### 48.13.7 SDMMC Response Register

**Name:** SDMMC_RRx [x=0..3]

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CMDRESP | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| CMDRESP | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CMDRESP | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CMDRESP | | | | | | | |

- **CMDRESP: Command Response**

The table below describes the mapping of command responses from the SD_SDIO/eMMC bus to these registers for each responses type. In this table, R[] refers to a bit range of the response data as transmitted on the SD_SDIO/eMMC bus.

| Type of response | Meaning of response | Response field | Response register |
|---|---|---|---|
| R1, R1b (normal response) | Card Status | R[39:8] | SDMMC_RR0[31:0] |
| R1b (Auto CMD12 response) | Card Status for Auto CMD12 | R[39:8] | SDMMC_RR3[31:0] |
| R1 (Auto CMD23 response) | Card Status for Auto CMD23 | R[39:8] | SDMMC_RR3[31:0] |
| R2 (CID, CSD register) | CID or CSD register | R[127:8] | SDMMC_RR0[31:0]<br>SDMMC_RR1[31:0]<br>SDMMC_RR2[31:0]<br>SDMMC_RR3[23:0] |
| R3 (OCR register) | OCR register for memory | R[39:8] | SDMMC_RR0[31:0] |
| R4 (OCR register) | OCR register for I/O | R[39:8] | SDMMC_RR0[31:0] |
| R5, R5b | SDIO response | R[39:8] | SDMMC_RR0[31:0] |
| R6 (Published RCA response) | New published RCA[31:16] and Card status bits | R[39:8] | SDMMC_RR0[31:0] |

### 48.13.8 SDMMC Buffer Data Port Register

**Name:** SDMMC_BDPR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| BUFDATA | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| BUFDATA | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| BUFDATA | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BUFDATA | | | | | | | |

- **BUFDATA: Buffer Data**

The SDMMC data buffer can be accessed through this 32-bit Data Port register.

### 48.13.9 SDMMC Present State Register

**Name:** SDMMC_PSR

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | CMDLL |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| DATLL | | | | WRPPL | CARDDPL | CARDSS | CARDINS |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | BUFRDEN | BUFWREN | RTACT | WTACT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | DLACT | CMDINHD | CMDINHC |

- **CMDINHC: Command Inhibit (CMD)**

If this bit is 0, it indicates the CMD line is not in use and the SDMMC can issue a command using the CMD line. This bit is set to 1 immediately after SDMMC_CR is written. This bit is cleared when the command response is received. Auto CMD12 and Auto CMD23 consist of two responses. In this case, this bit is not cleared by the CMD12 or CMD23 response, but by the Read/Write command response.

Status issuing Auto CMD12 is not read from this bit. So, if a command is issued during Auto CMD12 operation, the SDMMC manages to issue both commands: CMD12 and a command set by SDMMC_CR.

Even if the Command Inhibit (DAT) is set to 1, commands using only the CMD line can be issued if this bit is 0.

A change from 1 to 0 raises the Command Complete (CMDC) status flag in SDMMC_NISTR if SDMMC_NISTER.CMDC is set to 1. An interrupt is generated if SDMMC_NISIER.CMDC is set to 1.

If the SDMMC cannot issue the command because of a command conflict error (refer to CMDCRC in SDMMC_EISTR) or because of a ‘Command Not Issued By Auto CMD12’ error (refer to Section 48.13.31 “SDMMC Auto CMD Error Status Register”), this bit remains 1 and Command Complete is not set.

0: Can issue a command using only CMD line.

1: Cannot issue a command.

- **CMDINHD: Command Inhibit (DAT)**

This status bit is 1 if either the DAT Line Active (DLACT) or the Read Transfer Active (RTACT) is set to 1. If this bit is 0, it indicates that the SDMMC can issue the next command. Commands with a Busy signal belong to Command Inhibit (DAT) (ex. R1b, R5b type). A change from 1 to 0 raises the Transfer Complete (TRFC) status flag in SDMMC_NISTR if SDMMC_NISTER.TRFC is set to 1. An interrupt is generated if SDMMC_NISIER.TRFC is set to 1.

Note: The software can save registers in the 000-00Dh range for a suspend transaction after this bit has changed from 1 to 0.

0: Can issue a command which uses the DAT line(s).

1: Cannot issue a command which uses the DAT line(s).

- **DLACT: DAT Line Active**

This bit indicates whether one of the DAT lines on the bus is in use.

In the case of read transactions:

This status indicates whether a read transfer is executing on the bus. A change from 1 to 0 resulting from setting the Stop At Block Gap Request (STPBGR) raises the Block Gap Event (BLKGE) status flag in SDMMC_NISTR if SDMMC_NISTER.BLKGE is set to 1. An interrupt is generated if SDMMC_NISIER.BLKGE is set to 1. Refer to section “Read Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for details on timing.

This bit is set in either of the following cases:

– After the end bit of the read command.

– When writing 1 to SDMMC_BGCR.CONTR (Continue Request) to restart a read transfer.

This bit is SDMMC cleared in either of the following cases:

– When the end bit of the last data block is sent from the bus to the SDMMC. In case of ADMA2, the last block is designated by the last transfer of the Descriptor Table.

– When a read transfer is stopped at the block gap initiated by a Stop At Block Gap Request (STPBGR).

The SDMMC stops a read operation at the start of the interrupt cycle by driving the Read Wait (DAT[2] line) or by stopping the SD Clock. If the Read Wait signal is already driven (due to the fact that the data buffer cannot receive data), the SDMMC can continue to stop the read operation by driving the Read Wait signal. It is necessary to support the Read Wait in order to use the Suspend/Resume operation.

In the case of write transactions:

This status indicates that a write transfer is executing on the bus. A change from 1 to 0 raises the Transfer Complete (TRFC) status flag in SDMMC_NISTR if SDMMC_NISTER.TRFC is set to 1. An interrupt is generated if SDMMC_NISIER.TRFC is set to 1. Refer to section “Write Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for details on timing.

This bit is set in either of the following cases:

– After the end bit of the write command.

– When writing 1 to SDMMC_BGCR.CONTR (Continue Request) to continue a write transfer.

This bit is cleared in either of the following cases:

– When the card releases Write Busy of the last data block. If the card does not drive a Busy signal for 8 SDCLK, the SDMMC considers the card drive “Not Busy”. In the case of ADMA2, the last block is designated by the last transfer of the Descriptor Table.

– When the card releases Write Busy prior to wait for write transfer as a result of a Stop At Block Gap Request (STPBGR).

Command with Busy:

This status indicates whether a command that indicates Busy (ex. erase command for memory) is executing on the bus. This bit is set to 1 after the end bit of the command with Busy and cleared when Busy is de-asserted. A change from 1 to 0 raises the Transfer Complete (TRFC) status flag in SDMMC_NISTR if SDMMC_NISTER.TRFC is set to 1. An interrupt is generated if SDMMC_NISIER.TRFC is set to 1. Refer to Figures 2.11 to 2.13 in the “SD Host Controller Simplified Specification V3.00”.

0: DAT Line Inactive

1: DAT Line Active

- **WTACT: Write Transfer Active**

This bit indicates a write transfer is active. If this bit is 0, it means no valid write data exists in the SDMMC. Refer to section “Write Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

This bit is set to 1 in either of the following conditions:

– After the end bit of the write command.

– When a write operation is restarted by writing a 1 to SDMMC_BGCR.CONTR (Continue Request).

This bit is cleared to 0 in either of the following conditions:

– After getting the CRC status of the last data block as specified by the transfer count (single and multiple). In case of ADMA2, transfer count is designated by the descriptor table.

– After getting the CRC status of any block where a data transmission is about to be stopped by a Stop At Block Gap Request (STPBGR) of SDMMC_BGCR.

During a write transaction and as the result of the Stop At Block Gap Request (STPBGR) being set, a change from 1 to 0 raises the Block Gap Event (BLKGE) status flag in SDMMC_NISTR if SDMMC_NISTER.BLKGE is set to 1. An interrupt is generated if BLKGE is set to 1 in SDMMC_NISIER. This status is useful to determine whether non-DAT line commands can be issued during Write Busy.

- **RTACT: Read Transfer Active**

This bit is used to detect completion of a read transfer. Refer to section “Read Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

This bit is set to 1 in either of the following conditions:

– After the end bit of the read command.

– When a read operation is restarted by writing a 1 to SDMMC_BGCR.CONTR (Continue Request).

This bit is cleared to 0 in either of the following conditions:

– When the last data block as specified by Transfer Block Size (BLKSIZE) is transferred to the system.

– In case of ADMA2, end of read is designated by the descriptor table.

– When all valid data blocks in the SDMMC have been transferred to the system and no current block transfers are being sent as a result of the Stop At Block Gap Request (STPBGR) of SDMMC_BGCR being set to 1.

A change from 1 to 0 raises the Transfer Complete (TRFC) status flag in SDMMC_NISTR if SDMMC_NISTER.TRFC is set to 1. An interrupt is generated if SDMMC_NISIER.TRFC is set to 1.

- **BUFWREN: Buffer Write Enable**

This bit is used for non-DMA write transfers. This flag indicates if space is available for write data. If this bit is 1, data can be written to the buffer.

A change from 1 to 0 occurs when all the block data are written to the buffer.

A change from 0 to 1 occurs when top of block data can be written to the buffer. This raises the Buffer Write Ready (BRWRDY) status flag in SDMMC_NISTR if SDMMC_NISTER.BRWRDY is set to 1. An interrupt is generated if SDMMC_NISIER.BRWRDY is set to 1.

- **BUFRDEN: Buffer Read Enable**

This bit is used for non-DMA read transfers. This flag indicates that valid data exists in the SDMMC data buffer. If this bit is 1, readable data exists in the buffer.

A change from 1 to 0 occurs when all the block data is read from the buffer.

A change from 0 to 1 occurs when block data is ready in the buffer. This raises the Buffer Read Ready (BRDRDY) status flag in SDMMC_NISTR if SDMMC_NISTER.BRDRDY is set to 1. An interrupt is generated if SDMMC_NISIER.BRDRDY is set to 1.

- **CARDINS: Card Inserted**

This bit indicates wether a card has been inserted. The SDMMC debounces this signal so that the user does not need to wait for it to stabilize.

A change from 0 to 1 raises the Card Insertion (CINS) status flag in SDMMC_NISTR if SDMMC_NISTER.CINS is set to 1. An interrupt is generated if SDMMC_NISIER.CINS is set to 1.

A change from 1 to 0 raises the Card Removal (CREM) status flag in SDMMC_NISTR if SDMMC_NISTER.CREM is set to 1. An interrupt is generated if SDMMC_NISIER.CREM is set to 1.

The Software Reset For All (SWRSTALL) in SDMMC_SRR does not affect this bit.

- **CARDSS: Card State Stable**

This bit is used for testing. If it is 0, the CARDDPL is not stable. If this bit is set to 1, it means that the CARDDPL is stable. No Card state can be detected if this bit is set to 1 and CARDINS is set to 0.

The Software Reset For All (SWRSTALL) in SDMMC_SRR does not affect this bit.

0: Reset or debouncing

1: No card or card inserted

- **CARDDPL: Card Detect Pin Level**

This bit reflects the inverse value of the SDMMC_CD pin. Debouncing is not performed on this bit. This bit may be valid when CARDSS is set to 1, but it is not guaranteed because of the propagation delay. Use of this bit is limited to testing since it must be debounced by software.

0: No card present (SDMMC_CD = 1)

1: Card present (SDMMC_CD = 0)

- **WRPPL: Write Protect Pin Level**

The Write Protect Switch is supported for memory and combo cards. This bit reflects the SDMMC_WP pin.

0: Write protected (SDMMC_WP = 0)

1: Write enabled (SDMMC_WP = 1)

- **DATLL: DAT[3:0] Line Level**

This status is used to check the DAT line level to recover from errors, and for debugging. This is especially useful in detecting the Busy signal level from DAT[0].

- **CMDLL: CMD Line Level**

This status is used to check the CMD line level to recover from errors, and for debugging.

### 48.13.10 SDMMC Host Control 1 Register (SD_SDIO)

**Name:** SDMMC_HC1R (SD_SDIO)

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CARDDSEL | CARDDTL | – | DMASEL | | HSEN | DW | LEDCTRL |

- **LEDCTRL: LED Control**

This bit is used to caution the user not to remove the card while it is being accessed. If the software is going to issue multiple commands, this bit is set to 1 during all transactions.

0 (OFF): LED off

1 (ON): LED on

- **DW: Data Width**

This bit selects the data width of the SDMMC. It must be set to match the data width of the card.

0 (1_BIT): 1-bit mode

1 (4_BIT): 4-bit mode

Note: If the Extended Data Transfer Width is 1, this bit has no effect and the data width is 8-bit mode.

- **HSEN: High Speed Enable**

Before setting this bit, the user must check the High Speed Support (HSSUP) in SDMMC_CA0R.

If this bit is set to 0 (default), the SDMMC outputs CMD line and DAT lines at the falling edge of the SD clock (up to 25 MHz). If this bit is set to 1, the SDMMC outputs the CMD line and the DAT lines at the rising edge of the SD clock (up to 50 MHz).

If Preset Value Enable (PVALEN) in SDMMC_HC2R is set to 1, the user needs to reset SD Clock Enable (SDCLKEN) before changing this bit to avoid generating clock glitches. After setting this bit to 1, the user sets SDCLEN to 1 again.

0: Normal Speed mode.

1: High Speed mode.

Note:
1. This bit is effective only if SDMMC_MC1R.DDR is set to 0.
2. The clock divider (DIV) in SDMMC_CCR must be set to a value different from 0 when HSEN is 1.

- **DMASEL: DMA Select**

One of the supported DAM modes can be selected. The user must check support of DMA modes by referring the SDMMC_CA0R. Use of selected DMA is determined by DMA Enable (DMAEN) in SDMMC_TMR.

| Value | Name | Description |
|---|---|---|
| 0 | SDMA | SDMA is selected |
| 1 | – | Reserved |
| 2 | ADMA32 | 32-bit Address ADMA2 is selected |
| 3 | – | Reserved |

- **CARDDTL: Card Detect Test Level**

This bit is enabled while the Card Detect Signal Selection (CARDDSEL) is set to 1 and it indicates whether the card is inserted or not.

0: No card.

1: Card inserted.

- **CARDDSEL: Card Detect Signal Selection**

This bit selects the source for the card detection.

0: The SDMMC_CD pin is selected.

1: The Card Detect Test Level (CARDDTL) is selected (for test purpose).

### 48.13.11 SDMMC Host Control 1 Register (eMMC)

**Name:** SDMMC_HC1R (eMMC)

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | EXTDW | DMASEL | | HSEN | DW | – |

- **DW: Data Width**

This bit selects the data width of the SDMMC. It must be set to match the data width of the card.

0 (1_BIT): 1-bit mode

1 (4_BIT): 4-bit mode

Note: If the Extended Data Transfer Width is 1, this bit has no effect and the data width is 8-bit mode.

- **HSEN: High Speed Enable**

Before setting this bit, the user must check the High Speed Support (HSSUP) in SDMMC_CA0R.

If this bit is set to 0 (default), the SDMMC outputs CMD line and DAT lines at the falling edge of the SD clock (up to 25 MHz). If this bit is set to 1, the SDMMC outputs the CMD line and the DAT lines at the rising edge of the SD clock (up to 50 MHz).

If Preset Value Enable (PVALEN) in SDMMC_HC2R is set to 1, the user needs to reset the SD Clock Enable (SDCLKEN) before changing this bit to avoid generating clock glitches. After setting this bit to 1, the user sets SDCLEN to 1 again.

0: Normal Speed mode.

1: High Speed mode.

Note:
1. This bit is effective only if SDMMC_MC1R.DDR is set to 0.
2. The clock divider (DIV) in SDMMC_CCR must be set to a value different from 0 when HSEN is 1.

- **DMASEL: DMA Select**

One of the supported DAM modes can be selected. The user must check support of DMA modes by referring the SDMMC_CA0R. Use of selected DMA is determined by DMA Enable (DMAEN) in SDMMC_TMR.

| Value | Name | Description |
|---|---|---|
| 0 | SDMA | SDMA is selected |
| 1 | – | Reserved |
| 2 | ADMA32 | 32-bit Address ADMA2 is selected |
| 3 | – | Reserved |

- **EXTDW: Extended Data Width**

This bit controls the 8-bit Bus Width mode for embedded devices. Support of this function is indicated in 8-bit Support for Embedded Device in SDMMC_CA0R. If a device supports the 8-bit mode, this may be set to 1. If this bit is 0, the bus width is controlled by Data Width (DW).

### 48.13.12 SDMMC Power Control Register

**Name:** SDMMC_PCR

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | SDBPWR |

- **SDBPWR: SD Bus Power**

This bit is automatically cleared by the SDMMC if the card is removed. If this bit is cleared, the SDMMC stops driving SDMMC_CMD and SDMMC_DAT[7:0] (tri-state) and drives SDMMC_CK to low level.

### 48.13.13 SDMMC Block Gap Control Register (SD_SDIO)

**Name:** SDMMC_BGCR (SD_SDIO)

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | INTBG | RWCTRL | CONTR | STPBGR |

- **STPBGR: Stop At Block Gap Request**

This bit is used to stop executing read and write transactions at the next block gap for non-DMA, SDMA, and ADMA transfers. The user must leave this bit set to 1 until Transfer Complete (TRFC) in SDMMC_NISTR. Clearing both Stop At Block Gap Request and Continue Request does not cause the transaction to restart. This bit can be set whether the card supports the Read Wait signal or not.

During read transfers, the SDMMC stops the transaction by using the Read Wait signal (SDMMC_DAT[2]) if supported, or by stopping the SD clock otherwise.

In case of write transfers in which the user writes data to SDMMC_BDPR, this bit must be set to 1 after all the block of data is written. If this bit is set to 1, the user does not write data to SDMMC_BDPR.

This bit affects Read Transfer Active (RTACT), Write Transfer Active (WTACT), DAT Line Active (DLACT) and Command Inhibit (DAT) (CMDINHD) in SDMMC_PSR.

Refer to the “Abort Transaction” and “Suspend/Resume” sections in the “SD Host Controller Simplified Specification V3.00” for more details.

0: Transfer

1: Stop

- **CONTR: Continue Request**

This bit is used to restart a transaction which was stopped using a Stop At Block Gap Request (STPBGR). To cancel stop at the block gap, set STPBGR to 0 and set this bit to 1 to restart the transfer.

The SDMMC automatically clears this bit in either of the following cases:

  – In the case of a read transaction, the DAT Line Active (DLACT) changes from 0 to 1 as a read transaction restarts.

  – In the case of a write transaction, the Write Transfer Active (WTACT) changes from 0 to 1 as the write transaction restarts.

Therefore, it is not necessary to set this bit to 0. If STPBGR is set to 1, any write to this bit is ignored.

Refer to the “Abort Transaction” and “Suspend/Resume” sections in the “SD Host Controller Simplified Specification V3.00” for more details.

0: No affect

1: Restart

- **RWCTRL: Read Wait Control**

The Read Wait control is optional for SDIO cards. If the card supports Read Wait, set this bit to enable use of the Read Wait protocol to stop read data using the SDMMC_DAT[2] line. Otherwise, the SDMMC stops the SDCLK to hold read data, which restricts command generation. When the software detects an SD card insertion, this bit must be set according to the CCCR of the SDIO card. If the card does not support Read Wait, this bit shall never be set to 1, otherwise an SDMMC_DAT line conflict may occur. If this bit is set to 0, Suspend/Resume cannot be supported.

0: Disables Read Wait control.

1: Enables Read Wait control.

- **INTBG: Interrupt at Block Gap**

This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. Setting to 1 enables interrupt detection at the block gap for a multiple block transfer. If the SDIO card cannot signal an interrupt during a multiple block transfer, this bit should be set to 0. When the software detects an SDIO card insertion, it sets this bit according to the CCCR of the SDIO card.

0 (DISABLED): Interrupt detection disabled

1 (ENABLED): Interrupt detection enabled

### 48.13.14 SDMMC Block Gap Control Register (eMMC)

**Name:** SDMMC_BGCR (eMMC)

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | CONTR | STPBGR |

- **STPBGR: Stop At Block Gap Request**

This bit is used to stop executing read and write transactions at the next block gap for non-DMA, SDMA, and ADMA transfers. The user must leave this bit set to 1 until Transfer Complete (TRFC) in SDMMC_NISTR. Clearing both Stop At Block Gap Request and Continue Request does not cause the transaction to restart. This bit can be set whether the card supports the Read Wait signal or not.

During read transfers, the SDMMC stops the transaction by using the Read Wait signal (SDMMC_DAT[2]) if supported, or by stopping the SD clock otherwise.

In case of write transfers in which the user writes data to SDMMC_BDPR, this bit must be set to 1 after all the block of data is written. If this bit is set to 1, the user does not write data to SDMMC_BDPR.

This bit affects Read Transfer Active (RTACT), Write Transfer Active (WTACT), DAT Line Active (DLACT) and Command Inhibit (DAT) (CMDINHD) in SDMMC_PSR.

Refer to the “Abort Transaction” and “Suspend/Resume” sections in the “SD Host Controller Simplified Specification V3.00” for more details.

0: Transfer

1: Stop

- **CONTR: Continue Request**

This bit is used to restart a transaction which was stopped using a Stop At Block Gap Request (STPBGR). To cancel stop at the block gap, set STPBGR to 0 and set this bit to 1 to restart the transfer.

The SDMMC automatically clears this bit in either of the following cases:

– In the case of a read transaction, the DAT Line Active (DLACT) changes from 0 to 1 as a read transaction restarts.

– In the case of a write transaction, the Write Transfer Active (WTACT) changes from 0 to 1 as the write transaction restarts.

Therefore, it is not necessary to set this bit to 0. If STPBGR is set to 1, any write to this bit is ignored.

Refer to the “Abort Transaction” and “Suspend/Resume” sections in the “SD Host Controller Simplified Specification V3.00” for more details.

0: No affect

1: Restart

### 48.13.15 SDMMC Wakeup Control Register (SD_SDIO)

**Name:** SDMMC_WCR (SD_SDIO)

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | WKENCREM | WKENCINS | WKENCINT |

- **WKENCINT: Wakeup Event Enable on Card Interrupt**

This bit enables a wakeup event via Card Interrupt (CINT) in SDMMC_NISTR. This bit can be set to 1 if FN_WUS (Wakeup Support) in the CIS (Card Information Structure) is set to 1 in the SDIO card.

0 (DISABLED): Wakeup Event disabled

1 (ENABLED): Wakeup Event enabled

- **WKENCINS: Wakeup Event Enable on Card Insertion**

This bit enables a wakeup event via Card Insertion (CINS) in SDMMC_NISTR. FN_WUS (Wakeup Support) in the CIS (Card Information Structure) does not affect this bit.

0 (DISABLED): Wakeup Event disabled

1 (ENABLED): Wakeup Event enabled

- **WKENCREM: Wakeup Event Enable on Card Removal**

This bit enables a wakeup event via Card Removal (CREM) in SDMMC_NISTR. FN_WUS (Wakeup Support) in the CIS (Card Information Structure) does not affect this bit.

0 (DISABLED): Wakeup Event disabled

1 (ENABLED): Wakeup Event enabled

### 48.13.16 SDMMC Clock Control Register

**Name:** SDMMC_CCR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SDCLKFSEL | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| USDCLKFSEL | | CLKGSEL | – | – | SDCLKEN | INTCLKS | INTCLKEN |

- **INTCLKEN: Internal Clock Enable**

This bit is set to 0 when the SDMMC is not used or is awaiting a wakeup interrupt. In this case, its internal clock is stopped to reach a very low power state. Registers are still able to be read and written. The clock starts to oscillate when this bit is set to 1. Once the clock oscillation is stable, the SDMMC sets Internal Clock Stable (INTCLKS) in this register to 1.

This bit does not affect card detection.

0: The internal clock stops.

1: The internal clock oscillates.

- **INTCLKS: Internal Clock Stable**

This bit is set to 1 when the SD clock is stable after setting SDMMC_CCR.INTCLKEN (Internal Clock Enable) to 1. The user must wait to set SD Clock Enable (SDCLKEN) until this bit is set to 1.

0: Internal clock not ready

1: Internal clock ready

- **SDCLKEN: SD Clock Enable**

The SDMMC stops the SD Clock when writing this bit to 0. SDCLK Frequency Select (SDCLKFSEL) can be changed when this bit is 0. Then, the SDMMC maintains the same clock frequency until SDCLK is stopped (Stop at SDCLK = 0). If Card Inserted (CARDINS) in SDMMC_PSR is cleared, this bit is also cleared.

0: SD Clock disabled

1: SD Clock enabled

- **CLKGSEL: Clock Generator Select**

This bit is used to select the clock generator mode in the SDCLK Frequency Select field. If the Programmable mode is not supported (SDMMC_CA1R.CLKMULT (Clock Multiplier) set to 0), then this bit cannot be written and is always read at 0.

This bit depends on the setting of Preset Value Enable (PVALEN) in SDMMC_HC2R.

If PVALEN = 0, this bit is set by the user.

If PVALEN = 1, this bit is automatically set to a value specified in one of the SDMMC_PVRx.

0: Divided Clock mode (BASECLK is used to generate SDCLK).

1: Programmable Clock mode (MULTCLK is used to generate SDCLK).

- **USDCLKFSEL: Upper Bits of SDCLK Frequency Select**

These bits expand the SDCLK Frequency Select (SDCLKFSEL) to 10 bits. These two bits are assigned to bit 09-08 of the clock divider as described in SDCLKFSEL.

- **SDCLKFSEL: SDCLK Frequency Select**

This register is used to select the frequency of the SDCLK pin. There are two SDCLK Frequency modes according to Clock Generator Select (CLKGSEL).

The length of the clock divider (DIV) is extended to 10 bits (DIV[9:8] = USDCLKFSEL, DIV[7:0] = SDCLKFSEL)

– 10-bit Divided Clock Mode (CLKGSEL = 0): $f_{SDCLK} = f_{BASECLK}/(2 \times DIV)$. If DIV = 0 then $f_{SDCLK} = f_{BASECLK}$

– Programmable Clock Mode (CLKGSEL = 1): $f_{SDCLK} = f_{MULTCLK}/(DIV + 1)$

This field depends on the setting of Preset Value Enable (PVALEN) in SDMMC_HC2R.

If PVALEN = 0, this field is set by the user.

If PVALEN = 1, this field is automatically set to a value specified in one of the SDMMC_PVR.

### 48.13.17 SDMMC Timeout Control Register

**Name:** SDMMC_TCR

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | DTCVAL | | | |

- **DTCVAL: Data Timeout Counter Value**

This value determines the interval at which DAT line timeouts are detected. For more information about timeout generation, refer to Data Timeout Error (DATTEO) in SDMMC_EISTR. When setting this register, the user can prevent inadvertent timeout events by clearing the Data Timeout Error Status Enable (in SDMMC_EISTER).

$$TIMEOUT_{(\mu s)} = \frac{2^{13 + DTCVAL}}{f_{BASECLK(MHz)}}$$

Note: DTCVAL = $f_{(Hexa)}$ is reserved.

### 48.13.18 SDMMC Software Reset Register

**Name:** SDMMC_SRR

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | SWRSTDAT | SWRSTCMD | SWRSTALL |

- **SWRSTALL: Software reset for All**

This reset affects the entire SDMMC except the card detection circuit. During initialization, the SDMMC must be reset by setting this bit to 1. This bit is automatically cleared to 0 when SDMMC_CA0R and SDMMC_CA1R are valid and the user can read them. If this bit is set to 1, the user should issue a reset command and reinitialize the card.

