# I2C

This chapter describes the I2C of the device.

**Topic** **Page**

## 21.1 Introduction

The multi-master I2C peripheral provides an interface between a CPU and any I2C-bus-compatible device that connects via the I2C serial bus. External components attached to the I2C bus can serially transmit/receive up to 8-bit data to/from the CPU device through the two-wire I2C interface.

The I2C bus is a multi-master bus. The I2C controller does support the multi-master mode that allows more than one device capable of controlling the bus to be connected to it. Each I2C device is recognized by a unique address and can operate as either transmitter or receiver, according to the function of the device. In addition to being a transmitter or receiver, a device connected to the I2C bus can also be considered as master or slave when performing data transfers. Note that a master device is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. During this transfer, any device addressed by this master is considered a slave.

### *21.1.1 I2C Features*

The general features of the I2C controller are:

- Compliant with Philips I2C specification version 2.1
- Supports standard mode (up to 100K bits/s) and fast mode (up to 400K bits/s).
- Multimaster transmitter/slave receiver mode
- Multimaster receiver/slave transmitter mode
- Combined master transmit/receive and receive/transmit modes
- 7-bit and 10-bit device addressing modes
- Built-in 32-byte FIFO for buffered read or writes in each module
- Programmable clock generation
- Two DMA channels, one interrupt line

### *21.1.2 Unsupported I2C Features*

The I2C module features not supported in this device are shown in Table 21-1.

**Table 21-1. Unsupported I2C Features**

| Feature | Reason |
|---|---|
| SCCB Protocol | SCCB signal not pinned out |
| High Speed (3.4 MBPS) operation | Not supported |

## 21.2 Integration

This device includes three instantiations of the I2C module. This peripheral implements the multi-master I2C bus which allows serial transfer of 8-bit data to/from other I2C master/slave devices through a two-wire interface. There are three I2C modules instantiations called I2C0, I2C1, and I2C2. The I2C0 module is located in the Wake-up power domain. Figure 21-1 and Figure 21-2 show examples of a system with multiple I2C-compatible devices.