List of registers cleared to 0:

– "SDMMC SDMA System Address / Argument 2 Register"
– "SDMMC Block Size Register"
– "SDMMC Block Count Register"
– "SDMMC Argument 1 Register"
– "SDMMC Command Register"
– "SDMMC Transfer Mode Register"
– "SDMMC Response Register"
– "SDMMC Buffer Data Port Register"
– "SDMMC Present State Register" (except CMDLL, DATLL, WRPPL, CARDDDPL, CARDSS, CARDINS)
– "SDMMC Host Control 1 Register (SD_SDIO)"
– "SDMMC Host Control 1 Register (eMMC)"
– "SDMMC Power Control Register"
– "SDMMC Block Gap Control Register (SD_SDIO)"
– "SDMMC Block Gap Control Register (eMMC)"
– "SDMMC Wakeup Control Register (SD_SDIO)"
– "SDMMC Clock Control Register"
– "SDMMC Timeout Control Register"
– "SDMMC Normal Interrupt Status Register (SD_SDIO)"
– "SDMMC Error Interrupt Status Register (SD_SDIO)"
– "SDMMC Normal Interrupt Status Enable Register (SD_SDIO)"
– "SDMMC Error Interrupt Status Enable Register (SD_SDIO)"
– "SDMMC Normal Interrupt Signal Enable Register (SD_SDIO)"
– "SDMMC Error Interrupt Signal Enable Register (SD_SDIO)"
– "SDMMC Auto CMD Error Status Register"
– "SDMMC Host Control 2 Register (SD_SDIO)"
– "SDMMC ADMA Error Status Register"
– "SDMMC ADMA System Address Register"
– "SDMMC Slot Interrupt Status Register" (except NBCLKP, NBINTP, BWTPRE)
– "SDMMC Slot Interrupt Status Register"
– "SDMMC e.MMC Control 1 Register"
– "SDMMC e.MMC Control 2 Register"

– “SDMMC AHB Control Register”
– “SDMMC Clock Control 2 Register”
– “SDMMC Retuning Control 1 Register”
– “SDMMC Retuning Counter Value Register”
– “SDMMC Retuning Interrupt Status Enable Register”
– “SDMMC Retuning Interrupt Signal Enable Register”
– “SDMMC Retuning Interrupt Status Register”
– “SDMMC Tuning Control Register”
– “SDMMC Capabilities Control Register” (except KEY)

0: Work

1: Reset

- **SWRSTCMD: Software reset for CMD line**

Only part of a command circuit is reset.

The following registers and bits are cleared by this bit:

“SDMMC Present State Register”

– Command Inhibit (CMD) (CMDINHC)

“SDMMC Normal Interrupt Status Register (SD_SDIO)” and “SDMMC Normal Interrupt Status Register (eMMC)”

– Command Complete (CMDC)

0: Work

1: Reset

- **SWRSTDAT: Software reset for DAT line**

Only part of a data circuit is reset. The DMA circuit is also reset.

The following registers and bits are cleared by this bit:

“SDMMC Buffer Data Port Register”

– Buffer is cleared and initialized.

“SDMMC Present State Register”

– Buffer Read Enable (BUFRDEN)
– Buffer Write Enable (BUFWREN)
– Read Transfer Active (RTACT)
– Write Transfer Active (WTACT)
– DAT Line Active (DATLL)
– Command Inhibit (DAT) (CMDINHD)

“SDMMC Block Gap Control Register (SD_SDIO)”

– Continue Request (CONTR)
– Stop At Block Gap Request (STPBGR)

“SDMMC Normal Interrupt Status Register (SD_SDIO)”

– Buffer Read Ready (BRDRDY)
– Buffer Write Ready (BWRRDY)
– DMA Interrupt (DMAINT)

– Block Gap Event (BLKGE)
– Transfer Complete (TRFC)

0: Work

1: Reset

### 48.13.19 SDMMC Normal Interrupt Status Register (SD_SDIO)

**Name:** SDMMC_NISTR (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ERRINT | – | – | – | – | – | – | CINT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CREM | CINS | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete**

This bit is set when getting the end bit of the command response. Auto CMD12 and Auto CMD23 consist of two responses. Command Complete is not generated by the response of CMD12 or CMD23, but it is generated by the response of a read/write command. Refer to Command Inhibit (CMD) in SDMMC_PSR for details on how to control this bit.

This bit can only be set to 1 if SDMMC_NISTER.CMDC is set to 1. An interrupt can only be generated if SDMMC_NISIER.CMDC is set to 1.

Writing this bit to 1 clears this bit.

The table below shows that Command Timeout Error (CMDTEO) has a higher priority than Command Complete (CMDC). If both bits are set to 1, it can be considered that the response was not received correctly.

| CMDC | CMDTEO | Meaning of the status |
|---|---|---|
| 0 | 0 | Interrupted by another factor |
| Don’t care | 1 | Response not received within 64 SDCLK cycles |
| 1 | 0 | Response received |

0: No command complete

1: Command complete

- **TRFC: Transfer Complete**

This bit is set when a read/write transfer and a command with Busy is completed.

In the case of a Read Transaction:

This bit is set at the falling edge of the Read Transfer Active Status. The interrupt is generated in two cases. The first is when a data transfer is completed as specified by the data length (after the last data has been read to the system). The second is when data has stopped at the block gap and completed the data transfer by setting the Stop At Block Gap Request (STPBGR) in SDMMC_BGCR (after valid data has been read to the system). Refer to section “Read Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

In the case of a Write Transaction:

This bit is set at the falling edge of the DAT Line Active (DLACT) status. This interrupt is generated in two cases. The first is when the last data is written to the card as specified by the data length and the Busy signal is released. The second is when data transfers are stopped at the block gap by setting Stop At Block Gap Request (STPBGR) in SDMMC_BGCR and data transfers are completed. (After valid data is written to the card and the Busy signal is released). Refer to section “Write Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

In the case of command with Busy:

This bit is set when Busy is de-asserted. Refer to DAT Line Active (DLACT) and Command Inhibit (DAT) (CMDINHD) in SDMMC_PSR.

This bit can only be set to 1 if SDMMC_NISTER.TRFC is set to 1. An interrupt can only be generated if SDMMC_NISIER.TRFC is set to 1.

Writing this bit to 1 clears this bit.

The table below shows that Transfer Complete (TRFC) has a higher priority than Data Timeout Error (DATTEO). If both bits are set to 1, execution of a command can be considered to be completed.

| TRFC | DATTEO | Meaning of the status |
|---|---|---|
| 0 | 0 | Interrupted by another factor |
| 0 | 1 | Timeout occurred during transfer |
| 1 | Don't Care | Command execution complete |

0: Command execution is not complete.

1: Command execution is complete.

- **BLKGE: Block Gap Event**

If the Stop At Block Gap Request (STPBGR) in SDMMC_BGCR is set to 1, this bit is set when either a read or a write transaction is stopped at a block gap. If STPBGR is not set to 1, this bit is not set to 1.

In the case of a Read transaction:

This bit is set at the falling edge of the DAT Line Active (DLACT) status (when the transaction is stopped at SD bus timing). The Read Wait must be supported in order to use this function. Refer to section "Read Transaction Wait / Continue Timing" in the "SD Host Controller Simplified Specification V3.00" about the detailed timing.

In the case of a Write transaction:

This bit is set at the falling edge of the Write Transfer Active (WTACT) status (after getting the CRC status at SD bus timing). Refer to section "Write Transaction Wait / Continue Timing" in the "SD Host Controller Simplified Specification V3.00" for more details on the sequence of events.

This bit can only be set to 1 if SDMMC_NISTER.BLKGE is set to 1. An interrupt can only be generated if SDMMC_NISIER.BLKGE is set to 1.

Writing this bit to 1 clears this bit.

0: No block gap event

1: Transaction stopped at block gap

- **DMAINT: DMA Interrupt**

This status is set if the SDMMC detects the Host SDMA Buffer boundary during transfer. Refer to SDMA Buffer Boundary (BOUNDARY) in SDMMC_BSR.

In case of ADMA, by setting the "int" field in the descriptor table, the SDMMC raises this status flag when the descriptor line is completed. This status flag does not rise after Transfer Complete (TRFC).

This bit can only be set to 1 if SDMMC_NISTER.DMAINT is set to 1. An interrupt can only be generated if SDMMC_NISIER.DMAINT is set to 1.

Writing this bit to 1 clears this bit.

0: No DMA Interrupt

1: DMA Interrupt

- **BWRRDY: Buffer Write Ready**

This status is set to 1 if the Buffer Write Enable (BUFWREN) changes from 0 to 1. Refer to BUFWREN in SDMMC_PSR.

This bit can only be set to 1 if SDMMC_NISTER.BWRRDY is set to 1. An interrupt can only be generated if SDMMC_NISIER.BWRRDY is set to 1.

Writing this bit to 1 clears this bit.

0: Not ready to write buffer

1: Ready to write buffer

- **BRDRDY: Buffer Read Ready**

This status is set to 1 if the Buffer Read Enable (BUFRDEN) changes from 0 to 1. Refer to BUFRDEN in SDMMC_PSR. While processing the tuning procedure (Execute Tuning (EXTUN) in SDMMC_HC2R is set to 1), BRDRDY is set to 1 for every CMD19 execution.

This bit can only be set to 1 if SDMMC_NISTER.BRDRDY is set to 1. An interrupt can only be generated if SDMMC_NISIER.BRDRDY is set to 1.

Writing this bit to 1 clears this bit.

0: Not ready to read buffer

1: Ready to read buffer

- **CINS: Card Insertion**

This status is set if Card Inserted (CARDINS) in SDMMC_PSR changes from 0 to 1. When the user writes this bit to 1 to clear this status, the status of SDMMC_PSR.CARDINS must be confirmed because the card detect state may possibly be changed when the user clears this bit and no interrupt event can be generated.

This bit can only be set to 1 if SDMMC_NISTER.CINS is set to 1. An interrupt can only be generated if SDMMC_NISIER.CINS is set to 1.

Writing this bit to 1 clears this bit.

0: Card state unstable or card removed

1: Card inserted

- **CREM: Card Removal**

This status is set to 1 if Card Inserted (CARDINS) in SDMMC_PSR changes from 1 to 0. When the user writes this bit to 1 to clear this status, the status of SDMMC_PSR.CARDINS must be confirmed because the card detect state may possibly be changed when the user clears this bit and no interrupt event can be generated.

This bit can only be set to 1 if SDMMC_NISTER.CREM is set to 1. An interrupt can only be generated if SDMMC_NISIER.CREM is set to 1.

Writing this bit to 1 clears this bit.

0: Card state unstable or card inserted

1: Card removed

- **CINT: Card Interrupt**

Writing this bit to 1 does not clear this bit. It is cleared by resetting the SD card interrupt factor. In 1-bit mode, the SDMMC detects the Card Interrupt without SDCLK to support wakeup. In 4-bit mode, the Card Interrupt signal is sampled during the interrupt cycle, so there are some sample delays between the interrupt signal from the SD card and the interrupt to the system.

When this bit has been set to 1 and the user needs to start this interrupt service, Card Interrupt Status Enable (CINT) in SDMMC_NISTER may be set to 0 in order to clear the card interrupt statuses latched in the SDMMC and to stop driving

the interrupt signal to the system. After completion of the card interrupt service (it should reset interrupt factors in the SD card and the interrupt signal may not be asserted), set SDMMC_NISTER.CINT to 1 and start sampling the interrupt signal again.

Interrupt detected by DAT[1] is supported when there is one card per slot.

This bit can only be set to 1 if SDMMC_NISTER.CREM is set to 1. An interrupt can only be generated if SDMMC_NISIER.CREM is set to 1.

0: No card interrupt

1: Card interrupt

- **ERRINT: Error Interrupt**

If any of the bits in SDMMC_EISTR are set, then this bit is set. Therefore, the user can efficiently test for an error by checking this bit first. This bit is read-only.

0: No error

1: Error

### 48.13.20 SDMMC Normal Interrupt Status Register (eMMC)

**Name:** SDMMC_NISTR (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ERRINT | BOOTAR | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete**

This bit is set when getting the end bit of the command response. Auto CMD12 and Auto CMD23 consist of two responses. Command Complete is not generated by the response of CMD12 or CMD23, but it is generated by the response of a read/write command. Refer to CMRINHC in SDMMC_PSR for details on how to control this bit.

This bit can only be set to 1 if SDMMC_NISTER.CMDC is set to 1. An interrupt can only be generated if SDMMC_NISIER.CMDC is set to 1.

Writing this bit to 1 clears this bit.

The table below shows that Command Timeout Error (CMDTEO) has a higher priority than Command Complete (CMDC). If both bits are set to 1, it can be considered that the response was not received correctly.

| CMDC | CMDTEO | Meaning of the status |
|---|---|---|
| 0 | 0 | Interrupted by another factor |
| Don’t care | 1 | Response not received within 64 SDCLK cycles |
| 1 | 0 | Response received |

0: No command complete

1: Command complete

- **TRFC: Transfer Complete**

This bit is set when a read/write transfer and a command with Busy is completed.

In the case of a Read Transaction:

This bit is set at the falling edge of the Read Transfer Active Status. The interrupt is generated in two cases. The first is when a data transfer is completed as specified by the data length (after the last data has been read to the system). The second is when data has stopped at the block gap and completed the data transfer by setting the Stop At Block Gap Request (STPBGR) in SDMMC_BGCR (after valid data has been read to the system). Refer to section “Read Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

In the case of a Write Transaction:

This bit is set at the falling edge of the DAT Line Active (DLACT) status. This interrupt is generated in two cases. The first is when the last data is written to the card as specified by the data length and the Busy signal is released. The second is when data transfers are stopped at the block gap by setting Stop At Block Gap Request (STPBGR) in SDMMC_BGCR and data transfers are completed. (After valid data is written to the card and the Busy signal is released). Refer to section “Write Transaction Wait / Continue Timing” in the “SD Host Controller Simplified Specification V3.00” for more details on the sequence of events.

In the case of command with Busy:

This bit is set when Busy is de-asserted. Refer to DAT Line Active (DLACT) and Command Inhibit (DAT) (CMDINHD) in SDMMC_PSR.

This bit can only be set to 1 if SDMMC_NISTER.TRFC is set to 1. An interrupt can only be generated if SDMMC_NISIER.TRFC is set to 1.

Writing this bit to 1 clears this bit.

The table below shows that Transfer Complete (TRFC) has a higher priority than Data Timeout Error (DATTEO). If both bits are set to 1, execution of a command can be considered to be completed.

| TRFC | DATTEO | Meaning of the status |
|---|---|---|
| 0 | 0 | Interrupted by another factor |
| 0 | 1 | Timeout occurred during transfer |
| 1 | Don't Care | Command execution complete |

0: Command execution is not complete.

1: Command execution is complete.

- **BLKGE: Block Gap Event**

If the Stop At Block Gap Request (STPBGR) in SDMMC_BGCR is set to 1, this bit is set when either a read or a write transaction is stopped at a block gap. If STPBGR is not set to 1, this bit is not set to 1.

In the case of a Read transaction:

This bit is set at the falling edge of the DAT Line Active (DLACT) status (when the transaction is stopped at SD bus timing). The Read Wait must be supported in order to use this function. Refer to section "Read Transaction Wait / Continue Timing" in the "SD Host Controller Simplified Specification V3.00" about the detailed timing.

In the case of a Write transaction:

This bit is set at the falling edge of the Write Transfer Active (WTACT) status (after getting the CRC status at SD bus timing). Refer to section "Write Transaction Wait / Continue Timing" in the "SD Host Controller Simplified Specification V3.00" for more details on the sequence of events.

This bit can only be set to 1 if SDMMC_NISTER.BLKGE is set to 1. An interrupt can only be generated if SDMMC_NISIER.BLKGE is set to 1.

Writing this bit to 1 clears this bit.

0: No block gap event

1: Transaction stopped at block gap

- **DMAINT: DMA Interrupt**

This status is set if the SDMMC detects the Host SDMA Buffer boundary during transfer. Refer to SDMA Buffer Boundary (BOUNDARY) in SDMMC_BSR.

In case of ADMA, by setting "int" field in the descriptor table, the SDMMC raises this status flag when the descriptor line is completed. This status flag does not rise after the Transfer Complete (TRFC).

This bit can only be set to 1 if SDMMC_NISTER.DMAINT is set to 1. An interrupt can only be generated if SDMMC_NISIER.DMAINT is set to 1.

Writing this bit to 1 clears this bit.

0: No DMA interrupt

1: DMA interrupt

- **BWRRDY: Buffer Write Ready**

This status is set to 1 if Buffer Write Enable (BUFWREN) changes from 0 to 1. Refer to Buffer Write Enable (BUFWREN) in SDMMC_PSR.

This bit can only be set to 1 if SDMMC_NISTER.BWRRDY is set to 1. An interrupt can only be generated if SDMMC_NISIER.BWRRDY is set to 1.

Writing this bit to 1 clears this bit.

0: Not ready to write buffer

1: Ready to write buffer

- **BRDRDY: Buffer Read Ready**

This status is set to 1 if Buffer Read Enable (BUFRDEN) changes from 0 to 1. Refer to Buffer Read Enable (BUFRDEN) in SDMMC_PSR. While processing the tuning procedure (Execute Tuning (EXTUN) in SDMMC_HC2R is set to 1), BRDRDY is set to 1 for every CMD19 execution.

This bit can only be set to 1 if SDMMC_NISTER.BRDRDY is set to 1. An interrupt can only be generated if SDMMC_NISIER.BRDRDY is set to 1.

Writing this bit to 1 clears this bit.

0: Not ready to read buffer

1: Ready to read buffer

- **BOOTAR: Boot Acknowledge Received**

This bit is set to 1 when the SDMMC received a Boot Acknowledge pattern from the e.MMC.

This bit can only be set to 1 if SDMMC_NISTER.BOOTAR is set to 1. An interrupt can only be generated if SDMMC_NISIER.BOOTAR is set to 1.

Writing this bit to 1 clears this bit.

0: Boot Acknowledge pattern not received.

1: Boot Acknowledge pattern received.

- **ERRINT: Error Interrupt**

If any of the bits in SDMMC_EISTR are set, then this bit is set. Therefore, the user can efficiently test for an error by checking this bit first. This bit is read only.

0: No error

1: Error

### 48.13.21 SDMMC Error Interrupt Status Register (SD_SDIO)

**Name:** SDMMC_EISTR (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error**

This bit is set to 1 only if no response is returned within 64 SDCLK cycles from the end bit of the command. If the SDMMC detects a CMD line conflict, in which case Command CRC Error (CMDCRC) is also set to 1 as shown in Table 48-5, this bit is set without waiting for 64 SDCLK cycles because the command is aborted by the SDMMC.

This bit can only be set to 1 if SDMMC_EISTER.CMDTEO is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDTEO is set to 1.

Writing this bit to 1 clears this bit.

**Table 48-5. Relations between CMDCRC and CMDTEO**

| CMDCRC | CMDTEO | Types of error |
|---|---|---|
| 0 | 0 | No error |
| 0 | 1 | Response timeout error |
| 1 | 0 | Response CRC error |
| 1 | 1 | CMD line conflict |

- **CMDCRC: Command CRC Error**

The Command CRC Error is generated in two cases.

If a response is returned and the Command Timeout Error (CMDTEO) is set to 0 (indicating no command timeout), this bit is set to 1 when detecting a CRC error in the command response.

The SDMMC detects a CMD line conflict by monitoring the CMD line when a command is issued. If the SDMMC drives the CMD line to 1 level, but detects 0 level on the CMD line at the next SDCLK edge, then the SDMMC aborts the command (stops driving the CMD line) and sets this bit to 1. CMDTEO is also set to 1 to indicate a CMD line conflict (refer to Table 48-5).

This bit can only be set to 1 if SDMMC_EISTER.CMDCRC is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDCRC is set to 1.

Writing this bit to 1 clears this bit.

- **CMDEND: Command End Bit Error**

This bit is set to 1 when detecting that the end bit of a command response is 0.

This bit can only be set to 1 if SDMMC_EISTER.CMDEND is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDEND is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **CMDIDX: Command Index Error**

This bit is set to 1 if a Command Index error occurs in the command response.

This bit can only be set to 1 if SDMMC_EISTER.CMDIDX is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDIDX is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATTEO: Data Timeout Error**

This bit is set to 1 when detecting one of following timeout conditions.

  – Busy timeout for R1b, R5b response type (see “Physical Layer Simplified Specification V3.01” and “SDIO Simplified Specification V3.00”).

  – Busy timeout after Write CRC status.

  – Write CRC Status timeout.

  – Read data timeout

This bit can only be set to 1 if SDMMC_EISTER.DATTEO is set to 1. An interrupt can only be generated if SDMMC_EISIER.DATTEO is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATCRC: Data CRC error**

This bit is set to 1 when detecting a CRC error when transferring read data which uses the DAT line or when detecting that the Write CRC Status has a value other than “010”.

This bit can only be set to 1 if SDMMC_EISTER.DATCRC is set to 1. An interrupt can only be generated if SDMMC_EISIER.DATCRC is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATEND: Data End Bit Error**

This bit is set to 1 either when detecting 0 at the end bit position of read data which uses the DAT line or at the end bit position of the CRC Status.

This bit can only be set to 1 if SDMMC_EISTER.DATEND is set to 1. An interrupt can only be generated if SDMMC_EISIER.DATEND is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **CURLIM: Current Limit Error**

By setting SD Bus Power (SDBPWR) in SDMMC_PSR, the SDMMC is requested to supply power for the SD Bus. The SDMMC is protected from an illegal card by stopping power supply to the card, in which case this bit indicates a failure status. Reading 1 means the SDMMC is not supplying power to the card due to some failure. Reading 0 means that the SDMMC is supplying power and no error has occurred. The SDMMC may require some sampling time to detect the current limit.

This bit can only be set to 1 if SDMMC_EISTER.CURLIM is set to 1. An interrupt can only be generated if SDMMC_EISIER.CURLIM is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **ACMD: Auto CMD Error**

Auto CMD12 and Auto CMD23 use this error status. This bit is set to 1 when detecting that one of the 0 to 4 bits in SDMMC_AESR (SDMMC_ACESR[4:0]) has changed from 0 to 1. In the case of Auto CMD12, this bit is set to 1, not only when errors occur in Auto CMD12 but also when auto CMD12 is not executed due to the previous command error.

This bit can only be set to 1 if SDMMC_EISTER.ACMD is set to 1. An interrupt can only be generated if SDMMC_EISIER.ACMD is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **ADMA: ADMA Error**

This bit is set to 1 when the SDMMC detects errors during an ADMA-based data transfer. The state of the ADMA at an error occurrence is saved in SDMMC_AESR.

In addition, the SDMMC raises this status flag when it detects some invalid description data (Valid = 0) at the ST_FDS state (refer to section “Advanced DMA” in the “SD Host Controller Simplified Specification V3.00”). ADMA Error Status (ERRST) in SDMMC_AESR indicates that an error occurred in ST_FDS state. The user may find that the Valid bit is not set at the error descriptor.

This bit can only be set to 1 if SDMMC_EISTER.ADMA is set to 1. An interrupt can only be generated if SDMMC_EISIER.ADMA is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

### 48.13.22 SDMMC Error Interrupt Status Register (eMMC)

**Name:** SDMMC_EISTR (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOOTAE | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error**

This bit is set to 1 only if no response is returned within 64 SDCLK cycles from the end bit of the command. If the SDMMC detects a CMD line conflict, in which case Command CRC Error (CMDCRC) is also set to 1 as shown in Table 48-5, this bit is set without waiting for 64 SDCLK cycles because the command is aborted by the SDMMC.

This bit can only be set to 1 if SDMMC_EISTER.CMDTEO is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDTEO is set to 1.

Writing this bit to 1 clears this bit.

**Table 48-6. Relations between CMDCRC and CMDTEO**

| CMDCRC | CMDTEO | Types of error |
|---|---|---|
| 0 | 0 | No error |
| 0 | 1 | Response timeout error |
| 1 | 0 | Response CRC error |
| 1 | 1 | CMD line conflict |

- **CMDCRC: Command CRC Error**

The Command CRC Error is generated in two cases.

If a response is returned and Command Timeout Error (CMDTEO) is set to 0 (indicating no command timeout), this bit is set to 1 when detecting a CRC error in the command response.

The SDMMC detects a CMD line conflict by monitoring the CMD line when a command is issued. If the SDMMC drives the CMD line to 1 level, but detects 0 level on the CMD line at the next SDCLK edge, then the SDMMC aborts the command (stops driving the CMD line) and sets this bit to 1. CMDTEO is also set to 1 to indicate a CMD line conflict (refer to Table 48-5).

This bit can only be set to 1 if SDMMC_EISTER.CMDCRC is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDCRC is set to 1.

Writing this bit to 1 clears this bit.

- **CMDEND: Command End Bit Error**

This bit is set to 1 when detecting that the end bit of a command response is 0.

This bit can only be set to 1 if SDMMC_EISTER.CMDEND is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDEND is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **CMDIDX: Command Index Error**

This bit is set to 1 if a Command Index error occurs in the command response.

This bit can only be set to 1 if SDMMC_EISTER.CMDIDX is set to 1. An interrupt can only be generated if SDMMC_EISIER.CMDIDX is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATTEO: Data Timeout error**

This bit is set to 1 when detecting one of following timeout conditions.

  – Busy timeout for R1b, R5b response type (see “Physical Layer Simplified Specification V3.01” and “SDIO Simplified Specification V3.00”).

  – Busy timeout after Write CRC Status.

  – Write CRC Status timeout.

  – Read data timeout

This bit can only be set to 1 if SDMMC_EISTER.DATTEO is set to 1. An interrupt can only be generated if SDMMC_EISIER.DATTEO is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATCRC: Data CRC Error**

This bit is set to 1 when detecting a CRC error during a transfer of read data which uses the DAT line or when detecting that the Write CRC Status has a value other than “010”.

This bit can only be set to 1 if SDMMC_EISTER.DATCRC is set to 1. An interrupt can only be generated if SDMMC_EISIER. DATCRC is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **DATEND: Data End Bit Error**

This bit is set to 1 either when detecting 0 at the end bit position of read data which uses the DAT line or at the end bit position of the CRC Status.

This bit can only be set to 1 if SDMMC_EISTER.DATEND is set to 1. An interrupt can only be generated if SDMMC_EISIER.DATEND is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **CURLIM: Current Limit Error**

By setting SD Bus Power (SDBPWR) in SDMMC_PSR, the SDMMC is requested to supply power for the SD Bus. The SDMMC is protected from an illegal card by stopping power supply to the card, in which case this bit indicates a failure status. Reading 1 means the SDMMC is not supplying power to the card due to some failure. Reading 0 means that the SDMMC is supplying power and no error has occurred. The SDMMC may require some sampling time to detect the current limit.

This bit can only be set to 1 if SDMMC_EISTER.CURLIM is set to 1. An interrupt can only be generated if SDMMC_EISIER.CURLIM is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **ACMD: Auto CMD Error**

Auto CMD12 and Auto CMD23 use this error status. This bit is set to 1 when detecting that one of the 0 to 4 bits in SDMMC_AESR (SDMMC_ACESR[4:0]) has changed from 0 to 1. In the case of Auto CMD12, this bit is set to 1, not only when errors occur in Auto CMD12, but also when Auto CMD12 is not executed due to the previous command error.

This bit can only be set to 1 if SDMMC_EISTER.ACMD is set to 1. An interrupt can only be generated if SDMMC_EISIER.ACMD is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **ADMA: ADMA Error**

This bit is set to 1 when the SDMMC detects errors during an ADMA-based data transfer. The state of the ADMA at an error occurrence is saved in SDMMC_AESR.

In addition, the SDMMC raises this status flag when it detects some invalid description data (Valid = 0) at the ST_FDS state (refer to section “Advanced DMA” in the “SD Host Controller Simplified Specification V3.00”). ADMA Error Status (ERRST) in SDMMC_AESR indicates that an error occured in ST_FDS state. The user may find that the Valid bit is not set at the error descriptor.

This bit can only be set to 1 if SDMMC_EISTER.ADMA is set to 1. An interrupt can only be generated if SDMMC_EISIER.ADMA is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

- **BOOTAE: Boot Acknowledge Error**

This bit is set to 1 when detecting that the e.MMC Boot Acknowledge Status has a value other than “010”.

This bit can only be set to 1 if SDMMC_EISTER.BOOTAE is set to 1. An interrupt can only be generated if SDMMC_EISIER.BOOTAE is set to 1.

Writing this bit to 1 clears this bit.

0: No error

1: Error

### 48.13.23 SDMMC Normal Interrupt Status Enable Register (SD_SDIO)

**Name:** SDMMC_NISTER (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | CINT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CREM | CINS | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete Status Enable**

0 (MASKED): The CMDC status flag in SDMMC_NISTR is masked.

1 (ENABLED): The CMDC status flag in SDMMC_NISTR is enabled.

- **TRFC: Transfer Complete Status Enable**

0 (MASKED): The TRFC status flag in SDMMC_NISTR is masked.

1 (ENABLED): The TRFC status flag in SDMMC_NISTR is enabled.

- **BLKGE: Block Gap Event Status Enable**

0 (MASKED): The BLKGE status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BLKGE status flag in SDMMC_NISTR is enabled.

- **DMAINT: DMA Interrupt Status Enable**

0 (MASKED): The DMAINT status flag in SDMMC_NISTR is masked.

1 (ENABLED): The DMAINT status flag in SDMMC_NISTR is enabled.

- **BWRRDY: Buffer Write Ready Status Enable**

0 (MASKED): The BWRRDY status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BWRRDY status flag in SDMMC_NISTR is enabled.

- **BRDRDY: Buffer Read Ready Status Enable**

0 (MASKED): The BRDRDY status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BRDRDY status flag in SDMMC_NISTR is enabled.

- **CINS: Card Insertion Status Enable**

0 (MASKED): The CINS status flag in SDMMC_NISTR is masked.

1 (ENABLED): The CINS status flag in SDMMC_NISTR is enabled.

- **CREM: Card Removal Status Enable**

0 (MASKED): The CREM status flag in SDMMC_NISTR is masked.

1 (ENABLED): The CREM status flag in SDMMC_NISTR is enabled.

- **CINT: Card Interrupt Status Enable**

If this bit is set to 0, the SDMMC clears interrupt requests to the system. The Card Interrupt detection is stopped when this bit is cleared and restarted when this bit is set to 1. The user may clear this bit before servicing the Card Interrupt and may set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

0 (MASKED): The CINT status flag in SDMMC_NISTR is masked.

1 (ENABLED): The CINT status flag in SDMMC_NISTR is enabled.

### 48.13.24 SDMMC Normal Interrupt Status Enable Register (eMMC)

**Name:** SDMMC_NISTER (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | BOOTAR | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete Status Enable**

0 (MASKED): The CMDC status flag in SDMMC_NISTR is masked.

1 (ENABLED): The CMDC status flag in SDMMC_NISTR is enabled.

- **TRFC: Transfer Complete Status Enable**

0 (MASKED): The TRFC status flag in SDMMC_NISTR is masked.

1 (ENABLED): The TRFC status flag in SDMMC_NISTR is enabled.

- **BLKGE: Block Gap Event Status Enable**

0 (MASKED): The BLKGE status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BLKGE status flag in SDMMC_NISTR is enabled.

- **DMAINT: DMA Interrupt Status Enable**

0 (MASKED): The DMAINT status flag in SDMMC_NISTR is masked.

1 (ENABLED): The DMAINT status flag in SDMMC_NISTR is enabled.

- **BWRRDY: Buffer Write Ready Status Enable**

0 (MASKED): The BWRRDY status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BWRRDY status flag in SDMMC_NISTR is enabled.

- **BRDRDY: Buffer Read Ready Status Enable**

0 (MASKED): The BRDRDY status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BRDRDY status flag in SDMMC_NISTR is enabled.

- **BOOTAR: Boot Acknowledge Received Status Enable**

0 (MASKED): The BOOTAR status flag in SDMMC_NISTR is masked.

1 (ENABLED): The BOOTAR status flag in SDMMC_NISTR is enabled.

### 48.13.25 SDMMC Error Interrupt Status Enable Register (SD_SDIO)

**Name:** SDMMC_EISTER (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error Status Enable**

0 (MASKED): The CMDTEO status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDTEO status flag in SDMMC_EISTR is enabled.

- **CMDCRC: Command CRC Error Status Enable**

0 (MASKED): The CMDCRC status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDCRC status flag in SDMMC_EISTR is enabled.

- **CMDEND: Command End Bit Error Status Enable**

0 (MASKED): The CMDEND status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDEND status flag in SDMMC_EISTR is enabled.

- **CMDIDX: Command Index Error Status Enable**

0 (MASKED): The CMDIDX status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDIDX status flag in SDMMC_EISTR is enabled.

- **DATTEO: Data Timeout Error Status Enable**

0 (MASKED): The DATTEO status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATTEO status flag in SDMMC_EISTR is enabled.

- **DATCRC: Data CRC Error Status Enable**

0 (MASKED): The DATCRC status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATCRC status flag in SDMMC_EISTR is enabled.

- **DATEND: Data End Bit Error Status Enable**

0 (MASKED): The DATEND status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATEND status flag in SDMMC_EISTR is enabled.

- **CURLIM: Current Limit Error Status Enable**

0 (MASKED): The CURLIM status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CURLIM status flag in SDMMC_EISTR is enabled.

- **ACMD: Auto CMD Error Status Enable**

0 (MASKED): The ACMD status flag in SDMMC_EISTR is masked.

1 (ENABLED): The ACMD status flag in SDMMC_EISTR is enabled.

- **ADMA: ADMA Error Status Enable**

0 (MASKED): The ADMA status flag in SDMMC_EISTR is masked.

1 (ENABLED): The ADMA status flag in SDMMC_EISTR is enabled.

### 48.13.26 SDMMC Error Interrupt Status Enable Register (eMMC)

**Name:** SDMMC_EISTER (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOOTAE | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error Status Enable**

0 (MASKED): The CMDTEO status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDTEO status flag in SDMMC_EISTR is enabled.

- **CMDCRC: Command CRC Error Status Enable**

0 (MASKED): The CMDCRC status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDCRC status flag in SDMMC_EISTR is enabled.

- **CMDEND: Command End Bit Error Status Enable**

0 (MASKED): The CMDEND status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDEND status flag in SDMMC_EISTR is enabled.

- **CMDIDX: Command Index Error Status Enable**

0 (MASKED): The CMDIDX status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CMDIDX status flag in SDMMC_EISTR is enabled.

- **DATTEO: Data Timeout Error Status Enable**

0 (MASKED): The DATTEO status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATTEO status flag in SDMMC_EISTR is enabled.

- **DATCRC: Data CRC Error Status Enable**

0 (MASKED): The DATCRC status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATCRC status flag in SDMMC_EISTR is enabled.

- **DATEND: Data End Bit Error Status Enable**

0 (MASKED): The DATEND status flag in SDMMC_EISTR is masked.

1 (ENABLED): The DATEND status flag in SDMMC_EISTR is enabled.

- **CURLIM: Current Limit Error Status Enable**

0 (MASKED): The CURLIM status flag in SDMMC_EISTR is masked.

1 (ENABLED): The CURLIM status flag in SDMMC_EISTR is enabled.

- **ACMD: Auto CMD Error Status Enable**

0 (MASKED): The ACMD status flag in SDMMC_EISTR is masked.

1 (ENABLED): The ACMD status flag in SDMMC_EISTR is enabled.

- **ADMA: ADMA Error Status Enable**

0 (MASKED): The ADMA status flag in SDMMC_EISTR is masked.

1 (ENABLED): The ADMA status flag in SDMMC_EISTR is enabled.

- **BOOTAE: Boot Acknowledge Error Status Enable**

0 (MASKED): The BOOTAE status flag in SDMMC_EISTR is masked.

1 (ENABLED): The BOOTAE status flag in SDMMC_EISTR is enabled.

### 48.13.27 SDMMC Normal Interrupt Signal Enable Register (SD_SDIO)

**Name:** SDMMC_NISIER (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | CINT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CREM | CINS | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete Signal Enable**

0 (MASKED): No interrupt is generated when the CMDC status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the CMDC status rises in SDMMC_NISTR.

- **TRFC: Transfer Complete Signal Enable**

0 (MASKED): No interrupt is generated when the TRFC status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the TRFC status rises in SDMMC_NISTR.

- **BLKGE: Block Gap Event Signal Enable**

0 (MASKED): No interrupt is generated when the BLKGE status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BLKGE status rises in SDMMC_NISTR.

- **DMAINT: DMA Interrupt Signal Enable**

0 (MASKED): No interrupt is generated when the DMAINT status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the DMAINT status rises in SDMMC_NISTR.

- **BWRRDY: Buffer Write Ready Signal Enable**

0 (MASKED): No interrupt is generated when the BWRRDY status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BWRRDY status rises in SDMMC_NISTR.

- **BRDRDY: Buffer Read Ready Signal Enable**

0 (MASKED): No interrupt is generated when the BRDRDY status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BRDRDY status rises in SDMMC_NISTR.

- **CINS: Card Insertion Signal Enable**

0 (MASKED): No interrupt is generated when the CINS status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the CINS status rises in SDMMC_NISTR.

- **CREM: Card Removal Signal Enable**

0 (MASKED): No interrupt is generated when the CREM status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the CREM status rises in SDMMC_NISTR.

- **CINT: Card Interrupt Signal Enable**

0 (MASKED): No interrupt is generated when the CINT status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the CINT status rises in SDMMC_NISTR.

### 48.13.28 SDMMC Normal Interrupt Signal Enable Register (eMMC)

**Name:** SDMMC_NISIER (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | BOOTAR | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | BRDRDY | BWRRDY | DMAINT | BLKGE | TRFC | CMDC |

- **CMDC: Command Complete Signal Enable**

0 (MASKED): No interrupt is generated when the CMDC status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the CMDC status rises in SDMMC_NISTR.

- **TRFC: Transfer Complete Signal Enable**

0 (MASKED): No interrupt is generated when the TRFC status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the TRFC status rises in SDMMC_NISTR.

- **BLKGE: Block Gap Event Signal Enable**

0 (MASKED): No interrupt is generated when the BLKGE status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BLKGE status rises in SDMMC_NISTR.

- **DMAINT: DMA Interrupt Signal Enable**

0 (MASKED): No interrupt is generated when the DMAINT status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the DMAINT status rises in SDMMC_NISTR.

- **BWRRDY: Buffer Write Ready Signal Enable**

0 (MASKED): No interrupt is generated when the BWRRDY status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BWRRDY status rises in SDMMC_NISTR.

- **BRDRDY: Buffer Read Ready Signal Enable**

0 (MASKED): No interrupt is generated when the BRDRDY status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BRDRDY status rises in SDMMC_NISTR.

- **BOOTAR: Boot Acknowledge Received Signal Enable**

0 (MASKED): No interrupt is generated when the BOOTAR status rises in SDMMC_NISTR.

1 (ENABLED): An interrupt is generated when the BOOTAR status rises in SDMMC_NISTR.

### 48.13.29 SDMMC Error Interrupt Signal Enable Register (SD_SDIO)

**Name:** SDMMC_EISIER (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDTEO status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDTEO status rises in SDMMC_EISTR.

- **CMDCRC: Command CRC Error Signal Enable**

0 (MASKED): No interrupt is generated when the CDMCRC status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDCRC status rises in SDMMC_EISTR.

- **CMDEND: Command End Bit Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDEND status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDEND status rises in SDMMC_EISTR.

- **CMDIDX: Command Index Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDIDX status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDIDX status rises in SDMMC_EISTR.

- **DATTEO: Data Timeout Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATTEO status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATTEO status rises in SDMMC_EISTR.

- **DATCRC: Data CRC Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATCRC status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATCRC status rises in SDMMC_EISTR.

- **DATEND: Data End Bit Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATEND status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATEND status rises in SDMMC_EISTR.

- **CURLIM: Current Limit Error Signal Enable**

0 (MASKED): No interrupt is generated when the CURLIM status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CURLIM status rises in SDMMC_EISTR.

- **ACMD: Auto CMD Error Signal Enable**

0 (MASKED): No interrupt is generated when the ACMD status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the ACMD status rises in SDMMC_EISTR.

- **ADMA: ADMA Error Signal Enable**

0 (MASKED): No interrupt is generated when the ADMA status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the ADMA status rises in SDMMC_EISTR.

### 48.13.30 SDMMC Error Interrupt Signal Enable Register (eMMC)

**Name:** SDMMC_EISIER (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOOTAE | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Command Timeout Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDTEO status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDTEO status rises in SDMMC_EISTR.

- **CMDCRC: Command CRC Error Signal Enable**

0 (MASKED): No interrupt is generated when the CDMCRC status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDCRC status rises in SDMMC_EISTR.

- **CMDEND: Command End Bit Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDEND status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDEND status rises in SDMMC_EISTR.

- **CMDIDX: Command Index Error Signal Enable**

0 (MASKED): No interrupt is generated when the CMDIDX status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CMDIDX status rises in SDMMC_EISTR.

- **DATTEO: Data Timeout Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATTEO status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATTEO status rises in SDMMC_EISTR.

- **DATCRC: Data CRC Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATCRC status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATCRC status rises in SDMMC_EISTR.

- **DATEND: Data End Bit Error Signal Enable**

0 (MASKED): No interrupt is generated when the DATEND status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the DATEND status rises in SDMMC_EISTR.

- **CURLIM: Current Limit Error Signal Enable**

0 (MASKED): No interrupt is generated when the CURLIM status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the CURLIM status rises in SDMMC_EISTR.

- **ACMD: Auto CMD Error Signal Enable**

0 (MASKED): No interrupt is generated when the ACMD status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the ACMD status rises in SDMMC_EISTR.

- **ADMA: ADMA Error Signal Enable**

0 (MASKED): No interrupt is generated when the ADMA status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the ADMA status rises in SDMMC_EISTR.

- **BOOTAE: Boot Acknowledge Error Signal Enable**

0 (MASKED): No interrupt is generated when the BOOTAE status rises in SDMMC_EISTR.

1 (ENABLED): An interrupt is generated when the BOOTAE status rises in SDMMC_EISTR.

### 48.13.31 SDMMC Auto CMD Error Status Register

**Name:** SDMMC_ACESR

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CMDNI | – | – | ACMDIDX | ACMDEND | ACMDCRC | ACMDTEO | ACMD12NE |

- **ACMD12NE: Auto CMD12 Not Executed**

If a memory multiple block data transfer is not started due to a command error, this bit is not set to 1 because it is not necessary to issue Auto CMD12. Setting this bit to 1 means the SDMMC cannot issue Auto CMD12 to stop a memory multiple block data transfer due to some error. If this bit is set to 1, other error status bits (SDMMC_ACESR[4:1]) are meaningless.

This bit is set to 0 when an Auto CMD error is generated by Auto CMD23.

0: No error

1: Error

- **ACMDTEO: Auto CMD Timeout Error**

This bit is set to 1 if no response is returned within 64 SDCLK cycles from the end bit of the command. If this bit is set to 1, the other error status bits (SDMMC_ACESR[4:2]) are meaningless.

**Table 48-7. Relations between ACMDCRC and ACMDTEO**

| ACMDCRC | ACMDTEO | Types of error |
|---|---|---|
| 0 | 0 | No error |
| 0 | 1 | Response Timeout error |
| 1 | 0 | Response CRC error |
| 1 | 1 | CMD line conflict |

- **ACMDCRC: Auto CMD CRC Error**

This bit is set to 1 when detecting a CRC error in the command response (refer to Table 48-7 for more details).

- **ACMDEND: Auto CMD End Bit Error**

This bit is set to 1 when detecting that the end bit of the command response is 0.

0: No error

1: Error

- **ACMDIDX: Auto CMD Index Error**

This bit is set to 1 when the Command Index error occurs in response to a command.

0: No error

1: Error

- **CMDNI: Command Not Issued by Auto CMD12 Error**

Setting this bit to 1 means CMD_wo_DAT is not executed due to an Auto CMD12 error (SDMMC_ACESR[4:1]). This bit is set to 0 when Auto CMD Error is generated by Auto CMD23.

0: No error

1: Error

### 48.13.32 SDMMC Host Control 2 Register (SD_SDIO)

**Name:** SDMMC_HC2R (SD_SDIO)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| PVALEN | ASINTEN | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SLCKSEL | EXTUN | DRVSEL | | VS18EN | UHSMS | | |

- **UHSMS: UHS Mode Select**

This field is used to select one of the UHS-I modes and is effective when 1.8V Signal Enable (VS18EN) is set to 1.

If Preset Value Enable is set to 1, the SDMMC sets SDCLK Frequency Select (SDCLKFSEL), Clock Generator Select (CLKGSEL) in SDMMC_CCR and Driver Strength Select (DRVSEL) according to SDMMC_PVR. In this case, one of the preset value registers is selected by this field. The user needs to reset SD Clock Enable (SDCLKEN) before changing this field to avoid generating a clock glitch. After setting this field, the user sets SDCLKEN to 1 again.

| Value | Name | Description |
|---|---|---|
| 0 | SDR12 | UHS SDR12 Mode |
| 1 | SDR25 | UHS SDR25 Mode |
| 2 | SDR50 | UHS SDR50 Mode |
| 3 | SDR104 | UHS SDR104 Mode |
| 4 | DDR50 | UHS DDR50 Mode |

Note: This field is effective only if SDMMC_MC1R.DDR is set to 0.

- **VS18EN: 1.8V Signaling Enable**

This bit controls the voltage regulator for the I/O cell. 3.3V is supplied to the card regardless of the signaling voltage.

Setting this bit from 0 to 1 starts changing the signal voltage from 3.3V to 1.8V. The 1.8V regulator output must be stable within 5 ms.

Clearing this bit from 1 to 0 starts changing the signal voltage from 1.8V to 3.3V. The 3.3V regulator output must be stable within 5 ms.

The user can set this bit to 1 when the SDMMC supports 1.8V signaling (one of the support bits is set to 1: SDR50SUP, SDR104SUP or DDR50SUP in SDMMC_CA1R) and the card or device supports UHS-I (S18A = 1. Refer to “Bus Switch Voltage Switch Sequence in the “Physical Layer Simplified Specification V3.01”).

0: 3.3V signaling

1: 1.8V signaling

- **DRVSEL: Driver Strength Select**

The SDMMC output driver in 1.8V signaling is selected by this bit. In 3.3V signaling, this field is not effective. This field can be set according to the Driver Type A, C and D support bits in SDMMC_CA1R.

This field depends on the setting of Preset Value Enable (PVALEN):

– PVALEN = 0: This field is set by the user.

– PVALEN = 1: This field is automatically set by a value specified in one of the SDMMC_PVRx.

| Value | Name | Description |
|---|---|---|
| 0 | TYPEB | Driver Type B is selected (Default) |
| 1 | TYPEA | Driver Type A is selected |
| 2 | TYPEC | Driver Type C is selected |
| 3 | TYPED | Driver Type D is selected |

- **EXTUN: Execute Tuning**

This bit is set to 1 to start the tuning procedure and is automatically cleared when the tuning procedure is completed. The result of tuning is indicated to Sampling Clock Select (SCLKSEL). The tuning procedure is aborted by writing 0. Refer to Figure 2.29 in the "SD Host Controller Simplified Specification V3.00".

0: Not tuned or tuning completed

1: Execute tuning

- **SLCKSEL: Sampling Clock Select**

The SDMMC uses this bit to select the sampling clock to receive CMD and DAT.

This bit is set by the tuning procedure and is valid after completion of tuning (when EXTUN is cleared). Setting 1 means that tuning is completed successfully and setting 0 means that tuning has failed.

Writing 1 to this bit is meaningless and ignored. A tuning circuit is reset by writing to 0. This bit can be cleared by setting EXTUN to 1. Once the tuning circuit is reset, it takes time to complete the tuning sequence. Therefore, the user should keep this bit to 1 to perform a retuning sequence to complete a retuning sequence in a short time. Changing this bit is not allowed while the SDMMC is receiving a response or a read data block. Refer to Figure 2.29 in the "SD Host Controller Simplified Specification V3.00".

0: The fixed clock is used to sample data.

1: The tuned clock is used to sample data.

- **ASINTEN: Asynchronous Interrupt Enable**

This bit can be set to 1 if a card support asynchronous interrupts and Asynchronous Interrupt Support (ASINTSUP) is set to 1 in SDMMC_CA0R. Asynchronous interrupt is effective when DAT[1] interrupt is used in 4-bit SD mode. If this bit is set to 1, the user can stop the SDCLK during the asynchronous interrupt period to save power. During this period, the SDMMC continues to deliver the Card Interrupt to the host when it is asserted by the card.

0: Disabled

1: Enabled

- **PVALEN: Preset Value Enable**

As the operating SDCLK frequency and I/O driver strength depend on the system implementation, it is difficult to determine these parameters in the standard host driver. When Preset Value Enable (PVALEN) is set to 1, automatic SDCLK frequency generation and driver strength selection are performed without considering system-specific conditions. This bit enables the functions defined in SDMMC_PVR.

if this bit is set to 0, SDCLKFSEL, CLKGSEL in SDMMC_CCR and DRVSEL in SDMMC_HC2R are set by the user.

if this bit is set to 1, SDCLKFSEL, CLKGSEL in SDMMC_CCR and DRVSEL in SDMMC_HC2R are set by the SDMMC as specified in SDMMC_PVR.

0: SDCLK and Driver strength are controlled by the user.

1: Automatic selection by Preset Value is enabled.

### 48.13.33 SDMMC Host Control 2 Register (eMMC)

**Name:** SDMMC_HC2R (eMMC)

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| PVALEN | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SLCKSEL | EXTUN | DRVSEL | | HS200EN | | | |

- **HS200EN: HS200 Mode Enable**

This field is used to select the e.MMC HS200 mode. When HS200EN is set to $B_{(hexa)}$, the HS200 mode is enabled. Any other value except 0 is forbidden when interfacing an e.MMC device.

If Preset Value Enable is set to 1, SDMMC sets SDCLK Frequency Select (SDCLKFSEL), Clock Generator Select (CLKGSEL) in SDMMC_CCR and Driver Strength Select (DRVSEL) according to SDMMC_PVR. In this case, one of the preset value registers is selected by this field. The user needs to reset SD Clock Enable (SDCLKEN) before changing this field to avoid generating a clock glitch. After setting this field, the user sets SDCLKEN to 1 again.

Note: This field is effective only if DDR in SDMMC_MC1R is set to 0.

- **DRVSEL: Driver Strength Select**

The SDMMC output driver in 1.8V signaling is selected by this bit. In 3.3V signaling, this field is not effective. This field can be set according to the Driver Type A, C and D support bits in SDMMC_CA1R.

This field depends on setting of Preset Value Enable (PVALEN):

– PVALEN = 0: This field is set by the user.

– PVALEN = 1: This field is automatically set by a value specified in one of the SDMMC_PVRx.

| Value | Name | Description |
|---|---|---|
| 0 | TYPEB | Driver Type B is selected (Default) |
| 1 | TYPEA | Driver Type A is selected |
| 2 | TYPEC | Driver Type C is selected |
| 3 | TYPED | Driver Type D is selected |

- **EXTUN: Execute Tuning**

This bit is set to 1 to start the tuning procedure and is automatically cleared when the tuning procedure is completed. The result of tuning is indicated to Sampling Clock Select (SCLKSEL). The tuning procedure is aborted by writing 0. Refer to Figure 2.29 in the "SD Host Controller Simplified Specification V3.00".

0: Not tuned or tuning completed

1: Execute tuning

- **SLCKSEL: Sampling Clock Select**

The SDMMC uses this bit to select the sampling clock to receive CMD and DAT.

This bit is set by the tuning procedure and is valid after completion of tuning (when EXTUN is cleared). Setting 1 means that tuning is completed successfully and setting 0 means that tuning has failed.

Writing 1 to this bit is meaningless and ignored. A tuning circuit is reset by writing to 0. This bit can be cleared by setting EXTUN to 1. Once the tuning circuit is reset, it takes time to complete a tuning sequence. Therefore, the user should keep this bit to 1 to perform a retuning sequence to complete a retuning sequence in a short time. Changing this bit is not

allowed while the SDMMC is receiving a response or a read data block. Refer to Figure 2.29 in the “SD Host Controller Simplified Specification V3.00”.

0: The fixed clock is used to sample data.

1: The tuned clock is used to sample data.

- **PVALEN: Preset Value Enable**

As the operating SDCLK frequency and I/O driver strength depend on the system implementation, it is difficult to determine these parameters in the standard host driver. When Preset Value Enable (PVALEN) is set to 1, automatic SDCLK frequency generation and driver strength selection are performed without considering system-specific conditions. This bit enables the functions defined in SDMMC_PVR.

If this bit is set to 0, SDCLKFSEL, CLKGSEL in SDMMC_CCR and DRVSEL in SDMMC_HC2R are set by the user.

If this bit is set to 1, SDCLKFSEL, CLKGSEL in SDMMC_CCR and DRVSEL in SDMMC_HC2R are set by the SDMMC as specified in SDMMC_PVR.

0: SDCLK and Driver strength are controlled by the user.

1: Automatic selection by Preset Value is enabled.

### 48.13.34 SDMMC Capabilities 0 Register

**Name:** SDMMC_CA0R

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SLTYPE | | ASINTSUP | SB64SUP | – | V18VSUP | V30VSUP | V33VSUP |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| SRSUP | SDMASUP | HSSUP | – | ADMA2SUP | ED8SUP | MAXBLKL | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| BASECLKF | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TEOCLKU | – | TEOCLKF | | | | | |

Note: The Capabilities 0 Register is not supposed to be written by the user. However, the user can modify preset values only if Capabilities Write Enable (CAPWREN) is set to 1 in SDMMC_CACR.

- **TEOCLKF: Timeout Clock Frequency**

This bit shows the timeout clock frequency (TEOCLK) used to detect Data Timeout Error.

If this field is set to 0, the user must get the information via another method.

The Timeout Clock Unit (TEOCLKU) defines the unit of this field's value.

– TEOCLKU = 0: $f_{TEOCLK} = TEOCLKF_{KHz}$

– TEOCLKU = 1: $f_{TEOCLK} = TEOCLKF_{MHz}$

- **TEOCLKU: Timeout Clock Unit**

This bit shows the unit of the base clock frequency used to detect Data Timeout Error.

0: KHz

1: MHz

- **BASECLKF: Base Clock Frequency**

This value indicates the frequency of the base clock (BASECLK). The user uses this value to calculate the clock divider value (refer to SDCLK Frequency Select (SDCLKFSEL) in SDMMC_CCR).

If this field is set to 0, the user must get the information via another method.

$f_{BASECLK} = BASECLKF_{MHz}$

- **MAXBLKL: Max Block Length**

This field indicates the maximum block size that the user can read and write to the buffer in the SDMMC. Three sizes can be defined, as shown below. It is noted that the transfer block length is always 512 bytes for SD Memory Cards regardless of this field.

| Value | Name | Description |
|---|---|---|
| 0 | 512 | 512 bytes |
| 1 | 1024 | 1024 bytes |
| 2 | 2048 | 2048 bytes |
| 3 | NONE | Reserved |

- **ED8SUP: 8-Bit Support for Embedded Device**

This bit indicates whether the SDMMC is capable of using the 8-bit Bus Width mode.

0: 8-bit bus width not supported

1: 8-bit bus width supported

- **ADMA2SUP: ADMA2 Support**

This bit indicates whether the SDMMC is capable of using ADMA2.

0: ADMA2 not supported

1: ADMA2 supported

- **HSSUP: High Speed Support**

This bit indicates whether the SDMMC and the system support High Speed mode and they can supply SDCLK frequency from 25 MHz to 50 MHz.

0: High Speed not supported

1: High Speed supported

- **SDMASUP: SDMA Support**

This bit indicates whether the SDMMC is capable of using SDMA to transfer data between system memory and the SDMMC directly.

0: SDMA not supported

1: SDMA supported

- **SRSUP: Suspend/Resume Support**

This bit indicates whether the SDMMC supports the Suspend/Resume functionality. If this bit is set to 0, the user does not issue either Suspend or Resume commands because the Suspend and Resume mechanism (refer to “Suspend and Resume Mechanism” in the “SD Host Controller Simplified Specification V3.00”) is not supported.

0: Suspend/Resume not supported

1: Suspend/Resume supported

- **V33VSUP: Voltage Support 3.3V**

0: 3.3V Voltage supply not supported

1: 3.3V Voltage supply supported

- **V30VSUP: Voltage Support 3.0V**

0: 3.0V Voltage supply not supported

1: 3.0V Voltage supply supported

- **V18VSUP: Voltage Support 1.8V**

0: 1.8V Voltage supply not supported

1: 1.8V Voltage supply supported

- **SB64SUP: 64-Bit System Bus Support**

Reading this bit to 1 means that the SDMMC supports the 64-bit Address Descriptor mode and is connected to the 64-bit address system bus.

0: 64-bit address bus not supported

1: 64-bit address bus supported

- **ASINTSUP: Asynchronous Interrupt Support**

Refer to section “Asynchronous Interrupt” in the “SDIO Simplified Specification V3.00”.

0: Asynchronous interrupt not supported

1: Asynchronous interrupt supported

- **SLTYPE: Slot Type**

This field indicates usage of a slot by a specific system. An SDMMC control register set is defined per slot.

Embedded Slot for One Device means that only one non-removable device is connected to a bus slot.

The Standard Host Driver controls  a removable card (SLTYPE = 0) or one embedded device (SLTYPE = 1) connected to an SD bus slot.

| Value | Description |
|---|---|
| 0 | Removable Card Slot |
| 1 | Embedded Slot for One Device |
| 2 | Reserved |
| 3 | Reserved |

### 48.13.35 SDMMC Capabilities 1 Register

**Name:** SDMMC_CA1R

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| CLKMULT | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| RTMOD | | TSDR50 | – | TCNTRT | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | DRVDSUP | DRVCSUP | DRVASUP | – | DDR50SUP | SDR104SUP | SDR50SUP |

Note: The Capabilities 1 Register is not supposed to be written by the user. However, the user can modify preset values only if Capabilities Write Enable (CAPWREN) is set to 1 in SDMMC_CACR.

- **SDR50SUP: SDR50 Support**

0: SDR50 mode is not supported.

1: SDR50 mode is supported.

- **SDR104SUP: SDR104 Support**

0: SDR104 mode is not supported.

1: SDR104 mode is supported.

- **DDR50SUP: DDR50 Support**

0: DDR50 mode is not supported.

1: DDR50 mode is supported.

- **DRVASUP: Driver Type A Support**

0: Driver type A is not supported.

1: Driver type A is supported.

- **DRVCSUP: Driver Type C Support**

0: Driver type C is not supported.

1: Driver type C is supported.

- **DRVDSUP: Driver Type D Support**

0: Driver type D is not supported.

1: Driver type D is supported.

- **TCNTRT: Timer Count For Retuning**

This field indicates an initial value of the Retuning Timer for Retuning Mode (RTMOD) 1 to 3. Reading this field at 0 means that the Retuning Timer is disabled. The Retuning Timer initial value ranges from 0 to 1024 seconds.

$t_{TIMER} = 2^{(TCNTRT - 1)}$Seconds

- **TSDR50: Use Tuning for SDR50**

If this bit is set to 1, the SDMMC requires tuning to operate SDR50 (tuning is always required to operate SDR104).

0: SDR50 does not require tuning.

1: SDR50 requires tuning.

- **RTMOD: Retuning Modes**

This field selects the retuning method and limits the maximum data length.

There are two retuning timings: Retuning Request (RTREQ) controlled by the SDMMC, and expiration of a Retuning timer controlled by the user. By receiving either timing, the user executes the retuning procedure just before a next command issue.

The maximum data length per read/write command is restricted so that retuning procedures can be inserted during data transfers.

Retuning Mode 1:

The SDMMC does not have any internal logic to detect when retuning needs to be performed. In this case, the user should maintain all retuning timings by using the Retuning Timer. To enable inserting the retuning procedure during data transfers, the data length per Read/Write command must be limited to 4 Mbytes.

Retuning Mode 2:

The SDMMC has the capability to indicate the retuning timing by Retuning Request (RTREQ) during data transfers. Then the data length per Read/Write command must be limited to 4 Mbytes.

During non-data transfer, retuning timing is determined by either Retuning Request or Retuning Timer. If Retuning Request is used, Retuning Timer should be disabled.

Retuning Mode 3:

The SDMMC has the capability to take care of the retuning during data transfer (Auto Retuning). Retuning Request is not generated during data transfers and there is no limitation to data length per Read/Write command.

During non-data transfer, retuning timing is determined either by Retuning Request or Retuning Timer. If Retuning Request is used, Retuning Timer should be disabled.

| Value | Name | Description | Data Length |
|---|---|---|---|
| 0 | MODE1 | Timer | 4 Mbytes (Max) |
| 1 | MODE2 | Timer and Retuning Request | 4 Mbytes (Max) |
| 2 | MODE3 | Auto Retuning (for transfer) Timer and Retuning Request | Any |
| 3 | – | Reserved | – |

- **CLKMULT: Clock Multiplier**

This field indicates the multiplier factor between the Base Clock (BASECLK) used for the Divided Clock Mode and the Multiplied Clock (MULTCLK) used for the Programmable Clock mode (refer to SDMMC_CCR).

Reading this field to 0 means that the Programmable Clock mode is not supported.

$$f_{MULTCLK} = f_{BASECLK} \times (CLKMULT + 1)$$

### 48.13.36 SDMMC Maximum Current Capabilities Register

**Name:** SDMMC_MCCAR

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| MAXCUR18V | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| MAXCUR30V | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MAXCUR33V | | | | | | | |

- **MAXCUR33V: Maximum Current for 3.3V**

This field indicates the maximum current capability for 3.3V voltage. This value is meaningful only if V33VSUP is set to 1 in SDMMC_CA0R. Reading MAXCUR33V at 0 means that the user must get information via another method.

$$Imax_{mA} = 4 \times MAXCURR33V$$

- **MAXCUR30V: Maximum Current for 3.0V**

This field indicates the maximum current capability for 3.0V voltage. This value is meaningful only if V30VSUP is set to 1 in SDMMC_CA0R. Reading MAXCUR30V at 0 means that the user must get information via another method.

$$Imax_{mA} = 4 \times MAXCURR30V$$

- **MAXCUR18V: Maximum Current for 1.8V**

This field indicates the maximum current capability for 1.8V voltage. This value is meaningful only if V18VSUP is set to 1 in SDMMC_CA0R. Reading MAXCUR18V at 0 means that the user must get information via another method.

$$Imax_{mA} = 4 \times MAXCURR18V$$

### 48.13.37 SDMMC Force Event Register for Auto CMD Error Status

**Name:** SDMMC_FERACES

**Access:** Write-only

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CMDNI | – | – | ACMDIDX | ACMDEND | ACMDCRC | ACMDTEO | ACMD12NE |

- **ACMD12NE: Force Event for Auto CMD12 Not Executed**

For testing purposes, the user can write this bit to 1 to rise the ACMD12NE status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

- **ACMDTEO: Force Event for Auto CMD Timeout Error**

For testing purposes, the user can write this bit to 1 to rise the ACMDTEO status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

- **ACMDCRC: Force Event for Auto CMD CRC Error**

For testing purposes, the user can write this bit to 1 to rise the ACMDCRC status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

- **ACMDEND: Force Event for Auto CMD End Bit Error**

For testing purposes, the user can write this bit to 1 to rise the ACMDEND status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

- **ACMDIDX: Force Event for Auto CMD Index Error**

For testing purposes, the user can write this bit to 1 to rise the ACMDIDX status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

- **CMDNI: Force Event for Command Not Issued by Auto CMD12 Error**

For testing purposes, the user can write this bit to 1 to rise the CMDNI status flag in SDMMC_ACESR.

Writing this bit to 0 has no effect.

### 48.13.38 SDMMC Force Event Register for Error Interrupt Status

**Name:** SDMMC_FEREIS

**Access:** Write-only

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOOTAE | – | – | ADMA | ACMD |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURLIM | DATEND | DATCRC | DATTEO | CMDIDX | CMDEND | CMDCRC | CMDTEO |

- **CMDTEO: Force Event for Command Timeout Error**

For testing purposes, the user can write this bit to 1 to rise the CMDTEO status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **CMDCRC: Force Event for Command CRC Error**

For testing purposes, the user can write this bit to 1 to rise the CMDCRC status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **CMDEND: Force Event for Command End Bit Error**

For testing purposes, the user can write this bit to 1 to rise the CDMEND status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **CMDIDX: Force Event for Command Index Error**

For testing purposes, the user can write this bit to 1 to rise the CMDIDX status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **DATTEO: Force Event for Data Timeout error**

For testing purposes, the user can write this bit to 1 to rise the DATTEO status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **DATCRC: Force Event for Data CRC error**

For testing purposes, the user can write this bit to 1 to rise the DATCRC status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **DATEND: Force Event for Data End Bit Error**

For testing purposes, the user can write this bit to 1 to rise the DATEND status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **CURLIM: Force Event for Current Limit Error**

For testing purposes, the user can write this bit to 1 to rise the CURLIM status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **ACMD: Force Event for Auto CMD Error**

For testing purposes, the user can write this bit to 1 to rise the ACMD status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **ADMA: Force Event for ADMA Error**

For testing purposes, the user can write this bit to 1 to rise the ADMA status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

- **BOOTAE: Force Event for Boot Acknowledge Error**

For testing purposes, the user can write this bit to 1 to rise the BOOTAE status flag in SDMMC_EISTR.

Writing this bit to 0 has no effect.

### 48.13.39 SDMMC ADMA Error Status Register

**Name:** SDMMC_AESR

**Access:** Read-only

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | LMIS | ERRST | |

- **ERRST: ADMA Error State**

This field indicates the state of ADMA when an error has occurred during an ADMA data transfer. This field never indicates 2 because ADMA never stops in this state.

| Value | ADMA Error State when Error Occurred | Content of SDMMC_ASARx Registers |
|---|---|---|
| 0 | ST_STOP (Stop DMA) | Points to the descriptor following the error descriptor |
| 1 | ST_FDS (Fetch Descriptor) | Points to the error descriptor |
| 2 | – | (Not used) |
| 3 | ST_TRF (Transfer Data) | Points to the descriptor following the error descriptor |

- **LMIS: ADMA Length Mismatch Error**

This error occurs in the following two cases:

– While Block Count Enable (BCEN) is being set, the total data length specified by the Descriptor table is different from that specified by the Block Count (BLKCNT) and Transfer Block Size (BLKSIZE).

– The total data length cannot be divided by the Transfer Block Size (BLKSIZE).

0: No error

1: Error

### 48.13.40 SDMMC ADMA System Address Register

**Name:** SDMMC_ASAR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ADMASA | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| ADMASA | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ADMASA | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADMASA | | | | | | | |

- **ADMASA: ADMA System Address**

This field holds the byte address of the executing command of the descriptor table. The 32-bit address descriptor uses SDMMC_ASAR. At the start of ADMA, the user must set the start address of the descriptor table. The ADMA increments this register address, which points to the next Descriptor line to be fetched.

When the ADMA Error (ADMA) status flag rises, this field holds a valid descriptor address depending on the ADMA Error State (ERRST). The user must program Descriptor Table on 32-bit boundary and set 32-bit boundary address to this register. ADMA2 ignores the lower 2 bits of this register and assumes it to be 0.

### 48.13.41 SDMMC Preset Value Register

**Name:** SDMMC_PVRx [x=0..7]

**Access:** Read/Write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| DRVSEL | | – | – | – | CLKGSEL | SDCLKFSEL | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SDCLKFSEL | | | | | | | |

One of the Preset Value Registers is effective based on the selected bus speed mode. Table 48-8 defines the conditions to select one of the SDMMC_PVRs.

**Table 48-8. Preset Value Register Select Condition**

| Selected Bus Speed Mode | VS18EN (SDMMC_HC2R) | HSEN (SDMMC_HC1R) | UHSMS (SDMMC_HC2R) |
|---|---|---|---|
| Default Speed | 0 | 0 | don't care |
| High Speed | 0 | Response Timeout Error | don't care |
| SDR12 | 1 | don't care | 0 |
| SDR25 | 1 | don't care | 1 |
| SDR50 | 1 | don't care | 2 |
| SDR104/HS200 | 1 | don't care | 3 |
| DDR50 | 1 | don't care | 4 |
| Reserved | 1 | don't care | Other values |

Table 48-9 shows the effective Preset Value Register according to the Selected Bus Speed mode.

**Table 48-9. Preset Value Registers**

| SDMMC_PVRx | Selected Bus Speed Mode | Signal Voltage |
|---|---|---|
| SDMMC_PVR0 | Initialization | 3.3V or 1.8V |
| SDMMC_PVR1 | Default Speed | 3.3V |
| SDMMC_PVR2 | High Speed | 3.3V |
| SDMMC_PVR3 | SDR12 | 1.8V |
| SDMMC_PVR4 | SDR25 | 1.8V |
| SDMMC_PVR5 | SDR50 | 1.8V |
| SDMMC_PVR6 | SDR104/HS200 | 1.8V |
| SDMMC_PVR7 | DDR50 | 1.8V |

When Preset Value Enable (PVALEN) in SDMMC_HC2R is set to 1, SDCLK Frequency Select (SDLCKFSEL) and Clock Generator Select (CLKGSEL) in SDMMC_CCR, and Driver Strength Select (DRVSEL) in SDMMC_HC2R are automatically set based on the Selected Bus Speed mode. This means that the user does not need to set these fields when preset is enabled. A Preset Value Register for Initialization (SDMMC_PVR0) is not selected by Bus Speed mode. Before starting the initialization sequence, the user needs to set a clock preset value to SDCLKFSEL in SDMMC_CCR. PVALEN can be set to 1 after the initialization is completed.

Note: Preset Values in SDMMC_PVRx registers are not supposed to be written by the user. However, the user can modify preset values only if Capabilities Write Enable (CAPWREN) is set to 1 in SDMMC_CACR.

- **SDCLKFSEL: SDCLK Frequency Select**

Refer to SDCLKFSEL in SDMMC_CCR.

- **CLKGSEL: Clock Generator Select**

Refer to CLKGSEL in SDMMC_CCR.

- **DRVSEL: Driver Strength Select**

Refer to DRVSEL in SDMMC_HC2R.

### 48.13.42 SDMMC Slot Interrupt Status Register

**Name:** SDMMC_SISR

**Access:** Read-only

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | INTSSL | |

- **INTSSL: Interrupt Signal for Each Slot**

These status bits indicate the logical OR of Interrupt Signals and Wakeup Signal for each SDMMC instance in the product (INTSSL[x] corresponds to SDMMCx instance in the product).

### 48.13.43 SDMMC Host Controller Version Register

Name: SDMMC_HCVR

Access: Read-only

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| VVER | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SVER | | | | | | | |

- **SVER: Specification Version Number**

This status indicates the SD Host Controller Specification Version.

| Value | Description |
|---|---|
| 0 | SD Host Specification Version 1.00 |
| 1 | SD Host Specification Version 2.00, including the feature of the ADMA and Test Register |
| 2 | SD Host Specification Version 3.00 |

- **VVER: Vendor Version Number**

Reserved. Value subject to change. No functionality associated. This is the Atmel internal version of the macrocell.

### 48.13.44 SDMMC Additional Present State Register

**Name:** SDMMC_APSR

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | HDATLL | | | |

- **HDATLL: DAT[7:4] High Line Level**

This status is used to check the DAT[7:4] line level to recover from errors, and for debugging.

### 48.13.45 SDMMC e.MMC Control 1 Register

**Name:** SDMMC_MC1R

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCD | RSTN | BOOTA | OPD | DDR | – | CMDTYP | |

- **CMDTYP: e.MMC Command Type**

| Value | Name | Description |
|---|---|---|
| 0 | NORMAL | The command is not an e.MMC specific command. |
| 1 | WAITIRQ | This bit must be set to 1 when the e.MMC is in Interrupt mode (CMD40). Refer to “Interrupt Mode” in the “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”. |
| 2 | STREAM | This bit must be set to 1 in the case of Stream Read(CMD11) or Stream Write (CMD20). Only effective for e.MMC up to revision 4.41. |
| 3 | BOOT | Starts a Boot Operation mode at the next write to SDMMC_CR. Boot data are read directly from e.MMC device. |

- **DDR: e.MMC HSDDR Mode**

This bit selects the High Speed DDR mode.

0: High Speed DDR is not selected.

1: High Speed DDR is selected.

Note: The clock divider (DIV) in SDMMC_CCR must be set to a value different from 0 when HSEN is 1.

- **OPD: e.MMC Open Drain Mode**

This bit sets the command line in open drain.

0: The command line is in push-pull.

1: The command line is in open drain.

- **BOOTA: e.MMC Boot Acknowledge Enable**

This bit must be set according to the value of BOOT_ACK in the Extended CSD Register (refer to “Embedded MultiMedia Card (e.MMC) Electrical Standard 4.51”).

When this bit is set to 1, the SDMMC waits for boot acknowledge pattern from the e.MMC before receiving boot data.

If the boot acknowledge pattern is wrong, the BOOTAE status flag rises in SDMMC_EISTR if BOOTAE is set in SDMMC_EISTER. An interrupt is generated if BOOTAE is set in SDMMC_EISIER.

If the no boot acknowledge pattern is received, the DATTEO status flag rises in SDMMC_EISTR if DATTEO is set in SDMMC_EISTER. An interrupt is generated if DATTEO is set in SDMMC_EISIER.

- **RSTN: e.MMC Reset Signal**

This bit controls the e.MMC reset signal.

0: Reset signal is inactive.

1: Reset signal is active.

- **FCD: e.MMC Force Card Detect**

When using e.MMC, the user can set this bit to 1 to bypass the card detection procedure using the SDMMC_CD signal.

0 (DISABLED): e.MMC Forced Card Detect is disabled. The SDMMC_CD signal is used and debounce timing is applied.

1 (ENABLED): e.MMC Forced Card Detect is enabled.

### 48.13.46 SDMMC e.MMC Control 2 Register

**Name:** SDMMC_MC2R

**Access:** Write-only

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | ABOOT | SRESP |

- **SRESP: e.MMC Abort Wait IRQ**

This bit is used to exit from the Interrupt mode. When this bit is written to 1, the SDMMC sends the CMD40 response automatically. This brings the e.MMC from Interrupt mode to the standard Data Transfer mode. Writing this bit to 0 is ignored.

Note: This bit is only effective when CMD_TYP in SDMMC_MC1R is set to WAITIRQ.

- **ABOOT: e.MMC Abort Boot**

This bit is used to exit from Boot mode. Writing this bit to 1 exits the Boot Operation mode. Writing 0 is ignored.

### 48.13.47 SDMMC AHB Control Register

**Name:** SDMMC_ACR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | BMAX | |

- **BMAX: AHB Maximum Burst**

This field selects the maximum burst size in case of DMA transfer.

| Value | Name | Description |
|---|---|---|
| 0 | INCR16 | The maximum burst size is INCR16. |
| 1 | INCR8 | The maximum burst size is INCR8. |
| 2 | INCR4 | The maximum burst size is INCR4. |
| 3 | SINGLE | Only SINGLE transfers are performed. |

### 48.13.48 SDMMC Clock Control 2 Register

**Name:** SDMMC_CC2R

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | FSDCLKD |

- **FSDCLKD: Force SDCLK Disabled**

The user can choose to maintain the SDCLK during 8 SDCLK cycles after the end bit of the last data block in case of a read transaction, or after the end bit of the CRC status in case of a write transaction.

0: The SDCLK is forced and it cannot be stopped immediately after the transaction.

1: The SDCLK is not forced and it can be stopped immediately after the transaction.

### 48.13.49 SDMMC Retuning Control 1 Register

**Name:** SDMMC_RTC1R

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | TMREN |

- **TMREN: Retuning Timer Enable**

Enable the retuning timer.

0 (DISABLED): The retuning timer is disabled.

1 (ENABLED): The retuning timer is enabled.

### 48.13.50 SDMMC Retuning Control 2 Register

**Name:** SDMMC_RTC2R

**Access:** Write-only

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | RLD |

- **RLD: Retuning Timer Reload**

This bit is only efficient if the Retuning timer is enabled (TMREN is set to 1 in SDMMC_RTC1R). Once the Timer Counter Value (TCVAL) is set to a non-zero value in SDMMC_RTCVR, setting this bit to 1 starts the timer count. The retuning timer count restarts each time this bit is written to 1.

Writing this bit to 0 has no effect.

### 48.13.51 SDMMC Retuning Counter Value Register

**Name:** SDMMC_RTCVR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | TCVAL | | | |

- **TCVAL: Retuning Timer Counter Value**

The TCVAL value corresponds to the time, in seconds, before expiration of the retuning timer. This value must range between 1 and 11. Any other value results in the retuning timer to be disabled.

### 48.13.52 SDMMC Retuning Interrupt Status Enable Register

**Name:** SDMMC_RTISTER

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | TEVT |

- **TEVT: Retuning Timer Event**

0 (MASKED): The TEVT status flag in SDMMC_RTISTR is masked.

1 (ENABLED): The TEVT status flag in SDMMC_RTISTR is enabled.

### 48.13.53 SDMMC Retuning Interrupt Signal Enable Register

**Name:** SDMMC_RTISIER

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | TEVT |

- **TEVT: Retuning Timer Event**

0 (MASKED): No interrupt is generated when the TEVT status rises in SDMMC_RTISTR.

1 (ENABLED): An interrupt is generated when the TEVT status rises in SDMMC_RTISTR.

### 48.13.54 SDMMC Retuning Interrupt Status Register

**Name:** SDMMC_RTISTR

**Access:** Read/Write

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | TEVT |

- **TEVT: Retuning Timer Event**

This bit is set to 1 when the retuning timer count is elapsed if TEVT is set to 1 in SDMMC_RTISTER. An interrupt is generated if TEVT is set to 1 in SDMMC_RTISIER.

Writing this bit to 1 clears this bit.

0: No retuning timer event.

1: Retuning timer event.

### 48.13.55 SDMMC Retuning Status Slots Register

**Name:** SDMMC_RTSSR

**Access:** Read-only

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | TEVTSLOT | |

- **TEVTSLOT: Retuning Timer Event Slots**

These status bits indicate the TEVT status for each SDMMC instance in the product (TEVTSLOT[x] corresponds to SDMMCx instance in the product).

### 48.13.56 SDMMC Tuning Control Register

**Name:** SDMMC_TUNCR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | SMPLPT |

- **SMPLPT: Sampling Point**

This bit selects the position of the sampling point into the data window for SDR104 and HS200 modes.

0: Sampling point is set at 50% of the data window.

1: Sampling point is set at 75% of the data window.

### 48.13.57 SDMMC Capabilities Control Register

**Name:** SDMMC_CACR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| KEY | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | CAPWREN |

- **CAPWREN: Capabilities Write Enable**

This bit can only be written if KEY correspond to 46h.

0: Capabilities registers (SDMMC_CA0R and SDMMC_CA1R) cannot be written.

1: Capabilities registers (SDMMC_CA0R and SDMMC_CA1R) can be written.

- **KEY: Key**

| Value | Name | Description |
|---|---|---|
| 46h | KEY | Writing any other value in this field aborts the write operation of the CAPWREN bit.<br>Always reads as 0. |

### 48.13.58 SDMMC Calibration Control Register

**Name:** SDMMC_CALCR

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CALP | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | CALN | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CNTVAL | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | TUNDIS | ALWYSON | – | – | – | EN |

- **EN: PADs Calibration Enable**

This bit is automatically cleared once the calibration is done.

0: SDMMC I/O calibration disabled.

1: SDMMC I/O calibration enabled.

- **ALWYSON: Calibration Analog Always ON**

0: Calibration analog is shut down after each calibration.

1: Calibration analog remains powered after calibration.

- **TUNDIS: Calibration During Tuning Disabled**

0: Calibration is launched before each tuning.

1: Calibration is not launched at tuning.

- **CNTVAL: Calibration Counter Value**

Defines the number of HCLOCK cycles (divided by 4) required to cover the I/O calibration cell startup time.

$$CNTVAL_{Minimum} = \frac{t_{STARTUP}}{4 \times t_{HCLOCK}}$$

$t_{STARTUP}$ = 2 µs

- **CALN: Calibration N Status**

Calibration code for the n-channel transistors to match the required output impedance.

- **CALP: Calibration P Status**

Calibration code for the p-channel transistors to match the required output impedance.

# 49. Image Sensor Controller (ISC)

## 49.1 Description

The Image Sensor Controller (ISC) system manages incoming data from a parallel sensor. It supports a single active interface. The parallel interface protocol can use a free-running clock or a gated clock strategy. It supports the ITU-R BT 656/1120 422 protocol with a data width of 8 bits or 10 bits and raw Bayer format. The internal image processor includes adjustable white balance, color filter array interpolation, color correction, gamma correction, 12 bits to 10 bits compression, programmable color space conversion, horizontal and vertical chrominance subsampling module. The module also integrates a triple channel direct memory access controller master interface.

## 49.2 Embedded Characteristics

- Parallel 12-bit Interface for Raw Bayer, YCbCr, Monochrome and JPEG Compressed Sensor Interface
- BT.601/656/1120 Video Interface Supported
- Progressive Systems and Segmented Frame Systems
- Raw Bayer, YCbCr, Luminance (Black and White) Pixel Format Supported
- Resolution up to 2592 x 1944
- Input Pixel Clock up to 96 MHz
- Output Master Clock Generation
- Cropping
- Adjustable White Balance
- Raw Bayer Color Filter Array Interpolation
- Color Correction
- Gamma Correction
- Color Space Conversion
- Contrast and Brightness Control
- 4:4:4 to 4:2:2 Subsampler
- 4:2:2 to 4:2:0 Subsampler
- Rounding, Limiting and Packing unit
- Histogram Generation
- System Interface: Direct Memory Access Interface with Packed, Semi Planar and Planar output format
- Output Memory Format: 16 bpp RGB, 32 bpp RGB, 16 bpp, YCbCr 444, YCbCr 422, YCbCr 420, up to 12-bit raw Bayer

## 49.3 Block Diagram and Use Cases

### 49.3.1 Image Sensor Controller Functional Diagrams

**Figure 49-1. Image Sensor Controller Block Diagram**

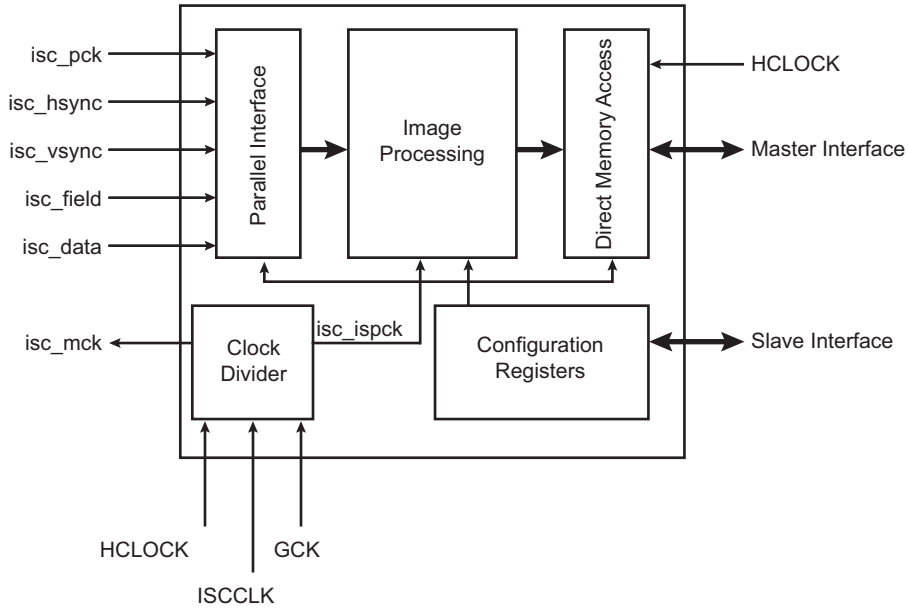
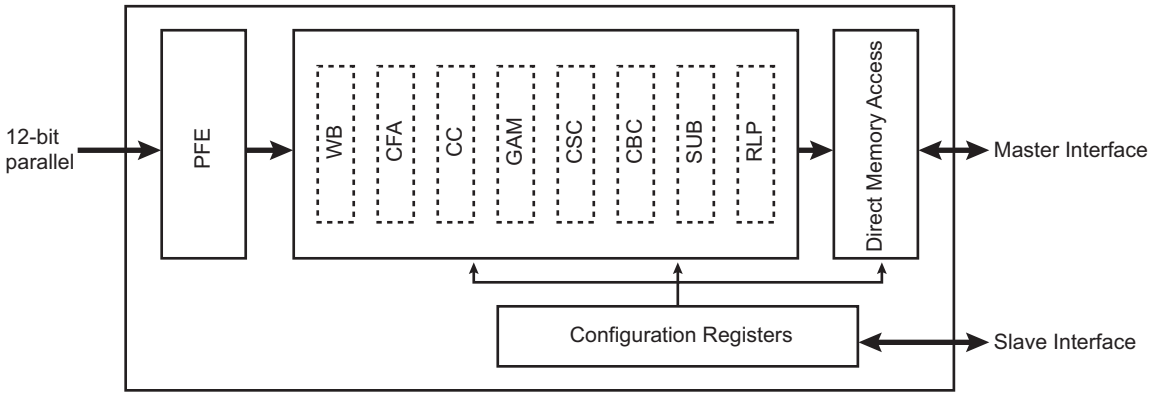



**Figure 49-2. Image Sensor Controller Raw Bayer Signal Processor**

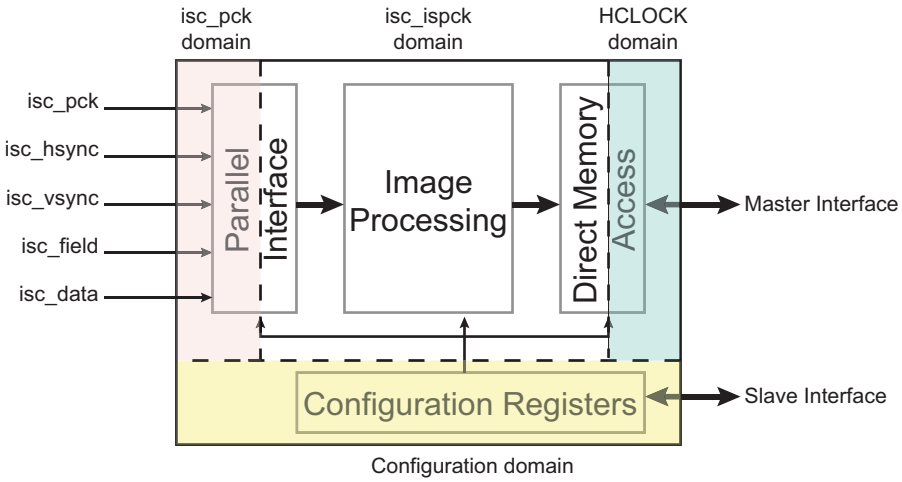



The ISC video pipeline integrates the following submodules:

- PFE: Parallel Front End to sample the camera sensor input stream
- WB: Programmable white balance in the Bayer domain
- CFA: Color filter array interpolation module
- CC: Programmable color correction
- GAM: Gamma correction
- CSC: Programmable color space conversion
- CBC: Contrast and Brightness control
- SUB: This module performs YCbCr444 to YCbCr420 chrominance subsampling.
- RLP: This module performs rounding, range limiting and packing of the incoming data.

### 49.3.2 Image Sensor Controller Clock Domain Diagram

**Figure 49-3. Clock Domain Hierarchy**




### 49.3.3 Image Sensor Controller Typical Use Cases

**Figure 49-4. Raw Bayer Sensor**

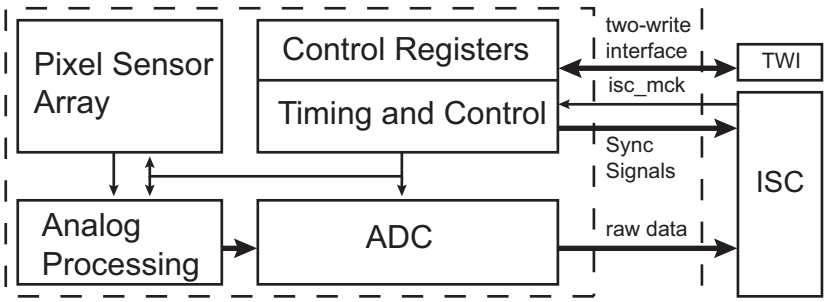



**Figure 49-5. Raw Bayer Sensor with Embedded Image Processor**

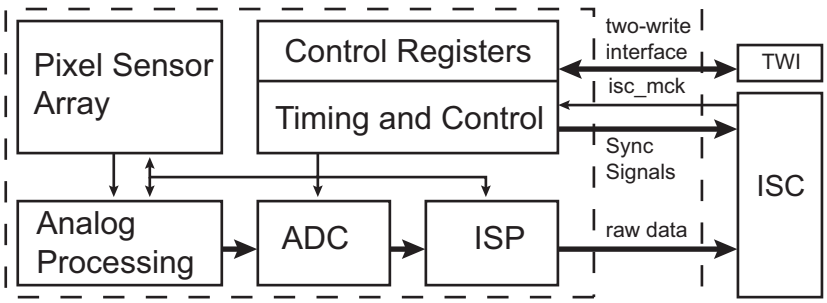


**Figure 49-6. BT656 Video Interface Sensor**

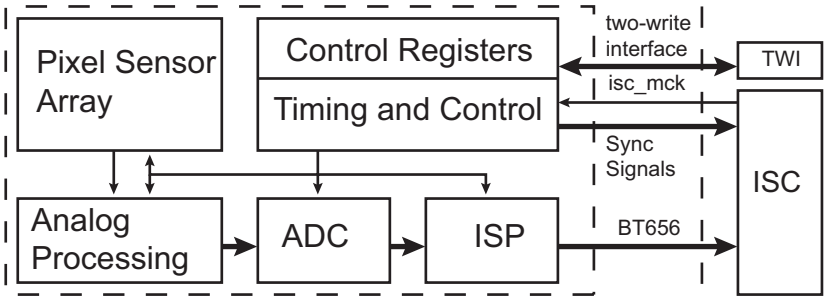



**Figure 49-7. Sensor with JPEG Output**

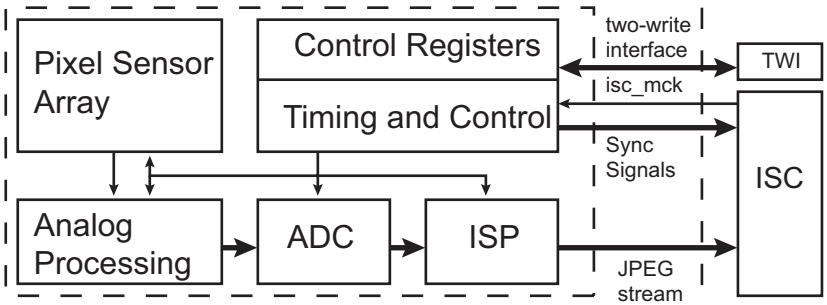
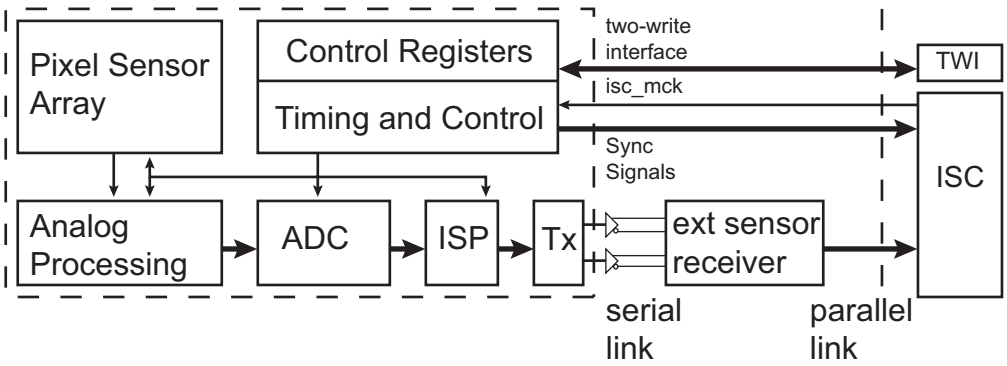



**Figure 49-8. Serial CMOS Sensor with External Parallel Bridge**




## 49.4 Product Dependencies

### 49.4.1 I/O Lines

The pins used for interfacing the ISC are multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the ISC pins to their peripheral function. If I/O lines of the ISC are not used by the application, they can be used for other purposes by the PIO controller.

**Table 49-1. I/O Lines**

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| ISC | ISC_D0 | PB26 | F |
| ISC | ISC_D0 | PC9 | C |
| ISC | ISC_D0 | PD7 | E |
| ISC | ISC_D1 | PB27 | F |

Table 49-1. I/O Lines (Continued)

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| ISC | ISC_D1 | PC10 | C |
| ISC | ISC_D1 | PD8 | E |
| ISC | ISC_D2 | PB28 | F |
| ISC | ISC_D2 | PC11 | C |
| ISC | ISC_D2 | PD9 | E |
| ISC | ISC_D3 | PB29 | F |
| ISC | ISC_D3 | PC12 | C |
| ISC | ISC_D3 | PD10 | E |
| ISC | ISC_D4 | PB30 | F |
| ISC | ISC_D4 | PC13 | C |
| ISC | ISC_D4 | PD11 | E |
| ISC | ISC_D4 | PD12 | F |
| ISC | ISC_D5 | PB31 | F |
| ISC | ISC_D5 | PC14 | C |
| ISC | ISC_D5 | PD12 | E |
| ISC | ISC_D5 | PD13 | F |
| ISC | ISC_D6 | PC0 | F |
| ISC | ISC_D6 | PC15 | C |
| ISC | ISC_D6 | PD13 | E |
| ISC | ISC_D6 | PD14 | F |
| ISC | ISC_D7 | PC1 | F |
| ISC | ISC_D7 | PC16 | C |
| ISC | ISC_D7 | PD14 | E |
| ISC | ISC_D7 | PD15 | F |
| ISC | ISC_D8 | PC2 | F |
| ISC | ISC_D8 | PC17 | C |
| ISC | ISC_D8 | PD6 | E |
| ISC | ISC_D8 | PD16 | F |
| ISC | ISC_D9 | PC3 | F |
| ISC | ISC_D9 | PC18 | C |
| ISC | ISC_D9 | PD5 | E |
| ISC | ISC_D9 | PD17 | F |
| ISC | ISC_D10 | PB24 | F |
| ISC | ISC_D10 | PC19 | C |
| ISC | ISC_D10 | PD4 | E |
| ISC | ISC_D10 | PD18 | F |
| ISC | ISC_D11 | PB25 | F |

Table 49-1. I/O Lines (Continued)

| Instance | Signal | I/O Line | Peripheral |
|---|---|---|---|
| ISC | ISC_D11 | PC20 | C |
| ISC | ISC_D11 | PD3 | E |
| ISC | ISC_D11 | PD19 | F |
| ISC | ISC_FIELD | PC8 | F |
| ISC | ISC_FIELD | PC25 | C |
| ISC | ISC_FIELD | PD18 | E |
| ISC | ISC_FIELD | PD23 | F |
| ISC | ISC_HSYNC | PC6 | F |
| ISC | ISC_HSYNC | PC23 | C |
| ISC | ISC_HSYNC | PD17 | E |
| ISC | ISC_HSYNC | PD22 | F |
| ISC | ISC_MCK | PC7 | F |
| ISC | ISC_MCK | PC24 | C |
| ISC | ISC_MCK | PD2 | E |
| ISC | ISC_MCK | PD11 | F |
| ISC | ISC_PCK | PC4 | F |
| ISC | ISC_PCK | PC21 | C |
| ISC | ISC_PCK | PD15 | E |
| ISC | ISC_PCK | PD20 | F |
| ISC | ISC_VSYNC | PC5 | F |
| ISC | ISC_VSYNC | PC22 | C |
| ISC | ISC_VSYNC | PD16 | E |
| ISC | ISC_VSYNC | PD21 | F |

### 49.4.2 Power Management

The peripheral clock is not continuously provided to the ISC. The programmer must first enable the ISC clock in the Power Management Controller (PMC) before using the ISC.

### 49.4.3 Interrupt Sources

The ISC interrupt line is connected on one of the internal sources of the Interrupt Controller. Using the ISC interrupt requires the Interrupt Controller to be programmed first.

Table 49-2. Peripheral IDs

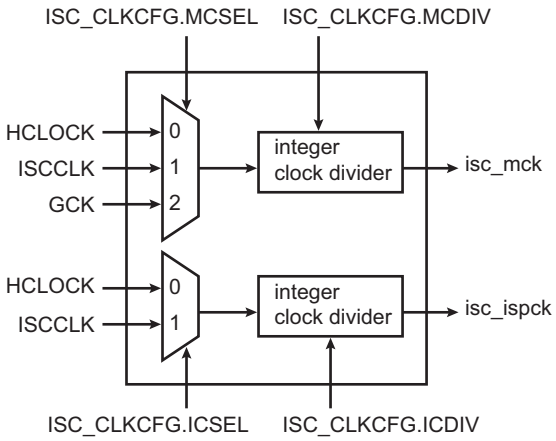
| Instance | ID |
|---|---|
| ISC | 46 |

## 49.5 Functional Description

### 49.5.1 ISC Clock Management

The ISC module provides the isc_mck output clock to the image sensor. The isc_mck clock has three selectable clock sources (ISC_CLKCFG.MCSEL field) and one programmable clock divider (ISC_CLKCFG.MCDIV). The clock is enabled using the ISC_CLKEN.MCEN field. The isc_mck is driven by the ISC and is the external reference clock of the CMOS sensor.

**Figure 49-9. Clock Divider Block Diagram**




The ISC digital pipeline requires internally a functional clock named isc_ispck. This clock is also fully programmable. This isc_ispck is enabled using the ISC_CLKEN.ICEN field. This clock is mandatory for ISC operation. The ISC module is designed to accept input signals that are asynchronous to the isc_ispck. Synchronization is done internally as long as the following relationship holds:

isc_pck frequency is less than or equal to isc_ispck, and isc_ispck is greater than or equal to HCLOCK.

#### 49.5.1.1 Software Requirement

A software write operation to the ISC_CLKEN or ISC_CLKDIS register requires double clock domain synchronization and is not permitted when the ISC_CLKSR.SIP is asserted.

### 49.5.2 Parallel Interface Timing Description

The parallel interface protocol supports two operating modes.

**Figure 49-10. Free-Running Pixel Clock**

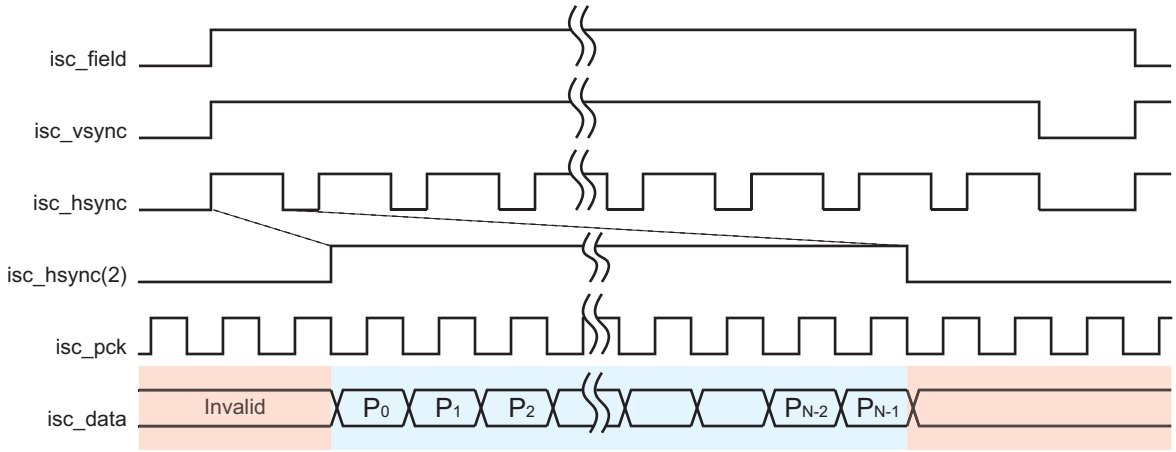
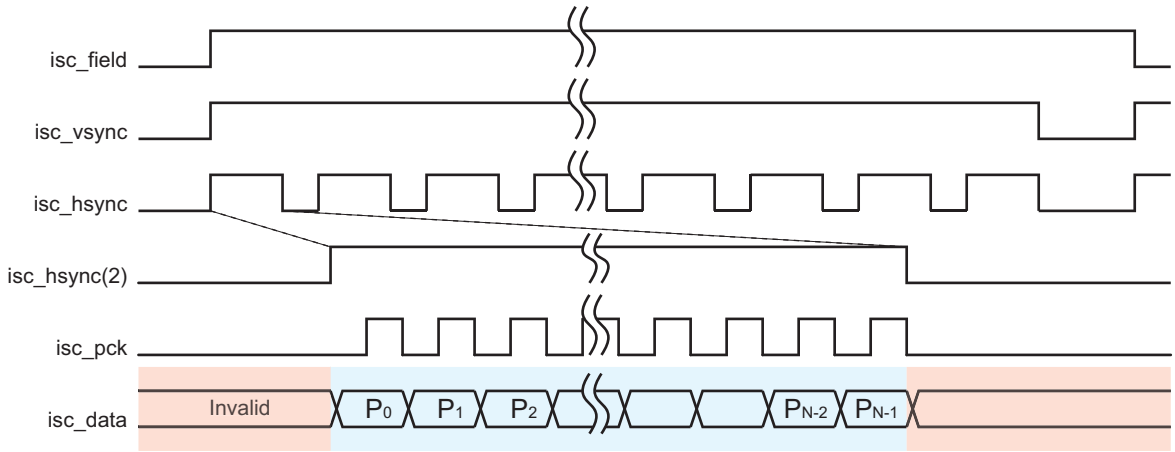



**Figure 49-11. Gated Pixel Clock**




### 49.5.3 BT.601/656/1120 Embedded Timing Synchronization Operation

The ISC module supports embedded synchronization decoding. When the ISC_PFE_CFG0.CCIR656 field is set, the decoder is activated and signals isc_vsync isc_hsync are not used to decode the valid pixels. If the CCIR10_8N is set, the bitstream is 10 bits wide, otherwise it is only 8 bits wide. When the ISC_PFE_CFG0.CCIR_CRC is set, the decoder automatically corrects the error.

**Figure 49-12. Field/Segment Timing Relationship for Interlaced and Segmented Frame Systems**

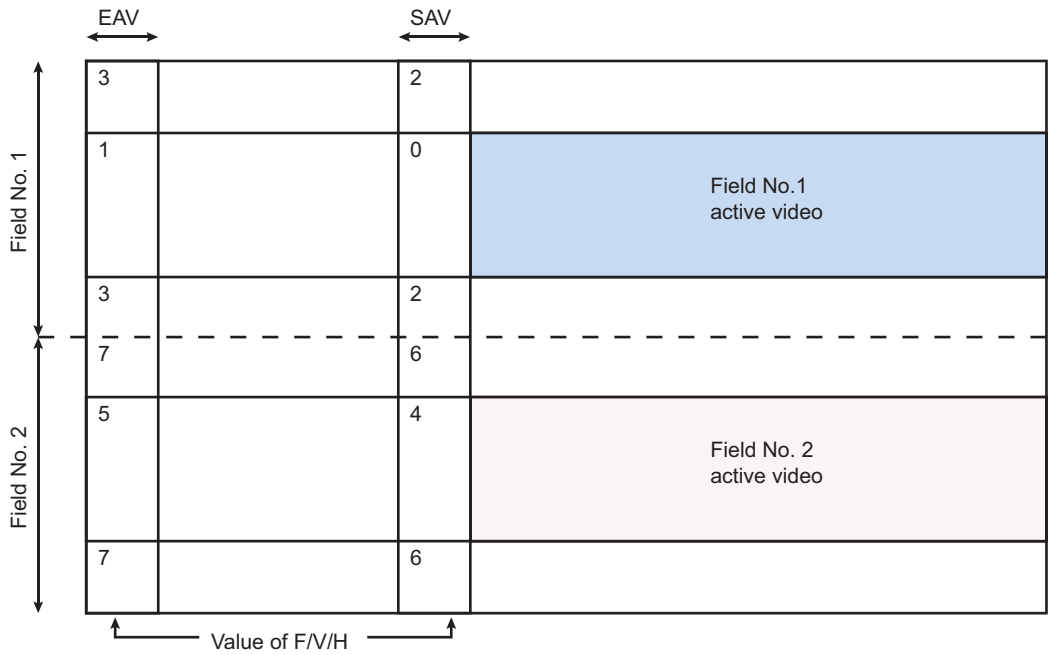
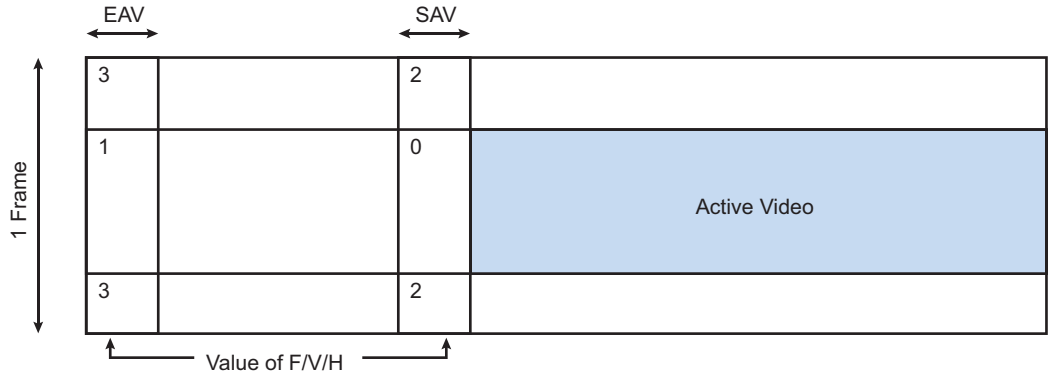



**Figure 49-13. Frame Timing Relationship for Progressive Systems**




### 49.5.4 Parallel Interface External Sensor Connections

#### 49.5.4.1 YCbCr, 10-bit CCIR656 with Embedded Synchronization

This mode is activated when fields ISC_PFE_CFG0.CCIR656 and ISC_PFE_CFG0.CCIR10_8N are both set.

| Interface Bit | First Word | Second Word | Third Word | Fourth Word |
|---|---|---|---|---|
| isc_data[11](MSB) | 1 | 0 | 0 | 1 |
| isc_data[10] | 1 | 0 | 0 | F |
| isc_data[9] | 1 | 0 | 0 | V |
| isc_data[8] | 1 | 0 | 0 | H |
| isc_data[7] | 1 | 0 | 0 | P3 |
| isc_data[6] | 1 | 0 | 0 | P2 |
| isc_data[5] | 1 | 0 | 0 | P1 |

| Interface Bit | First Word | Second Word | Third Word | Fourth Word |
|---|---|---|---|---|
| isc_data[4] | 1 | 0 | 0 | P0 |
| isc_data[3] | 1 | 0 | 0 | 0 |
| isc_data[2] | 1 | 0 | 0 | 0 |
| isc_data[1] | not used | not used | not used | not used |
| isc_data[0] | not used | not used | not used | not used |

#### 49.5.4.2 YCbCr, 8-bit CCIR656 with Embedded Synchronization

This mode is activated when field ISC_PFE_CFG0.CCIR656 is set and field ISC_PFE_CFG0.CCIR10_8N is cleared.

| Interface Bit | First Word | Second Word | Third Word | Fourth Word |
|---|---|---|---|---|
| isc_data[11](MSB) | 1 | 0 | 0 | 1 |
| isc_data[10] | 1 | 0 | 0 | F |
| isc_data[9] | 1 | 0 | 0 | V |
| isc_data[8] | 1 | 0 | 0 | H |
| isc_data[7] | 1 | 0 | 0 | P3 |
| isc_data[6] | 1 | 0 | 0 | P2 |
| isc_data[5] | 1 | 0 | 0 | P1 |
| isc_data[4] | 1 | 0 | 0 | P0 |
| isc_data[3] | not used | not used | not used | not used |
| isc_data[2] | not used | not used | not used | not used |
| isc_data[1] | not used | not used | not used | not used |
| isc_data[0] | not used | not used | not used | not used |

#### 49.5.4.3 RAW Bayer Parallel Interface

The table below shows how to connect the data bus of a RAW Bayer sensor.

| Interface | Bayer 12-bit | Bayer 11-bit | Bayer 10-bit | Bayer 9-bit | Bayer 8-bit |
|---|---|---|---|---|---|
| isc_data[11](MSB) | DOUT[11] | DOUT[10] | DOUT[9] | DOUT[8] | DOUT[7] |
| isc_data[10] | DOUT[10] | DOUT[9] | DOUT[8] | DOUT[7] | DOUT[6] |
| isc_data[9] | DOUT[9] | DOUT[8] | DOUT[7] | DOUT[6] | DOUT[5] |
| isc_data[8] | DOUT[8] | DOUT[7] | DOUT[6] | DOUT[5] | DOUT[4] |
| isc_data[7] | DOUT[7] | DOUT[6] | DOUT[5] | DOUT[4] | DOUT[3] |
| isc_data[6] | DOUT[6] | DOUT[5] | DOUT[4] | DOUT[3] | DOUT[2] |
| isc_data[5] | DOUT[5] | DOUT[4] | DOUT[3] | DOUT[2] | DOUT[1] |
| isc_data[4] | DOUT[4] | DOUT[3] | DOUT[2] | DOUT[1] | DOUT[0] |
| isc_data[3] | DOUT[3] | DOUT[2] | DOUT[1] | DOUT[0] | Not Used |

| Interface | Bayer 12-bit | Bayer 11-bit | Bayer 10-bit | Bayer 9-bit | Bayer 8-bit |
|---|---|---|---|---|---|
| isc_data[2] | DOUT[2] | DOUT[1] | DOUT[0] | Not Used | Not Used |
| isc_data[1] | DOUT[1] | DOUT[0] | Not Used | Not Used | Not Used |
| isc_data[0] | DOUT[0] | Not Used | Not Used | Not Used | Not Used |

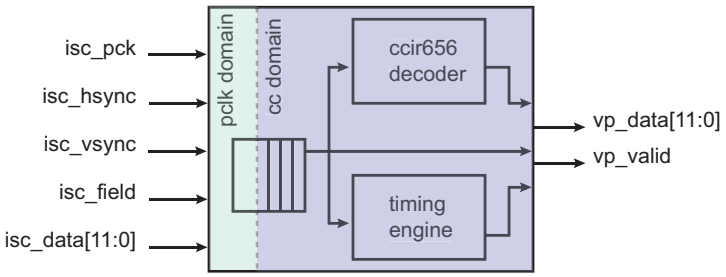
#### 49.5.4.4 Monochrome Parallel Interface

The table below shows how to connect the data bus of a Monochrome sensor.

| Interface | Mono 12-bit | Mono 11-bit | Mono 10-bit | Mono 9-bit | Mono 8-bit |
|---|---|---|---|---|---|
| isc_data[11](MSB) | DOUT[11] | DOUT[10] | DOUT[9] | DOUT[8] | DOUT[7] |
| isc_data[10] | DOUT[10] | DOUT[9] | DOUT[8] | DOUT[7] | DOUT[6] |
| isc_data[9] | DOUT[9] | DOUT[8] | DOUT[7] | DOUT[6] | DOUT[5] |
| isc_data[8] | DOUT[8] | DOUT[7] | DOUT[6] | DOUT[5] | DOUT[4] |
| isc_data[7] | DOUT[7] | DOUT[6] | DOUT[5] | DOUT[4] | DOUT[3] |
| isc_data[6] | DOUT[6] | DOUT[5] | DOUT[4] | DOUT[3] | DOUT[2] |
| isc_data[5] | DOUT[5] | DOUT[4] | DOUT[3] | DOUT[2] | DOUT[1] |
| isc_data[4] | DOUT[4] | DOUT[3] | DOUT[2] | DOUT[1] | DOUT[0] |
| isc_data[3] | DOUT[3] | DOUT[2] | DOUT[1] | DOUT[0] | Not Used |
| isc_data[2] | DOUT[2] | DOUT[1] | DOUT[0] | Not Used | Not Used |
| isc_data[1] | DOUT[1] | DOUT[0] | Not Used | Not Used | Not Used |
| isc_data[0] | DOUT[0] | Not Used | Not Used | Not Used | Not Used |

### 49.5.5 Parallel Front End (PFE) Module

**Figure 49-14. PFE Block Diagram**




The Parallel Front End module performs data resampling across clock domain boundary. It includes a CCIR656 decoder used to convert a standard ITU-R BT.656 stream to 24-bit digital video. It also generates pixels, syncs flags and valid signals to the main video pipeline. It ouputs field, video and synchronization signals. The PFE can optionally crop and limit the incoming pixel stream to a predefined horizontal and vertical value. By default, the PFE only relies on the cmos sensor horizontal and vertical reference to sample the incoming pixel stream. A pixel is sampled if, and only if, the vertical and horizontal synchronizations are valid and a pixel clock edge is detected.

ISC_PFE_CFG0.BPS shows the number of bits per sample. The PFE module outputs a 12-bit data on the vp_data[11:0] bus, and asserts the vp_valid signal when the data can be sampled.

| PFE VP_DATA Mapping | Raw Bayer 12-bit | Raw Bayer 10-bit | YUV422 8-bit | YUV422 10-bit | Mono 12-bit |
|---|---|---|---|---|---|
| VP_DATA[11] | RGGB[11] | RGGB[9] | YC422[7] | YC422[9] | Y[11] |
| VP_DATA[10] | RGGB[10] | RGGB[8] | YC422[6] | YC422[8] | Y[10] |
| VP_DATA[9] | RGGB[9] | RGGB[7] | YC422[5] | YC422[7] | Y[9] |
| VP_DATA[8] | RGGB[8] | RGGB[6] | YC422[4] | YC422[6] | Y[8] |
| VP_DATA[7] | RGGB[7] | RGGB[5] | YC422[3] | YC422[5] | Y[7] |
| VP_DATA[6] | RGGB[6] | RGGB[4] | YC422[2] | YC422[4] | Y[6] |
| VP_DATA[5] | RGGB[5] | RGGB[3] | YC422[1] | YC422[3] | Y[5] |
| VP_DATA[4] | RGGB[4] | RGGB[2] | YC422[0] | YC422[2] | Y[4] |
| VP_DATA[3] | RGGB[3] | RGGB[1] | YC422[7] or 0 | YC422[1] | Y[3] |
| VP_DATA[2] | RGGB[2] | RGGB[0] | YC422[6] or 0 | YC422[0] | Y[2] |
| VP_DATA[1] | RGGB[1] | RGGB[9] or 0 | YC422[5] or 0 | YC422[9] or 0 | Y[1] |
| VP_DATA[0] | RGGB[0] | RGGB[8] or 0 | YC422[4] or 0 | YC422[8] or 0 | Y[0] |

Note: When ISC_PFE_CFG0.REP is set, missing VP_DATA LSBs are replaced with replicated LSBs of the incoming stream, otherwise they are forced to zero.

The PFE module also includes logic to synchronize capture request with the incoming pixel stream. Two operating modes are available: Single Shot and Continuous Acquisition. When the ISC_PFE_CFG0.CONT field is cleared, the ISC transfers a single image to memory,

**Figure 49-15. Single Shot Mode**

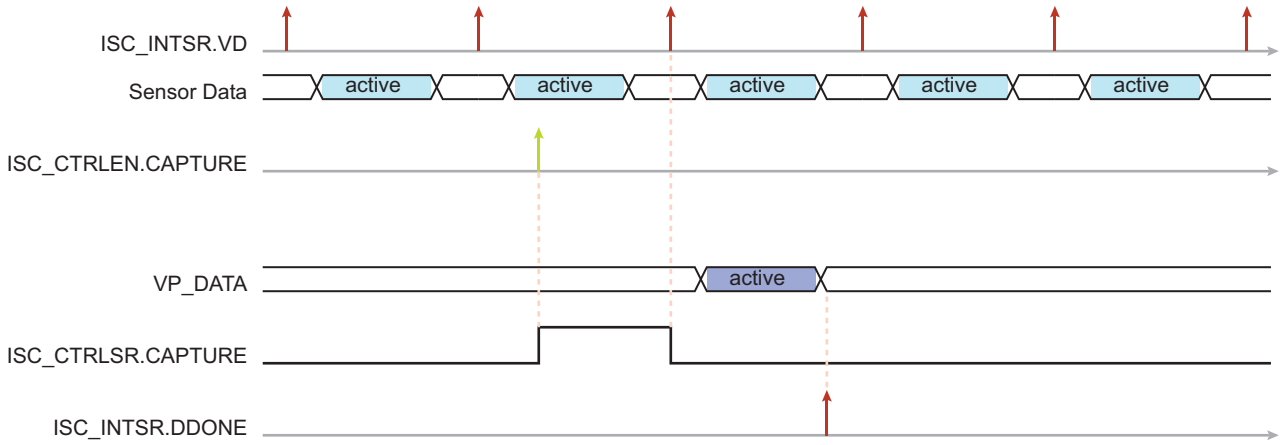
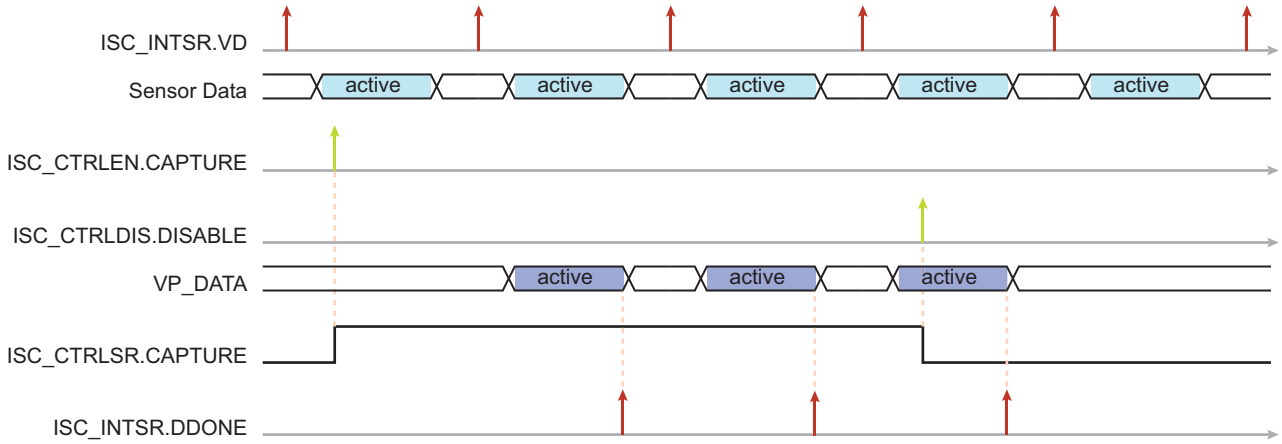
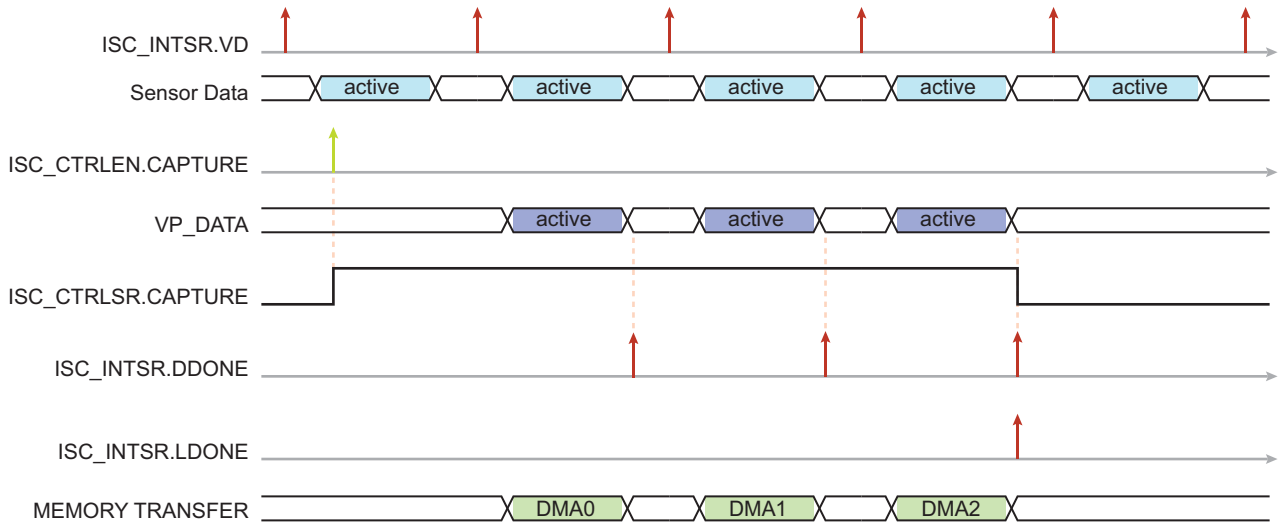


**Figure 49-16. Continuous Acquisition Mode**




When Continuous Acquisition mode is activated (ISC_PFE_CFG0.CONT is set), the data transfer terminates when either a DMA end of list is reached, a software disable is performed or a software reset is activated. The ISC_INTSR.DDONE is set at the end of the DMA data transfer.

**Figure 49-17. Continuous Acquisition, DMA Terminated**



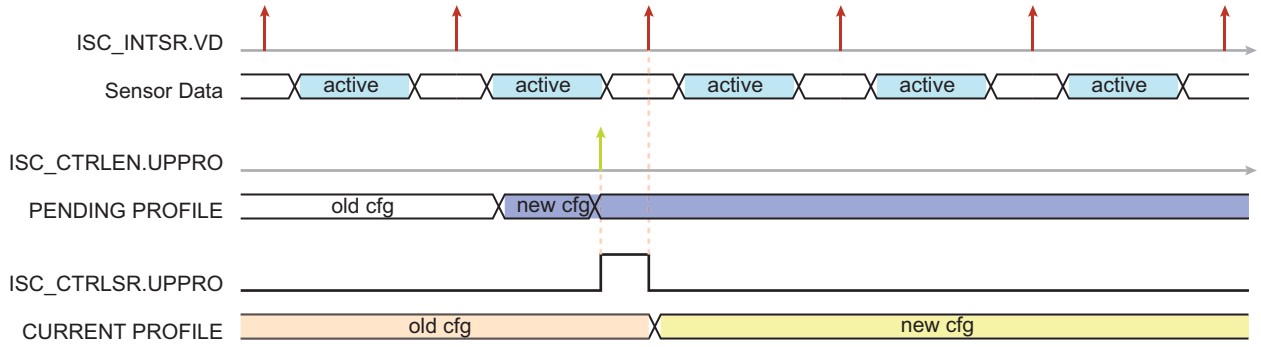

The linked list DMA transfer is terminated when an item of the list is programmed with the field ISC_DCTRL.NDE cleared or when the ISC_DNDA.NDA field is equal to zero. This configuration also clears the ISC_CTRLSR.CAPTURE field and sets the ISC_INTSR.LDONE interrupt flag.

The linked list DMA transfer starts if the field ISC_DCTRL.NDE is set and if the field ISC_DNDA.NDA is different from zero.

#### 49.5.5.1 Update the ISC Profile

Each ISC register is double-buffered to simplify the software configuration and the synchronization with the associated frame buffer. When the configuration of the ISC is modified, the ISC_CTRLEN.UPPRO field must be set to transfer the configuration from the input buffer to the ISC video pipeline.

**Figure 49-18. Update Profile Timing Diagram**



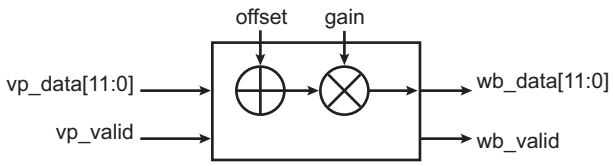

#### 49.5.5.2 Software Requirements

Writing to the ISC_CTRLEN or ISC_CTRLDIS register requires a double domain synchronization, so it is forbidden to write these registers when the ISC_CTRLSR.SIP bit is asserted.

### 49.5.6 White Balance (WB) Module

The White Balance (WB) module captures the vp_data[11:0] bus from the PFE module when the vp_valid signal is asserted, and it generates a wb_data[11:0] data along with its validity signal wb_valid. When ISC_WB_CTRL.ENABLE is set, each Bayer color component (R, Gr, B, Gb) can be manually adjusted using an offset and a gain. The Bayer pattern is adjustable using the ISC_WB_CFG.BAYCFG field.

**Figure 49-19. WB Block Diagram**



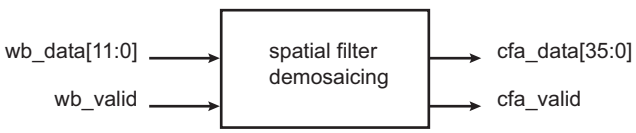

There are four {gain, offset} sets for each Bayer component. The output value is clipped.

| ISC_WB_CTRL.ENABLE | WB_DATA Slice | Value |
|---|---|---|
| 0 | wb_data[11:0] | vp_data[11:0] |
| 1 | wb_data[11:0] | clipped((vp_data[11:0]+offset)*gain) |

### 49.5.7 Color Filter Array (CFA) Interpolation Module

In a single-sensor system, each cell on the sensor has a specific color filter and microlens positioned above it. The raw data obtained from the sensor do not have the full R/G/B information at each cell position. Color interpolation is required to retrieve the missing components. The CFA module samples the wb_data[11:0] 12-bit bus when wb_valid is asserted and generates a 36-bit width data bus cfa_data[35:0] with the validity bit cfa_valid.

**Figure 49-20. CFA Block Diagram**



| ISC_CFA_CTRL.ENABLE | CFA_DATA Slice | Value |
|---|---|---|
| 0 | cfa_data[35:24] | wb_data[11:0] |
| | cfa_data[23:12] | wb_data[11:0] |
| | cfa_data[11:0] | wb_data[11:0] |
| 1 | cfa_data[35:24] | R = spatial_filter_R(wb_data[11:0]) |
| | cfa_data[23:12] | G = spatial_filter_G(wb_data[11:0]) |
| | cfa_data[11:0] | B = spatial_filter_B(wb_data[11:0]) |

The filter kernel size is 5, and requires two additional lines to initialize the filter. When ISC_CFA_CFG.EITPOL is set, the missing information is interpolated from the nearest neighbor. If ISC_CFG_CFG.EITPOL is cleared, only valid pixels are used to initialize the filter kernel, but the output number of lines is less than the input number of lines. In that case, four lines are consumed for filling the kernel.

#### 49.5.7.1 Frame Size Requirement when Edge Interpolation is Off, ISC_CFA_CFG.EITPOL Cleared

- Minimum number of rows (in): 5
- Minimum number of columns (in): 5
- Number or rows after CFA: Number of rows (in) - 4
- Number of columns after CFA: Number of columns (in) - 4

#### 49.5.7.2 Frame Size Requirement when Edge Interpolation is On, ISC_CFA_CFG.EITPOL Set

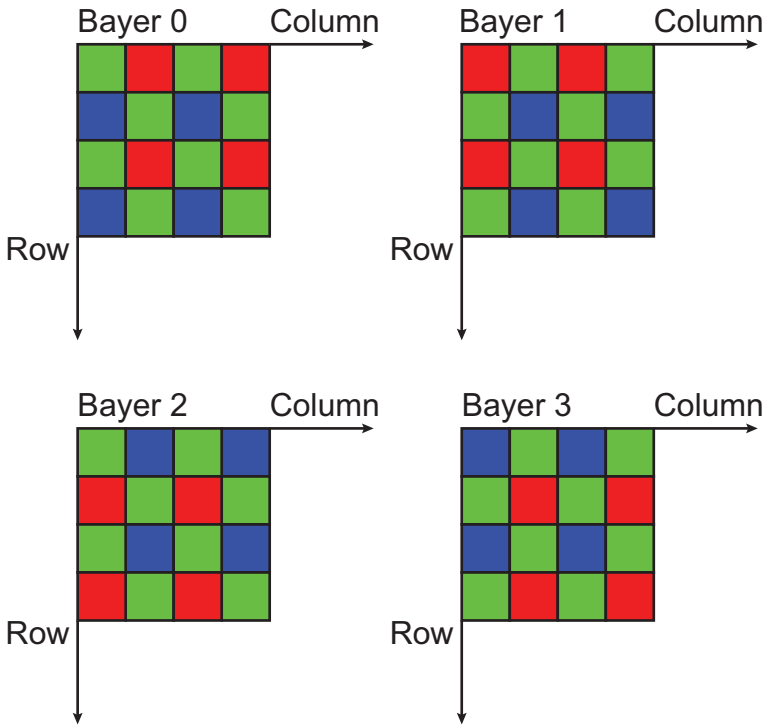
- Minimum number of rows (in): 3
- Minimum number of columns (in): 3
- Number of rows after CFA: Number of rows (in)
- Number of columns after CFA: Number of columns (in)

#### 49.5.7.3 Bayer Mode and Edge Interpolation Description

When Edge Interpolation mode (ISC_CFA_CFG.EITPOL) is activated, dummy lines are generated using rows and columns replication.

The CFA module supports four sensor alignments using the ISC_CFA_CFG.BAYCFG field. Refer to Figure 49-21.

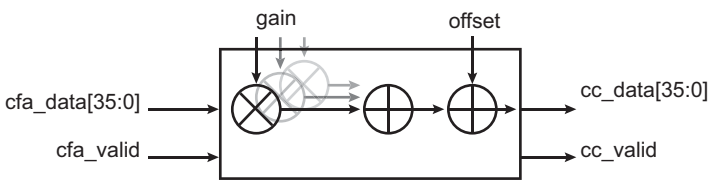
**Figure 49-21. Supported Color Filter Array Patterns**




### 49.5.8 Color Correction (CC) Module

RGB color correction is used to compensate for cross color bleeding in the filter used with the image sensor. The module samples the cfa_data[35:0] 36-bit bus when cfa_valid is asserted and generate a cc_data[35:0] 36-bit wide bus and a cc_valid signal.

**Figure 49-22. CC Block Diagram**




There are three {gain, offset} sets for color component R, G, B.

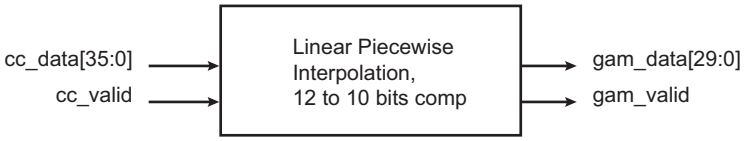
| ISC_CC_CTRL.ENABLE | CC_DATA Slice | Value |
|---|---|---|
| 0 | cc_data[35:24] | cfa_data[35:24] |
| | cc_data[23:12] | cfa_data[23:12] |
| | cc_data[11:0] | cfa_data[11:0] |
| 1 | cc_data[35:24] | R=clipped(sum(cfa_data_x * gain_Rx) + offset_R) |
| | cc_data[23:12] | G=clipped(sum(cfa_data_x * gain_Gx) + offset_G) |
| | cc_data[11:0] | B=clipped(sum(cfa_data_x * gain_Bx) + offset_B) |

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} RRGAIN & RGGAIN & RBGAIN \\ GRGAIN & GGGAIN & GBGAIN \\ BRGAIN & BGGAIN & BBGAIN \end{bmatrix} \times \begin{bmatrix} \text{cfa\_data[35:24]} \\ \text{cfa\_data[23:12]} \\ \text{cfa\_data[11:0]} \end{bmatrix} + \begin{bmatrix} ROFST \\ GOFST \\ BOFST \end{bmatrix}$$

### 49.5.9 Gamma Curve (GAM) Module

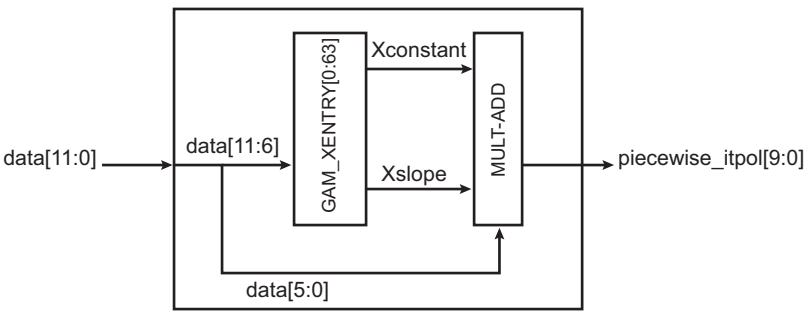
The GAM module samples the cc_data[35:0] bus when cc_valid is asserted, and generates gam_data[29:0] 30-bit width data along with the validity signal gam_valid. Imaging devices have non-linear characteristics, but the transfer function is approximated by a power function. The intensity of each of the linear RGB components is transformed to a non-linear signal through the use of the gamma correction submodule. The power function is linearly interpolated using 64 breakpoints. This also performs a 12-bit to 10-bit compression. The polynomial for the linear interpolation between breakpoints is ***i*** and ***i*** +1. Consequently, for each breakpoint, two values are required: constant and slope. The table values are programmable through the user interface when the gamma correction module is disabled (ISC_GAM_CTRL.ENABLE is cleared). ISC_GAM_RENTRY is used for Red gamma correction. ISC_GAM_GENTRY is used for Green gamma correction. ISC_GAM_BENTRY is used for Blue gamma correction. Each table entry is composed of a 10-bit (signed) slope and a 10-bit constant.

**Figure 49-23. GAM Block Diagram**




| ISC_GAM_CTRL.ENABLE | ISC_GAM_CTRL.XLUT | GAM_DATA Slice | Value |
|---|---|---|---|
| 0 | 0 | gam_data[29:0] | cc_data[29:0] |
| 1 | 0 | gam_data[29:20] | cc_data[35:26] |
| | | gam_data[19:10] | cc_data[23:14] |
| | | gam_data[9:0] | cc_data[11:2] |
| 1 | 1 | gam_data[29:20] | R=piecewise_itpol(cc_data_r[35:24]) |
| | | gam_data[19:10] | G=piecewise_itpol(cc_data_r[23:12]) |
| | | gam_data[9:0] | B=piecewise_itpol(cc_data_r[11:0]) |

**Figure 49-24. Piecewise Linear Interpolation Block Diagram**



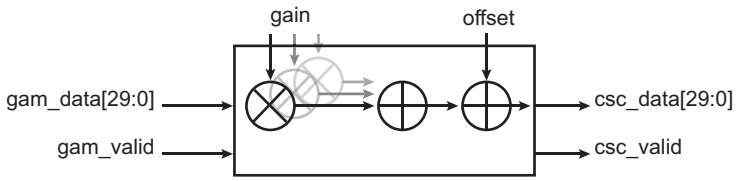

The interpolation consists of three tables that store the function values GAM_XENTRY[0:63] where X stands for R, G and B. The input of the table has six bits. It outputs a slope and a constant. The slope is later multiplied by the

data lsb (6-bit) and added to a constant. The final value is the gamma-corrected value of the input. This module performs a 12-to-10 compression.

### 49.5.10 Color Space Conversion (CSC) Module

By converting an image from RGB to YCbCr color space, it is possible to separate Y, Cb and Cr information. The CSC samples the gam_data[29:0] 30-bit data bus, extracts YCbCr information from the sampled data, and then generates the color-converted data csc_data[29:0] and the validity signal csc_valid.

**Figure 49-25. CSC Block Diagram**



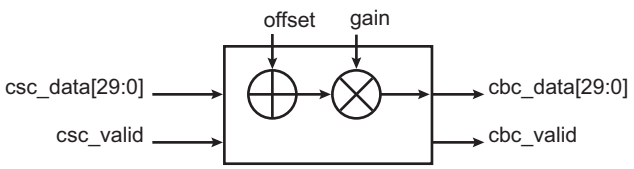

| ISC_CSC_CTRL.ENABLE | CSC_DATA Slice | Value |
|---|---|---|
| 0 | csc_data[29:0] | gam_data[29:0] |
| 1 | csc_data[29:20] | Y = clipped(sum(gam_data_x * gain_Yx) + offset_y << 2) |
| | csc_data[19:10] | Cb = clipped(sum(gam_data_x * gain_Cbx)+offset_cb << 2) |
| | csc_data[9:0] | Cr = clipped(sum(gam_data_x * gain_Crx)+offset_cr << 2) |

$$\begin{bmatrix} Y \\ CB \\ CR \end{bmatrix} = \begin{bmatrix} YR & YG & YB \\ CBR & CBG & CBB \\ CRR & CRG & CRB \end{bmatrix} \times \begin{bmatrix} \text{gam\_data[29:20]} \\ \text{gam\_data[19:10]} \\ \text{gam\_data[9:0]} \end{bmatrix} + \begin{bmatrix} YOFST \\ CBOFST \\ CROFST \end{bmatrix}$$

### 49.5.11 Contrast and Brightness

Luminance is adjusted through the use of Brightness Offset and Contrast Gain. Chrominance is left unchanged. The CBC samples the csc_data[29:0] 30-bit bus when csc_valid is asserted and generates cbc_data[29:0] with the validity signal cbc_valid.

**Figure 49-26. CBC Block Diagram**

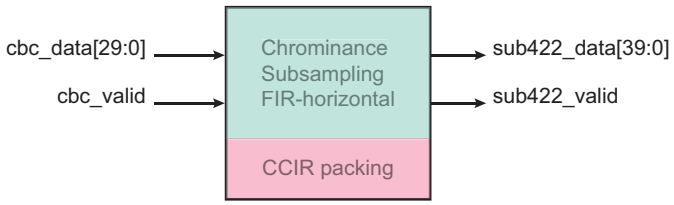



| ISC_CBC_CTRL.ENABLE | ISC_CBC_CFG.CCIR | CBC_DATA Slice | Value |
|---|---|---|---|
| 0 | 0 | cbc_data[29:0] | csc_data[29:0] |
| 1 | 0 | cbc_data[29:20] | Y = clipped((csc_data[29:20]+offset)*gain) |
| | | cbc_data[19:10] | Cb = csc_data[19:10] |
| | | cbc_data[9:0] | Cr = csc_data[9:0] |
| 1 | 1 | cbc_data[29:10] | 0 |
| | | cbc_data[9:0] | ccir656 stream with luminance correction |

### 49.5.12 4:4:4 To 4:2:2 Chrominance Horizontal Subsampler (SUB422) Module

The color space conversion output stream is a full-bandwidth YCbCr 4:4:4 signal. The chrominance subsampling divides the horizontal chrominance sampling rate by two. A horizontal low pass filter is applied to avoid aliasing effect. The SUB422 module samples 444 full scale YCbCr cbc_data[29:0] 30-bit data, performs horizontal subsampling and generates the sub422_data[39:0] 40-bit data bus with its validity signal sub422_valid.

**Figure 49-27. SUB422 Block Diagram**




| ISC_SUB422_CTRL.ENABLE | ISC_SUB422_CFG.CCIR | SUB422_DATA Slice | Value |
|---|---|---|---|
| 0 | 0 | sub422_data[29:0] | cbc_data[29:0] |
| 1 | 0 | sub422_data[39:30] | Y1= cbc_data1[29:20] |
| | | sub422_data[29:20] | Y0 = cbc_data0[29:20] |
| | | sub422_data[19:10] | Cb = filter_hor(cbc_data[19:10]) |
| | | sub422_data[9:0] | Cr = filter_hor(cbc_data[9:0]) |
| 1 | 1 | sub422_data[39:30] | Y1 = cbc_data[9:0] |
| | | sub422_data[29:20] | Y0 = cbc_data[9:0] |
| | | sub422_data[19:10] | Cb = cbc_data[9:0] |
| | | sub422_data[9:0] | Cr = cbc_data[9:0] |

The filter_hor function included in the sub422 module is the chrominance horizontal filter.

| sub422 data slice | YCbCr mapping |
|---|---|
| sub422_data[39:30] | Y1 (sample *n*) |
| sub422_data[29:20] | Y0 (sample *n-1*) |
| sub422_data[19:10] | Cb (from filter) |
| sub422_data[9:0] | Cr (from filter) |

The filter chrominance position is selectable through the use of the ISC_SUB422_CFG.FILTER field.

**Figure 49-28. Cosited Filter Configuration**

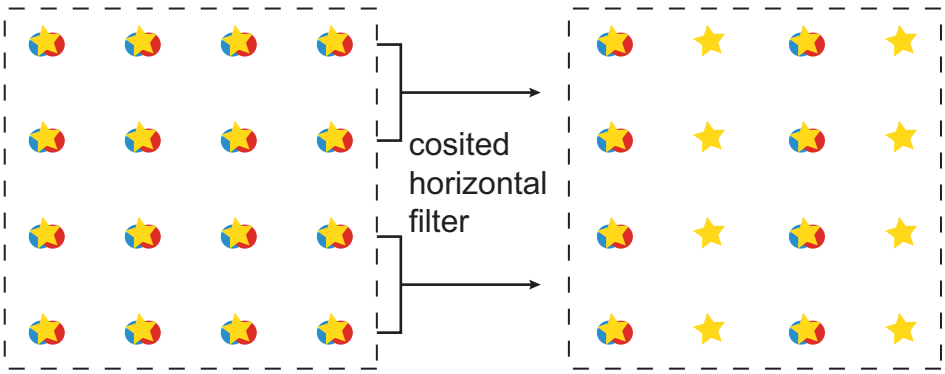
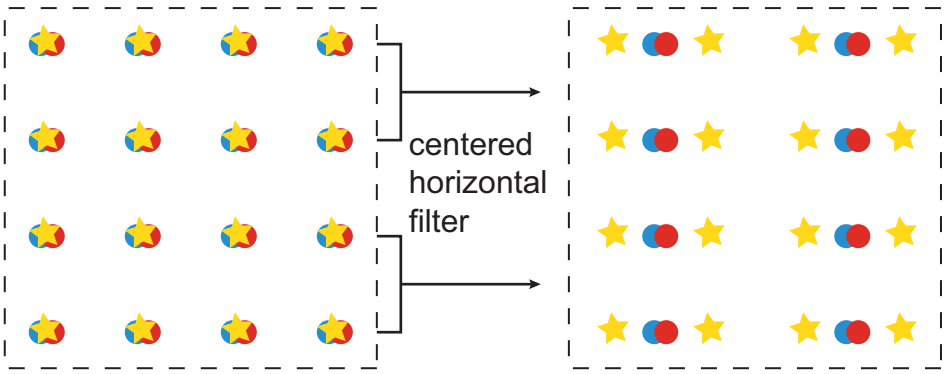



**Figure 49-29. Centered Filter Configuration**



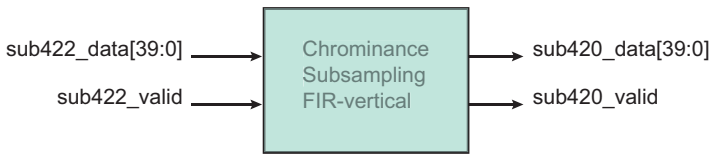

The SUB422 module performs luminance and chrominance packing. When the line length is odd, the missing luminance is a copy of the last but one luminance. It also means that the final dma stream written to memory is equal to the original horizontal size plus one when the line length is odd.

| SUB422_DATA Slice | Line Length Even | Line Length Odd |
|---|---|---|
| sub422_data[39:30] | Y(n) | Y(n-1) |
| sub422_data[29:20] | Y(n-1) | Y(n-1) |
| sub422_data[19:10] | Cb (filtered) | Cb (filtered) |
| sub422_data[9:0] | Cr (filtered) | Cr (filtered) |

### 49.5.13 4:2:2 To 4:2:0 Chrominance Vertical Subsampler (SUB420) Module

The chrominance subsampling divides the vertical chrominance sampling rate by two. A vertical low pass filter is applied to avoid aliasing effect. Two different filters are used when the source frame is interlaced, and the filter configuration depends on the field value (the field is propagated in the video pipeline).

**Figure 49-30. SUB420 Block Diagram**



The SUB420 module samples the sub422_data[39:0] 40-bit data when sub422_valid is asserted, then it performs a vertical subsampling and generates a valid sub420_data[39:0] 40-bit word and the corresponding sub420_valid signal.

| ISC_SUB420_CTRL.ENABLE | SUB420_DATA Slice | Value |
|---|---|---|
| 0 | sub420_data[39:0] | sub422_data[39:0] |
| 1 | sub420_data[39:30] | Y1 = sub422_data[39:30] |
| | sub420_data[29:20] | Y0 = sub422_data[29:20] |
| | sub420_data[19:10] | Cb = filter_ver(sub422[19:10]) |
| | sub420_data[9:0] | Cr = filter_ver(sub422[9:0]) |

The vertical filter is a two-tap filter; for progressive content the coefficient i {1, 1}. When an interlaced field is downsampled, the coefficients are different between the top and the bottom fields.

**Figure 49-31. Vertical Chrominance Filter for Progressive Content (Cosited Chrominance Example)**

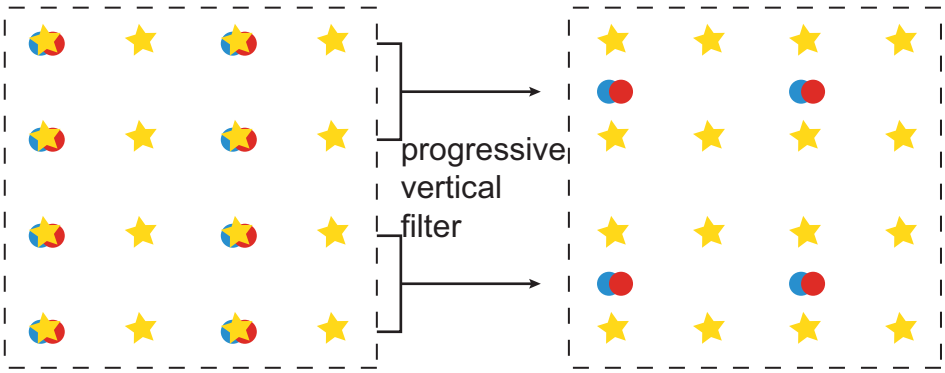



**Figure 49-32. Field-dependent Chrominance Filter for Interlaced Content (Cosited Chrominance Example)**

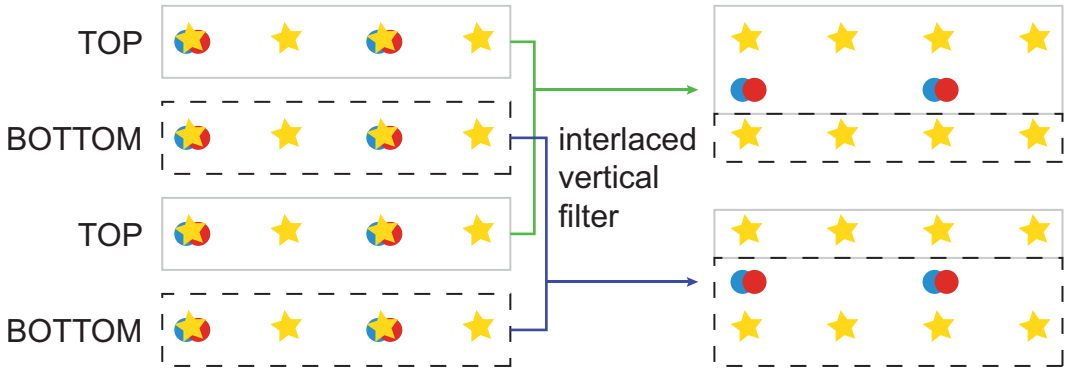



**Table 49-3. Filter Configuration**

| ISC_SUB420_CTRL.FILTER | Field | Filter Configuration |
|---|---|---|
| 0 | progressive | {1, 1} |
| 1 | 0 (TOP) | {3, 1} |
| | 1 (BOTTOM) | {1, 3} |

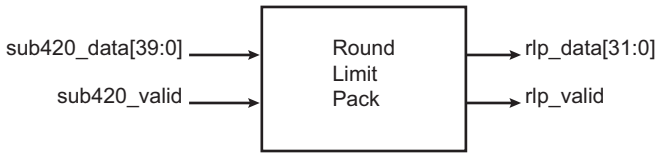
**Table 49-4. Output Line Length Configuration**

| SUB420 Input Number of Rows | SUB420 Luminance Rows | SUB420 Chrominance Rows |
|---|---|---|
| M rows, M odd | M rows | (M+1)/2 rows |
| M rows, M even | M rows | M/2 rows |

### 49.5.14 Rounding, Limiting and Packing (RLP) Module

This module is used to round, limit and pack in the incoming pixel stream before the DMA master module. The RLP samples the sub420_data[39:0] 40-bit data bus and generates rlp_data[31:0] 32-bit data words with the associated validity signal rlp_valid.

**Figure 49-33. RLP Block Diagram**




| ISC_RLP_CFG | RLP_DATA Slice | Value |
|---|---|---|
| DAT8 | rlp_data[31:8] | 0 |
| | rlp_data[7:0] | sub420_data[11:4] |
| DAT9 | rlp_data[31:9] | 0 |
| | rlp_data[8:0] | sub420_data[11:3] |
| DAT10 | rlp_data[31:10] | 0 |
| | rlp_data[9:0] | sub420_data[11:2] |
| DAT11 | rlp_data[31:11] | 0 |
| | rlp_data[10:0] | sub420_data[11:1] |
| DAT12 | rlp_data[31:12] | 0 |
| | rlp_data[11:0] | sub420_data[11:0] |
| DATY8 | rlp_data[31:8] | 0 |
| | rlp_data[7:0] | Y = rounded(sub420_data[29:22]) |
| DATY10 | rlp_data[31:8] | 0 |
| | rlp_data[7:0] | Y = sub420_data[29:20]) |
| ARGB444 | rlp_data[31:16] | 0 |
| | rlp_data[15:12] | A = alpha[7:4] |
| | rlp_data[11:8] | R = sub420_data[29:26] |
| | rlp_data[7:4] | G = sub420_data[19:16] |
| | rlp_data[3:0] | B = sub420_data[9:6] |

| ISC_RLP_CFG | RLP_DATA Slice | Value |
|---|---|---|
| ARGB555 | rlp_data[31:16] | 0 |
| | rlp_data[15] | A = alpha[7] |
| | rlp_data[14:10] | R = sub420_data[29:25] |
| | rlp_data[9:5] | G = sub420_data[19:15] |
| | rlp_data[4:0] | B = sub420_data[9:5] |
| RGB565 | rlp_data[31:16] | 0 |
| | rlp_data[15:11] | R = sub420_data[29:25] |
| | rlp_data[10:5] | G = sub420_data[19:14] |
| | rlp_data[4:0] | B = sub420_data[9:5] |
| RGB32 | rlp_data[31:24] | A = alpha[7:0] |
| | rlp_data[23:16] | R = sub420_data[29:22] |
| | rlp_data[15:8] | G = sub420_data[19:12] |
| | rlp_data[7:0] | B = sub420_data[9:2] |
| YCbCr422, YCbCr420 | rlp_data[31:24] | Y1 = round(sub420_data[39:32]) |
| | rlp_data[23:16] | Y0 = round(sub420_data[29:22]) |
| | rlp_data[15:8] | Cb = round(sub420_data[19:12]) |
| | rlp_data[7:0] | Cr = round(sub420_data[9:2]) |
| YCbCr422, YCbCr420 | rlp_data[31:24] | Y1 = round_limit(sub420_data[39:32]) |
| | rlp_data[23:16] | Y0 = round_limit(sub420_data[29,22]) |
| | rlp_data[15:8] | Cb = round_limit(sub420_data[19,12]) |
| | rlp_data[7:0] | Cr = round_limit(sub420_data[9:2]) |
| Undefined | rlp_data[31:0] | sub420_data[31:0] |

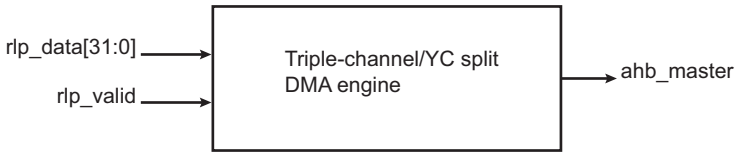
The round_limit function provides a simple method to round and limit the range of both Luminance and Chrominance signals.

| ISC_RLP_CFG | 8-bit Full Range | 8-bit Limited Range |
|---|---|---|
| Y | 0–255 | 16–235 |
| Cb | 0–255 | 16–240 |
| Cr | 0–255 | 16–240 |

### 49.5.15 DMA Interface

The descriptor-based DMA interface supports multiple buffers. A DMA stride value shows the offset between two consecutive lines (in bytes). If the stride is set to zero, the frame buffer is contiguous. When the ISC_DCTRL.WB field is set (Write Back), the DMA interface performs a single write operation to the ISC_DCTRL register, and sets ISC_DCTRL[7] to one and ISC_DCTRL[6] to the value of the frame field when interlaced content is being used. That means that interlaced fields are tagged with their relevant field values. The Write Back operation is always performed when the whole frame has been transferred to memory.

**Figure 49-34. DMA Engine Block Diagram**




| ISC_DCFG.IMODE | DMA Engine Input Data |
|---|---|
| PACKED8 | rlp_data[7:0] |
| PACKED16 | rlp_data[15:0] |
| PACKED32 | rlp_data[31:0] |
| YC422SP | rlp_data[31:0] |
| YC422P | rlp_data[31:0] |
| YC420SP | rlp_data[31:0] |
| YC420P | rlp_data[31:0] |

When a bus error is detected, an interrupt flag is set. If the error occurs on a write operation, ISC_INTSR.WERR is asserted. If the error occurs on a read operation, ISC_INTSR.RERR is asserted. The ISC_INTSR.WERRID field gives details on the first error channel identifier.

#### 49.5.15.1 Descriptor Memory Address Mapping

| ISC_DCFG.IMODE | ISC_DAD0.AD0 | ISC_DAD1.AD1 | ISC_DAD2.AD2 |
|---|---|---|---|
| PACKED8, PACKED16, PACKED32 | data address | not used | not used |
| YC422SP | Y address | CbCr address | not used |
| YC422P | Y address | Cb address | Cr address |
| YC420SP | Y address | CbCr address | not used |
| YC420P | Y address | Cb address | Cr address |

#### 49.5.15.2 Descriptor Memory Mapping

Three descriptor views are available. Descriptor view 0 is used when the pixel or data stream is packed. Descriptor view 1 is used for YCbCr semi-planar pixel stream. Descriptor view 2 is used for YCbCr planar pixel stream.

**Table 49-5. ISC_DCTRL.DVIEW = 0**

| Address | Register |
|---|---|
| ISC_DNDA+0x00 | ISC_DCTRL |
| ISC_DNDA+0x04 | ISC_DNDA |
| ISC_DNDA+0x08 | ISC_DAD0 |
| ISC_DNDA+0x0C | ISC_DST0 |

**Table 49-6. ISC_DCTRL.DVIEW = 1**

| Address | Register |
|---|---|
| ISC_DNDA+0x00 | ISC_DCTRL |
| ISC_DNDA+0x04 | ISC_DNDA |
| ISC_DNDA+0x08 | ISC_DAD0 |
| ISC_DNDA+0x0C | ISC_DST0 |
| ISC_DNDA+0x10 | ISC_DAD1 |
| ISC_DNDA+0x14 | ISC_DST1 |

**Table 49-7. ISC_DCTRL.DVIEW = 2**

| Address | Register |
|---|---|
| ISC_DNDA+0x00 | ISC_DCTRL |
| ISC_DNDA+0x04 | ISC_DNDA |
| ISC_DNDA+0x08 | ISC_DAD0 |
| ISC_DNDA+0x0C | ISC_DST0 |
| ISC_DNDA+0x10 | ISC_DAD1 |
| ISC_DNDA+0x14 | ISC_DST1 |
| ISC_DNDA+0x18 | ISC_DAD2 |
| ISC_DNDA+0x1C | ISC_DST2 |

### 49.5.15.3 Example: Memory Mapping for 16-bit Packed, DMA Interface IMODE = 1 at ISC_DAD0.AD0 Location

**Table 49-8. DAT8 Packing (ISC_RLP_CFG.MODE)**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAW12 | – | – | – | – | – | – | – | – | rlp_data1[7:0] | | | | | | | | – | – | – | – | – | – | – | – | rlp_data0[7:0] | | | | | | | |

**Table 49-9. DAT9 Packing (ISC_RLP_CFG.MODE)**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAW12 | – | – | – | – | – | – | – | rlp_data1[8:0] | | | | | | | | | – | – | – | – | – | – | – | rlp_data0[8:0] | | | | | | | | |

**Table 49-10. DAT10 Packing (ISC_RLP_CFG.MODE)**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAW12 | – | – | – | – | – | – | rlp_data1[9:0] | | | | | | | | | | – | – | – | – | – | – | rlp_data0[9:0] | | | | | | | | | |

**Table 49-11. DAT11 Packing (ISC_RLP_CFG.MODE)**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAW12 | – | – | – | – | – | rlp_data1[10:0] | | | | | | | | | | | – | – | – | – | – | isc_data0[10:0] | | | | | | | | | | |

**Table 49-12. DAT12 Packing (ISC_RLP_CFG.MODE)**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAW12 | – | – | – | – | rlp_data1[11:0] | | | | | | | | | | | | – | – | – | – | rlp_data0[11:0] | | | | | | | | | | | |

### 49.5.15.4 Example: Memory Mapping for 12-bit YC420SP, DMA Interface IMODE = 5

**Table 49-13. Y Channel Located at ISC_DAD0.AD0 Memory Address**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Y 8-bit | rlp_data1[31:24] | | | | | | | | rlp_data1[23:16] | | | | | | | | rlp_data0[31:24] | | | | | | | | rlp_data0[23:16] | | | | | | | |

**Table 49-14. CbCr Channel Located at ISC_DAD1.AD1 Memory Address**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CC 16-bit | rlp_data1[15:0] | | | | | | | | | | | | | | | | rlp_data0[15:0] | | | | | | | | | | | | | | | |

### 49.5.15.5 Example: Memory Mapping for 12-bit YC420P, DMA Interface IMODE = 6

**Table 49-15. Y Channel Located at ISC_DAD0.AD0 Memory Address**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Y 8-bit | rlp_data1[31:24] | | | | | | | | rlp_data1[23:16] | | | | | | | | rlp_data0[31:24] | | | | | | | | rlp_data0[23:16] | | | | | | | |

**Table 49-16. Cb Channel Located at ISC_DAD1.AD1 Memory Address**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Cb 8-bit | rlp_data3[15:8] | | | | | | | | rlp_data2[15:8] | | | | | | | | rlp_data1[15:8] | | | | | | | | rlp_data0[15:8] | | | | | | | |

**Table 49-17. Cr Channel Located at ISC_DAD2.AD2**

| Mem addr | 0x3 | | | | | | | | 0x2 | | | | | | | | 0x1 | | | | | | | | 0x0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Cr 8-bit | rlp_data3[7:0] | | | | | | | | rlp_data2[7:0] | | | | | | | | rlp_data1[7:0] | | | | | | | | rlp_data0[7:0] | | | | | | | |

### 49.5.16 Histogram Module

For each possible pixel value, the histogram counts the number of times the value was encountered in the current image. RGGB Bayer, RAW data or luminance histogram are available. There are 512 entries in the histogram entries, and each histogram bin can count up to 2^20 data. As the table entries are limited, each bin is actually a range, i.e., least significant bits are ignored. A write to ISC_CTRLEN.HISREQ initiates a new histogram. The counting operation ends when ISC_INTSR.HISDONE is set. At that time, a software or hardware dma transfer copies the table from the interface to the internal or external memory. To clear the table content (for a new operation), use the ISC_CTRLEN.HISCLR field. An automatic clear (reset after read) is available when bit ISC_HIS_CFG.RAR is set. In that case, as soon as the data is read from the table, the table entry is cleared.

**Figure 49-35. Histogram Block Diagram**

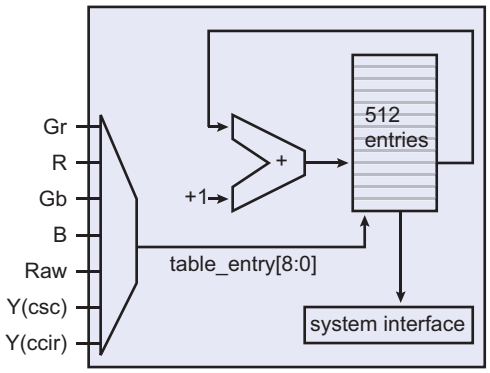
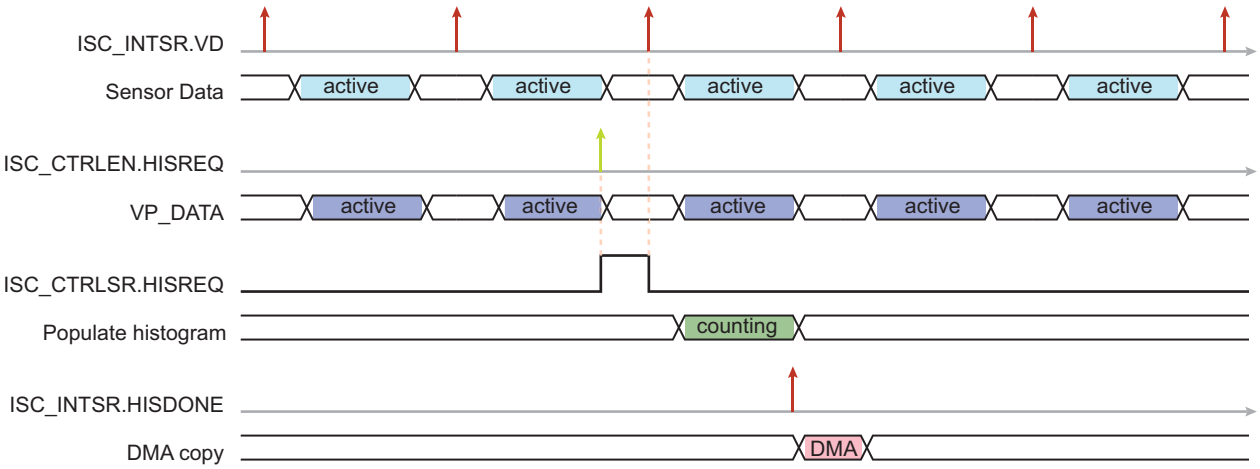



**Figure 49-36. Histogram Request timing diagram**



## 49.6 Image Sensor Controller (ISC) User Interface

**Table 49-18. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x00 | Control Enable Register | ISC_CTRLEN | Write-only | – |
| 0x04 | Control Disable Register | ISC_CTRLDIS | Write-only | – |
| 0x08 | Control Status Register | ISC_CTRLSR | Read-only | 0x00000000 |
| 0x0C | Parallel Front End Configuration 0 Register | ISC_PFE_CFG0 | Read/Write | 0x00000000 |
| 0x10 | Parallel Front End Configuration 1 Register | ISC_PFE_CFG1 | Read/Write | 0x00000000 |
| 0x14 | Parallel Front End Configuration 2 Register | ISC_PFE_CFG2 | Read/Write | 0x00000000 |
| 0x18 | Clock Enable Register | ISC_CLKEN | Write-only | – |
| 0x1C | Clock Disable Register | ISC_CLKDIS | Write-only | – |
| 0x20 | Clock Status Register | ISC_CLKSR | Read-only | 0x00000000 |
| 0x24 | Clock Configuration Register | ISC_CLKCFG | Read/Write | 0x00000000 |
| 0x28 | Interrupt Enable Register | ISC_INTEN | Write-only | – |
| 0x2C | Interrupt Disable Register | ISC_INTDIS | Write-only | – |
| 0x30 | Interrupt Mask Register | ISC_INTMASK | Read-only | 0x00000000 |
| 0x34 | Interrupt Status Register | ISC_INTSR | Read-only | 0x00000000 |
| 0x38–0x3C | Reserved | – | – | 0x00000000 |
| 0x40–0x54 | Reserved | – | – | 0x00000000 |
| 0x58 | White Balance Control Register | ISC_WB_CTRL | Read/Write | 0x00000000 |
| 0x5C | White Balance Configuration Register | ISC_WB_CFG | Read/Write | 0x00000000 |
| 0x60 | White Balance Offset for R, GR Register | ISC_WB_O_RGR | Read/Write | 0x00000000 |
| 0x64 | White Balance Offset for B, GB Register | ISC_WB_O_BGB | Read/Write | 0x00000000 |
| 0x68 | White Balance Gain for R, GR Register | ISC_WB_G_RGR | Read/Write | 0x00000000 |
| 0x6C | White Balance Gain for B, GB Register | ISC_WB_G_BGB | Read/Write | 0x00000000 |
| 0x70 | Color Filter Array Control Register | ISC_CFA_CTRL | Read/Write | 0x00000000 |
| 0x74 | Color Filter Array Configuration Register | ISC_CFA_CFG | Read/Write | 0x00000000 |
| 0x78 | Color Correction Control Register | ISC_CC_CTRL | Read/Write | 0x00000000 |
| 0x7C | Color Correction RR RG Register | ISC_CC_RR_RG | Read/Write | 0x00000000 |
| 0x80 | Color Correction RB OR Register | ISC_CC_RB_OR | Read/Write | 0x00000000 |
| 0x84 | Color Correction GR GG Register | ISC_CC_GR_GG | Read/Write | 0x00000000 |
| 0x88 | Color Correction GB OG Register | ISC_CC_GB_OG | Read/Write | 0x00000000 |
| 0x8C | Color Correction BR BG Register | ISC_CC_BR_BG | Read/Write | 0x00000000 |
| 0x90 | Color Correction BB OB Register | ISC_CC_BB_OB | Read/Write | 0x00000000 |
| 0x94 | Gamma Correction Control Register | ISC_GAM_CTRL | Read/Write | 0x00000000 |
| 0x98 | Gamma Correction Blue Entry 0 | ISC_GAM_BENTRY0 | Read/Write | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0x194 | Gamma Correction Blue Entry 63 | ISC_GAM_BENTRY63 | Read/Write | 0x00000000 |

**Table 49-18. Register Mapping (Continued)**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x198 | Gamma Correction Green Entry 0 | ISC_GAM_GENTRY0 | Read/Write | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0x294 | Gamma Correction Green Entry 63 | ISC_GAM_GENTRY63 | Read/Write | 0x00000000 |
| 0x298 | Gamma Correction Red Entry 0 | ISC_GAM_RENTRY0 | Read/Write | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0x394 | Gamma Correction Red Entry 63 | ISC_GAM_RENTRY63 | Read/Write | 0x00000000 |
| 0x398 | Color Space Conversion Control Register | ISC_CSC_CTRL | Read/Write | 0x00000000 |
| 0x39C | Color Space Conversion YR, YG Register | ISC_CSC_YR_YG | Read/Write | 0x00000000 |
| 0x3A0 | Color Space Conversion YB, OY Register | ISC_CSC_YB_OY | Read/Write | 0x00000000 |
| 0x3A4 | Color Space Conversion CBR CBG Register | ISC_CSC_CBR_CBG | Read/Write | 0x00000000 |
| 0x3A8 | Color Space Conversion CBB OCB Register | ISC_CSC_CBB_OCB | Read/Write | 0x00000000 |
| 0x3AC | Color Space Conversion CRR CRG Register | ISC_CSC_CRR_CRG | Read/Write | 0x00000000 |
| 0x3B0 | Color Space Conversion CRB OCR Register | ISC_CSC_CRB_OCR | Read/Write | 0x00000000 |
| 0x3B4 | Contrast and Brightness Control Register | ISC_CBC_CTRL | Read/Write | 0x00000000 |
| 0x3B8 | Contrast and Brightness Configuration Register | ISC_CBC_CFG | Read/Write | 0x00000000 |
| 0x3BC | Contrast and Brightness, Brightness Register | ISC_CBC_BRIGHT | Read/Write | 0x00000000 |
| 0x3C0 | Contrast and Brightness, Contrast Register | ISC_CBC_CONTRAST | Read/Write | 0x00000000 |
| 0x3C4 | Subsampling 4:4:4 to 4:2:2 Control Register | ISC_SUB422_CTRL | Read/Write | 0x00000000 |
| 0x3C8 | Subsampling 4:4:4 to 4:2:2 Configuration Register | ISC_SUB422_CFG | Read/Write | 0x00000000 |
| 0x3CC | Subsampling 4:2:2 to 4:2:0 Control Register | ISC_SUB420_CTRL | Read/Write | 0x00000000 |
| 0x3D0 | Rounding, Limiting and Packing Configuration Register | ISC_RLP_CFG | Read/Write | 0x00000000 |
| 0x3D4 | Histogram Control Register | ISC_HIS_CTRL | Read/Write | 0x00000000 |
| 0x3D8 | Histogram Configuration Register | ISC_HIS_CFG | Read/Write | 0x00000000 |
| 0x3DC | Reserved | – | – | – |
| 0x3E0 | DMA Configuration Register | ISC_DCFG | Read/Write | 0x00000000 |
| 0x3E4 | DMA Control Register | ISC_DCTRL | Read/Write | 0x00000000 |
| 0x3E8 | DMA Descriptor Address Register | ISC_DNDA | Read/Write | 0x00000000 |
| 0x3EC | DMA Address 0 Register | ISC_DAD0 | Read/Write | 0x00000000 |
| 0x3F0 | DMA Stride 0 Register | ISC_DST0 | Read/Write | 0x00000000 |
| 0x3F4 | DMA Address 1 Register | ISC_DAD1 | Read/Write | 0x00000000 |
| 0x3F8 | DMA Stride 1 Register | ISC_DST1 | Read/Write | 0x00000000 |
| 0x3FC | DMA Address 2 Register | ISC_DAD2 | Read/Write | 0x00000000 |
| 0x400 | DMA Stride 2 Register | ISC_DST2 | Read/Write | 0x00000000 |
| 0x404–0x40C | Reserved | – | – | |

**Table 49-18. Register Mapping (Continued)**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x410 | Histogram Entry 0 | ISC_HIS_ENTRY0 | Read-only | 0x00000000 |
| ... | ... | ... | ... | ... |
| 0xBFC | Histogram Entry 511 | ISC_HIS_ENTRY511 | Read-only | 0x00000000 |

### 49.6.1 ISC Control Enable Register 0

**Name:** ISC_CTRLEN

**Address:** 0xF0008000

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | HISCLR | HISREQ | UPPRO | CAPTURE |

- **CAPTURE: Capture Input Stream Command**

0: Writing a zero to this bit has no effect.

1: Write one to start a single shot capture or a multiple frame.

- **UPPRO: Update Profile**

0: Writing a zero to this bit has no effect.

1: Write one to update the color profile.

- **HISREQ: Histogram Request**

0: Writing a zero to this bit has no effect.

1: Write one to update the histogram table.

- **HISCLR: Histogram Clear**

0: Writing a zero to this bit has no effect.

1: Write one to clear the histogram table.

### 49.6.2 ISC Control Disable Register 0

**Name:** ISC_CTRLDIS

**Address:** 0xF0008004

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | SWRST |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | DISABLE |

- **DISABLE: Capture Disable**

0: Writing a zero to this bit has no effect.

1: Write one to end the capture at the next Vertical Synchronization Detection.

- **SWRST: Software Reset**

0: Writing a zero to this bit has not effect.

1: Write one to perform a software reset of the interface.

### 49.6.3 ISC Control Status Register 0

**Name:** ISC_CTRLSR

**Address:** 0xF0008008

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SIP | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | FIELD | – | HISREQ | UPPRO | CAPTURE |

- **CAPTURE: Capture pending**

0: Capture mode is disabled.

1: Capture is pending.

- **UPPRO: Profile Update Pending**

0: There is no profile update pending request.

1: Indicates that the profile update request is still pending.

- **HISREQ: Histogram Request Pending**

0: There is no histogram pending request.

1: Indicates that the histogram request is still pending.

- **FIELD: Field Status (only relevant when the video stream is interlaced)**

0: The current field/segment is a top field

1: The current field/segment is a bottom field.

- **SIP: Synchronization In Progress**

0: The double domain synchronization is terminated.

1: The double domain synchronization is in progress.

### 49.6.4 ISC Parallel Front End Configuration 0 Register

**Name:** ISC_PFE_CFG0

**Address:** 0xF000800C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| REP | BPS | | | CCIR_REP | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| SKIPCNT | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | ROWEN | COLEN | CCIR10_8N | CCIR_CRC | CCIR656 | GATED |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CONT | MODE | | | FPOL | PPOL | VPOL | HPOL |

- **HPOL: Horizontal Synchronization Polarity**

0: HSYNC signal is active high, i.e. valid pixels are sampled when HSYNC is asserted.

1: HSYNC signal is active low, i.e. valid pixels are sampled when HSYNC is deasserted.

- **VPOL: Vertical Synchronization Polarity**

0: VSYNC signal is active high, i.e. valid pixels are sampled when VSYNC is asserted.

1: VSYNC signal is active low, i.e. valid pixels are sampled when VSYNC is deasserted.

- **PPOL: Pixel Clock Polarity**

0: The pixel stream is sampled on the rising edge of the pixel clock.

1: The pixel stream is sampled on the falling edge of the pixel clock.

- **FPOL: Field Polarity**

0: Top field is sampled when F value is 0; Bottom field is sampled when F value is 1

1: Top field is sampled when F value is 1; Bottom field is sampled when F value is 0

- **MODE: Parallel Front End Mode**

| Value | Name | Description |
|---|---|---|
| 0 | PROGRESSIVE | Video source is progressive. |
| 1 | DF_TOP | Video source is interlaced, two fields are captured starting with top field. |
| 2 | DF_BOTTOM | Video source is interlaced, two fields are captured starting with bottom field. |
| 3 | DF_IMMEDIATE | Video source is interlaced, two fields are captured immediately. |
| 4 | SF_TOP | Video source is interlaced, one field is captured starting with the top field. |
| 5 | SF_BOTTOM | Video source is interlaced, one field is captured starting with the bottom field. |
| 6 | SF_IMMEDIATE | Video source is interlaced, one field is captured starting immediately. |

- **CONT: Continuous Acquisition**

0: Single Shot mode

1: Video mode

- **GATED: Gated input clock**

0: The external pixel clock is free running.

1: The external pixel clock is gated.

- **CCIR656: CCIR656 input mode**

0: HSYNC and VSYNC signals are used to synchronize the input stream.

1: Embedded synchronization is used.

- **CCIR_CRC: CCIR656 CRC Decoder**

0: Embedded CRC is discarded.

1: Embedded CRC is decoded.

- **CCIR10_8N: CCIR 10 bits or 8 bits**

0: 8-bit mode

1: 10-bit mode

- **COLEN: Column Cropping Enable**

0: Column Cropping is disabled.

1: Column Cropping is enabled.

- **ROWEN: Row Cropping Enable**

0: Row Cropping is disabled

1: Row Cropping is enabled.

- **SKIPCNT: Frame Skipping Counter**

- **CCIR_REP: CCIR Replication**

0: Unused bits are stuck at 0.

1: Unused bits are copied from MSB.

- **BPS: Bits Per Sample**

| Value | Name | Description |
|---|---|---|
| 0 | TWELVE | 12-bit input |
| 1 | ELEVEN | 11-bit input |
| 2 | TEN | 10-bit input |
| 3 | NINE | 9-bit input |
| 4 | EIGHT | 8-bit input |

- **REP: Up Multiply with Replication**

0: Unused bits are stuck at 0.

1: Unused bits are copied from MSB.

### 49.6.5 ISC Parallel Front End Configuration 1 Register

**Name:** ISC_PFE_CFG1

**Address:** 0xF0008010

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| COLMAX | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| COLMAX | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| COLMIN | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| COLMIN | | | | | | | |

- **COLMIN: Column Minimum Limit**

Horizontal starting position of the cropping area

- **COLMAX: Column Maximum Limit**

Horizontal ending position of the cropping area

### 49.6.6 ISC Parallel Front End Configuration 2 Register

**Name:** ISC_PFE_CFG2

**Address:** 0xF0008014

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| ROWMAX | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ROWMAX | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ROWMIN | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ROWMIN | | | | | | | |

- **ROWMIN: Row Minimum Limit**

Vertical starting position of the cropping area

- **ROWMAX: Row Maximum Limit**

Vertical ending position of the cropping area

### 49.6.7 ISC Clock Enable Register

**Name:** ISC_CLKEN

**Address:** 0xF0008018

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | MCEN | ICEN |

- **ICEN: ISP Clock Enable**

0: No effect.

1: Enables the ISP clock.

- **MCEN: Master Clock Enable**

0: No effect.

1: Enables the master clock.

### 49.6.8 ISC Clock Disable Register

**Name:** ISC_CLKDIS

**Address:** 0xF000801C

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | MCSWRST | ICSWRST |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | MCDIS | ICDIS |

- **ICDIS: ISP Clock Disable**

0: No effect.

1: Disables the ISP clock.

- **MCDIS: Master Clock Disable**

0: No effect.

1: Disables the master clock.

- **ICSWRST: ISP Clock Software Reset**

0: No effect.

1: Software reset the ISP clock.

- **MCSWRST: Master Clock Software Reset**

0: No effect.

1: Software reset the master clock.

### 49.6.9 ISC Clock Status Register

**Name:** ISC_CLKSR

**Address:** 0xF0008020

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| SIP | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | MCSR | ICSR |

- **ICSR: ISP Clock Status Register**

0: The ISP clock is disabled.

1: The ISP clock is enabled.

- **MCSR: Master Clock Status Register**

0: The master clock is disabled.

1: The master clock is enabled.

- **SIP: Synchronization In Progress**

0: The double domain synchronization operation is over.

1: The double domain synchronization operation is in progress.

### 49.6.10 ISC Clock Configuration Register

**Name:** ISC_CLKCFG

**Address:** 0xF0008024

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | MCSEL | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| MCDIV | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | ICSEL |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ICDIV | | | | | | | |

- **ICDIV: ISP Clock Divider**

$$f_{cc} = \frac{f_{ccref}}{ICDIV + 1}$$

- **ICSEL: ISP Clock Selection**

0: HCLOCK is selected.

1: ISCCLK is selected.

- **MCDIV: Master Clock Divider**

$$f_{mc} = \frac{f_{mcref}}{MCDIV + 1}$$

- **MCSEL: Master Clock Reference Clock Selection**

0: HCLOCK is selected.

1: ISCCLK is selected.

2: GCK is selected.

### 49.6.11 ISC Interrupt Enable Register

**Name:** ISC_INTEN

**Address:** 0xF0008028

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | CCIRERR | HDTO | VDTO | DAOV | VFPOV |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | RERR | – | – | – | WERR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | HISCLR | HISDONE | – | – | LDONE | DDONE |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | DIS | SWRST | – | – | HD | VD |

- **VD: Vertical Synchronization Detection Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **HD: Horizontal Synchronization Detection Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **SWRST: Software Reset Completed Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **DIS: Disable Completed Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **DDONE: DMA Done Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **LDONE: DMA List Done Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **HISDONE: Histogram Completed Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **HISCLR: Histogram Clear Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **WERR: Write Channel Error Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **RERR: Read Channel Error Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **VFPOV: Vertical Front Porch Overflow Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **DAOV: Data Overflow Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **VDTO: Vertical Synchronization Timeout Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **HDTO: Horizontal Synchronization Timeout Interrupt Enable**

0: No effect

1: The interrupt is enabled.

- **CCIRERR: CCIR Decoder Error Interrupt Enable**

0: No effect

1: The interrupt is enabled.

### 49.6.12 ISC Interrupt Disable Register

**Name:** ISC_INTDIS

**Address:** 0xF000802C

**Access:** Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | CCIRERR | HDTO | VDTO | DAOV | VFPOV |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | RERR | – | – | – | WERR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | HISCLR | HISDONE | – | – | LDONE | DDONE |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | DIS | SWRST | – | – | HD | VD |

- **VD: Vertical Synchronization Detection Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **HD: Horizontal Synchronization Detection Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **SWRST: Software Reset Completed Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **DIS: Disable Completed Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **DDONE: DMA Done Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **LDONE: DMA List Done Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **HISDONE: Histogram Completed Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **HISCLR: Histogram Clear Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **WERR: Write Channel Error Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **RERR: Read Channel Error Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **VFPOV: Vertical Front Porch Overflow Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **DAOV: Data Overflow Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **VDTO: Vertical Synchronization Timeout Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **HDTO: Horizontal Synchronization Timeout Interrupt Disable**

0: No effect

1: The interrupt is disabled.

- **CCIRERR: CCIR Decoder Error Interrupt Disable**

0: No effect

1: The interrupt is disabled.

### 49.6.13 ISC Interrupt Mask Register

**Name:** ISC_INTMASK

**Address:** 0xF0008030

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | CCIRERR | HDTO | VDTO | DAOV | VFPOV |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | RERR | – | – | – | WERR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | HISCLR | HISDONE | – | – | LDONE | DDONE |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | DIS | SWRST | – | – | HD | VD |

- **VD: Vertical Synchronization Detection Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **HD: Horizontal Synchronization Detection Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **SWRST: Software Reset Completed Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **DIS: Disable Completed Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **DDONE: DMA Done Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **LDONE: DMA List Done Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **HISDONE: Histogram Completed Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **HISCLR: Histogram Clear Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **WERR: Write Channel Error Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **RERR: Read Channel Error Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **VFPOV: Vertical Front Porch Overflow Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **DAOV: Data Overflow Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **VDTO: Vertical Synchronization Timeout Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **HDTO: Horizontal Synchronization Timeout Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

- **CCIRERR: CCIR Decoder Error Interrupt Mask**

0: The interrupt is disabled.

1: The interrupt is enabled.

### 49.6.14 ISC Interrupt Status Register

**Name:** ISC_INTSR

**Address:** 0xF0008034

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | CCIRERR | HDTO | VDTO | DAOV | VFPOV |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | RERR | – | WERRID | | WERR |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | HISCLR | HISDONE | – | – | LDONE | DDONE |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | DIS | SWRST | – | – | HD | VD |

- **VD: Vertical Synchronization Detected Interrupt**

0: No vertical synchronization detection since the last read of the Interrupt Status register

1: A vertical synchronization has been detected.

- **HD: Horizontal Synchronization Detected Interrupt**

0: No horizontal synchronization detection since the last read of the Interrupt Status register

1: A horizontal synchronization has been detected.

- **SWRST: Software Reset Completed Interrupt**

0: No software reset completion since the last read of the Interrupt Status register

1: The software reset has completed.

- **DIS: Disable Completed Interrupt**

0: The disable has not occurred since the last read of the Interrupt Status register.

1: The disable has completed.

- **DDONE: DMA Done Interrupt**

0: No DMA Transfer Done Interrupt has occurred since the last read of the Interrupt Status register.

1: The DMA Transfer Done Interrupt has occurred.

- **LDONE: DMA List Done Interrupt**

0: No DMA List Done Interrupt has occurred since the last read of the Interrupt Status register.

1: The DMA List Done Interrupt has occurred.

- **HISDONE: Histogram Completed Interrupt**

0: No Histogram Completed Interrupt has been raised since the last read of the Interrupt Status register.

1: The Histogram Completed Interrupt has occurred.

- **HISCLR: Histogram Clear Interrupt**

0: No Histogram Clear Interrupt has been raised since the last read of the Interrupt Status register.

1: The Histogram Clear Interrupt has occurred.

- **WERR: Write Channel Error Interrupt**

0: No write channel error since the last read of the Interrupt Status register

1: A write channel error occurred.

- **WERRID: Write Channel Error Identifier**

| Value | Name | Description |
|---|---|---|
| 0 | CH0 | An error occurred for Channel 0 (RAW/RGB/Y) |
| 1 | CH1 | An error occurred for Channel 1 (CbCr/Cb) |
| 2 | CH2 | An error occurred for Channel 2 (Cr) |
| 3 | WB | Write back channel error |

- **RERR: Read Channel Error Interrupt**

0: No read channel error since the last read of the Interrupt Status register

1: A read channel error occurred when the ISC read the descriptor.

- **VFPOV: Vertical Front Porch Overflow Interrupt**

0: No vertical front porch error occurred since the last read of the Interrupt Status register.

1: The vertical synchronization has been detected but the DMA channel is still busy.

- **DAOV: Data Overflow Interrupt**

0: No data overflow error occurred since the last reset of the Interrupt Status register.

1: A data overflow occurred.

- **VDTO: Vertical Synchronization Timeout Interrupt**

0: A vertical synchronization is detected.

1: No vertical synchronization is detected.

- **HDTO: Horizontal Synchronization Timeout Interrupt**

0: A horizontal synchronization is detected.

1: No horizontal synchronization is detected.

- **CCIRERR: CCIR Decoder Error Interrupt**

0: No CCIR CRC error detected since the last read of the Interrupt Status register

1: A CCIR CRC error has been detected.

### 49.6.15 ISC White Balance Control Register

**Name:** ISC_WB_CTRL

**Address:** 0xF0008058

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | ENABLE |

- **ENABLE: White Balance Enable**

0: The white balance is disabled.

1: The white balance is enabled.

### 49.6.16 ISC White Balance Configuration Register

**Name:** ISC_WB_CFG

**Address:** 0xF000805C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | BAYCFG | |

- **BAYCFG: White Balance Bayer Configuration (Pixel Color Pattern)**

| Value | Name | Description |
|---|---|---|
| 0 | GRGR | Starting Row configuration is G R G R (Red Row) |
| 1 | RGRG | Starting Row configuration is R G R G (Red Row) |
| 2 | GBGB | Starting Row configuration is G B G B (Blue Row) |
| 3 | BGBG | Starting Row configuration is B G B G (Blue Row) |

### 49.6.17 ISC White Balance Offset for R, GR Register

**Name:** ISC_WB_O_RGR

**Address:** 0xF0008060

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | GROFST | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GROFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | ROFST | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ROFST | | | | | | | |

- **ROFST: Offset Red Component (signed 13 bits 1:12:0)**
- **GROFST: Offset Green Component for Red Row (signed 13 bits 1:12:0)**

### 49.6.18 ISC White Balance Offset for B and GB Register

**Name:** ISC_WB_O_BGB

**Address:** 0xF0008064

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | GBOFST | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GBOFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOFST | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BOFST | | | | | | | |

- **BOFST: Offset Blue Component (signed 13 bits, 1:12:0)**
- **GBOFST: Offset Green Component for Blue Row (signed 13 bits, 1:12:0)**

### 49.6.19 ISC White Balance Gain for R, GR Register

**Name:** ISC_WB_G_RGR

**Address:** 0xF0008068

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | GRGAIN | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| GRGAIN | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | RGAIN | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RGAIN | | | | | | | |

- **RGAIN: Red Component Gain (unsigned 13 bits, 0:4:9)**
- **GRGAIN: Green Component (Red row) Gain (unsigned 13 bits, 0:4:9)**

### 49.6.20 ISC White Balance Gain for B, GB Register

**Name:** ISC_WB_G_BGB

**Address:** 0xF000806C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | GBGAIN | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GBGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | BGAIN | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BGAIN | | | | | | | |

- **BGAIN: Blue Component Gain (unsigned 13 bits, 0:4:9)**
- **GBGAIN: Green Component (Blue row) Gain (unsigned 13 bits, 0:4:9)**

### 49.6.21 ISC Color Filter Array Control Register

**Name:** ISC_CFA_CTRL

**Address:** 0xF0008070

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | ENABLE |

- **ENABLE: Color Filter Array Interpolation Enable**

0: Color Filter Array Interpolation is disabled.

1: Color Filter Array Interpolation is enabled.

### 49.6.22 ISC Color Filter Array Configuration Register

**Name:** ISC_CFA_CFG

**Address:** 0xF0008074

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | EITPOL | – | – | BAYCFG | |

- **BAYCFG: Color Filter Array Pattern**

| Value | Name | Description |
|---|---|---|
| 0 | GRGR | Starting row configuration is G R G R (red row) |
| 1 | RGRG | Starting row configuration is R G R G (red row |
| 2 | GBGB | Starting row configuration is G B G B (blue row) |
| 3 | BGBG | Starting row configuration is B G B G (blue row) |

- **EITPOL: Edge Interpolation**

0: Edges are not interpolated.

1: Edge interpolation is performed.

### 49.6.23 ISC Color Correction Control Register

**Name:** ISC_CC_CTRL

**Address:** 0xF0008078

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | ENABLE |

- **ENABLE: Color Correction Enable**

0: Color correction is disabled.

1: Color correction is enabled.

### 49.6.24 ISC Color Correction RR RG Register

**Name:** ISC_CC_RR_RG

**Address:** 0xF000807C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | RGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | RRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RRGAIN | | | | | | | |

- **RRGAIN: Red Gain for Red Component (signed 12 bits, 1:3:8)**
- **RGGAIN: Green Gain for Red Component (signed 12 bits, 1:3:8)**

### 49.6.25 ISC Color Correction RB OR Register

**Name:** ISC_CC_RB_OR

**Address:** 0xF0008080

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | ROFST | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ROFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | RBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RBGAIN | | | | | | | |

- **RBGAIN: Blue Gain for Red Component (signed 12 bits, 1:3:8)**
- **ROFST: Red Component Offset (signed 13 bits, 1:12:0)**

### 49.6.26 ISC Color Correction GR GG Register

**Name:** ISC_CC_GR_GG

**Address:** 0xF0008084

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | GGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | GRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GRGAIN | | | | | | | |

- **GRGAIN: Red Gain for Green Component (signed 12 bits, 1:3:8)**
- **GGGAIN: Green Gain for Green Component (signed 12 bits, 1:3:8)**

### 49.6.27 ISC Color Correction GB OG Register

**Name:** ISC_CC_GB_OG

**Address:** 0xF0008088

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | ROFST | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| ROFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | GBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GBGAIN | | | | | | | |

- **GBGAIN: Blue Gain for Green Component (signed 12 bits, 1:3:8)**
- **ROFST: Green Component Offset (signed 13 bits, 1:12:0)**

### 49.6.28 ISC Color Correction BR BG Register

**Name:** ISC_CC_BR_BG

**Address:** 0xF000808C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | BGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| BGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | BRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BRGAIN | | | | | | | |

- **BRGAIN: Red Gain for Blue Component (signed 12 bits, 1:3:8)**
- **BGGAIN: Green Gain for Blue Component (signed 12 bits, 1:3:8)**

### 49.6.29 ISC Color Correction BB OB Register

**Name:** ISC_CC_BB_OB

**Address:** 0xF0008090

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | BOFST | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| BOFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | BBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BBGAIN | | | | | | | |

- **BBGAIN: Blue Gain for Blue Component (signed 12 bits, 1:3:8)**
- **BOFST: Blue Component Offset (signed 13 bits, 1:12:0)**

### 49.6.30 ISC Gamma Correction Control Register

**Name:** ISC_GAM_CTRL

**Address:** 0xF0008094

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | | | RENABLE | GENABLE | BENABLE | ENABLE |

- **ENABLE: Gamma Correction Enable**

0: Gamma correction is disabled.

1: Gamma correction is enabled.

- **BENABLE: Gamma Correction Enable for B Channel**

0: 12 bits to 10 bits compression is performed skipping two bits.

1: Piecewise interpolation is used to perform 12 bits to 10 bits compression for the blue channel.

- **GENABLE: Gamma Correction Enable for G Channel**

0: 12 bits to 10 bits compression is performed skipping two bits.

1: Piecewise interpolation is used to perform 12 bits to 10 bits compression for the green channel.

- **RENABLE: Gamma Correction Enable for R Channel**

0: 12 bits to 10 bits compression is performed skipping two bits.

1: Piecewise interpolation is used to perform 12 bits to 10 bits compression for the red channel.

### 49.6.31 ISC Gamma Correction Blue Entry Register

**Name:** ISC_GAM_BENTRYx[x=0...63]

**Address:** 0xF0008098

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | BCONSTANT | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| BCONSTANT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | BSLOPE | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BSLOPE | | | | | | | |

- **BSLOPE: Blue Color Slope for Piecewise Interpolation (signed 10 bits 1:3:6)**
- **BCONSTANT: Blue Color Constant for Piecewise Interpolation (unsigned 10 bits 0:10:0)**

### 49.6.32 ISC Gamma Correction Green Entry Register

**Name:** ISC_GAM_GENTRYx[x=0..63]

**Address:** 0xF0008198

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | GCONSTANT | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| GCONSTANT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | GSLOPE | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GSLOPE | | | | | | | |

- **GSLOPE: Green Color Slope for Piecewise Interpolation (signed 10 bits 1:3:6)**
- **GCONSTANT: Green Color Constant for Piecewise Interpolation (unsigned 10 bits 0:10:0)**

### 49.6.33 ISC Gamma Correction Red Entry Register

**Name:** ISC_GAM_RENTRYx[x=0..63]

**Address:** 0xF0008298

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | RCONSTANT | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| RCONSTANT | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | RSLOPE | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RSLOPE | | | | | | | |

- **RSLOPE: Red Color Slope for Piecewise Interpolation (signed 10 bits 1:3:6)**
- **RCONSTANT: Red Color Constant for Piecewise Interpolation (unsigned 10 bits 0:10:0)**

### 49.6.34 Color Space Conversion Control Register

**Name:** ISC_CSC_CTRL

**Address:** 0xF0008398

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| – | – | – | – | – | – | – | – |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| – | – | – | – | – | – | – | – |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| – | – | – | – | – | – | – | ENABLE |

- **ENABLE: RGB to YCbCr Color Space Conversion Enable**

0: Color space conversion is disabled.

1: Color space conversion is enabled.

### 49.6.35 Color Space Conversion YR YG Register

**Name:** ISC_CSC_YR_YG

**Address:** 0xF000839C

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | YGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| YGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | YRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| YRGAIN | | | | | | | |

- **YRGAIN: Reg Gain for Luminance (signed 12 bits 1:3:8)**
- **YGGAIN: Green Gain for Luminance (signed 12 bits 1:3:8)**

### 49.6.36 Color Space Conversion YB OY Register

**Name:** ISC_CSC_YB_OY

**Address:** 0xF00083A0

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | YOFST | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| YOFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | YBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| YBGAIN | | | | | | | |

- **YBGAIN: Blue Gain for Luminance Component (12 bits signed 1:3:8)**
- **YOFST: Luminance Offset (11 bits signed 1:10:0)**

### 49.6.37 Color Space Conversion CBR CBG Register

**Name:** ISC_CSC_CBR_CBG

**Address:** 0xF00083A4

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CBGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CBGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CBRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CBRGAIN | | | | | | | |

- **CBRGAIN: Red Gain for Blue Chrominance (signed 12 bits, 1:3:8)**
- **CBGGAIN: Green Gain for Blue Chrominance (signed 12 bits 1:3:8)**

### 49.6.38 Color Space Conversion CBB OCB Register

**Name:** ISC_CSC_CBB_OCB

**Address:** 0xF00083A8

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | CBOFST | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CBOFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CBBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CBBGAIN | | | | | | | |

- **CBBGAIN: Blue Gain for Blue Chrominance (signed 12 bits 1:3:8)**
- **CBOFST: Blue Chrominance Offset (signed 11 bits 1:10:0)**

### 49.6.39 Color Space Conversion CRR CRG Register

**Name:** ISC_CSC_CRR_CRG

**Address:** 0xF00083AC

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CRGGAIN | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CRGGAIN | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CRRGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CRRGAIN | | | | | | | |

- **CRRGAIN: Red Gain for Red Chrominance (signed 12 bits 1:3:8)**
- **CRGGAIN: Green Gain for Red Chrominance (signed 12 bits 1:3:8)**

### 49.6.40 Color Space Conversion CRB OCR Register

**Name:** ISC_CSC_CRB_OCR

**Address:** 0xF00083B0

**Access:** Read/Write

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | CROFST | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CROFST | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | CRBGAIN | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CRBGAIN | | | | | | | |

- **CRBGAIN: Blue Gain for Red Chrominance (signed 12 bits 1:3:8)**
- **CROFST: Red Chrominance Offset (signed 11 bits 1:10:0)**