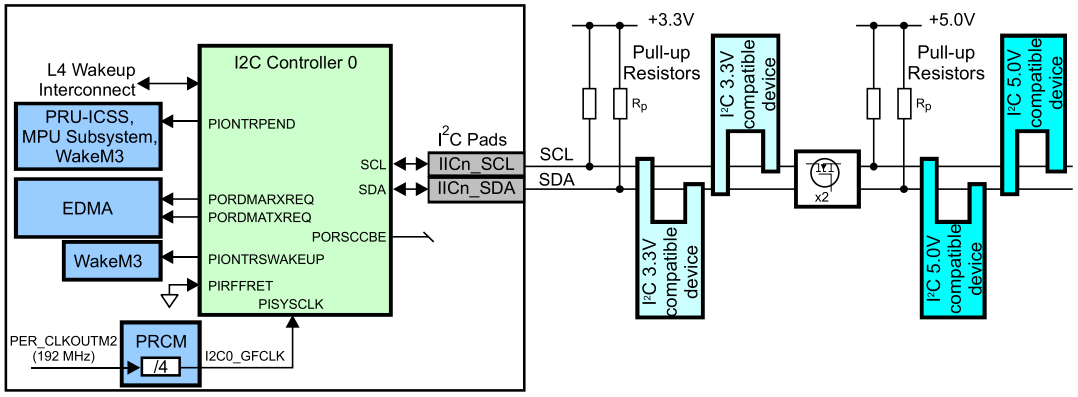



**Figure 21-1. I2C0 Integration and Bus Application**

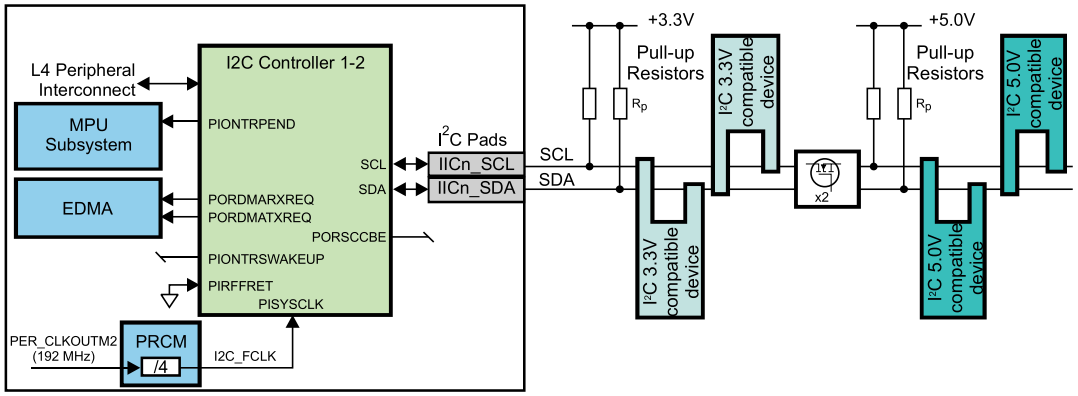



**Figure 21-2. I2C(1–2) Integration and Bus Application**

### 21.2.1 I2C Connectivity Attributes

The general connectivity attributes for the I2C module are shown in Table 21-2 and Table 21-3.

**Table 21-2. I2C0 Connectivity Attributes**

| Attributes | Type |
|---|---|
| Power Domain | Wakeup Domain |
| Clock Domain | PD_WKUP_L4_WKUP_GCLK (Interface/OCP)<br>PD_WKUP_I2C0_GFCLK (Func) |
| Reset Signals | WKUP_DOM_RST_N |
| Idle/Wakeup Signals | Smart Idle / Wakeup |
| Interrupt Requests | 1 interrupt to MPU Subsystem (I2C0INT), PRU-ICSS, and WakeM3 |
| DMA Requests | 2 DMA requests to EDMA (I2CTXEVT0, I2CRXEVT0) |
| Physical Address | L4 Wakeup slave port |

**Table 21-3. I2C(1–2) Connectivity Attributes**

| Attributes | Type |
|---|---|
| Power Domain | Peripheral Domain |
| Clock Domain | PD_PER_L4LS_GCLK (Interface/OCP)<br>PD_PER_I2C_FCLK (Func) |
| Reset Signals | PER_DOM_RST_N |
| Idle/Wakeup Signals | Smart Idle |
| Interrupt Requests | 1 interrupt per instance to MPU Subsystem (I2C1INT, I2C2INT) |
| DMA Requests | 2 DMA requests per instance to EDMA (I2CTXEVTx, I2CRXEVTx) |
| Physical Address | L4 Peripheral slave port |

### 21.2.2 I2C Clock and Reset Management

The I2C controllers have separate bus interface and functional clocks. During power-down mode, the I2C*x*_SCL and I2C*x*_SDA are configured as inputs.

**Table 21-4. I2C Clock Signals**

| Clock Signal | Max Freq | Reference / Source | Comments |
|---|---|---|---|
| **I2C0 Clock Signals** | | | |
| PIOCPCLK<br>Interface clock | MHz | | pd_wkup_l4_wkup_gclk<br>From PRCM |
| PISYSCLK<br>Functional clock | 48 MHz | PER_CLKOUTM2 / 4 | pd_wkup_i2c0_gfclk<br>From PRCM |
| **I2C(1-2) Clock Signals** | | | |
| PIOCPCLK<br>Interface clock | 100 MHz | CORE_CLKOUTM4 / 2 | pd_per_l4ls_gclk<br>From PRCM |
| PISYSCLK<br>Functional clock | 48 MHz | PER_CLKOUTM2 / 4 | pd_per_ic2_fclk<br>From PRCM |

### 21.2.3 I2C Pin List

The external signals (I2C*x*_SDA, I2C*x*_SCL) on the device use standard LVCMOS I/Os and may not meet full compliance with the I2C specifications for Fast-mode devices for slope control and input filtering (spike suppression) to improve the EMC behavior.

**Table 21-5. I2C Pin List**

| Pin | Type | Description |
|---|---|---|
| I2C*x*_SCL | I/OD[(1)] | I2C serial clock (open drain) |
| I2C*x*_SDA | I/OD | I2C serial data (open drain) |

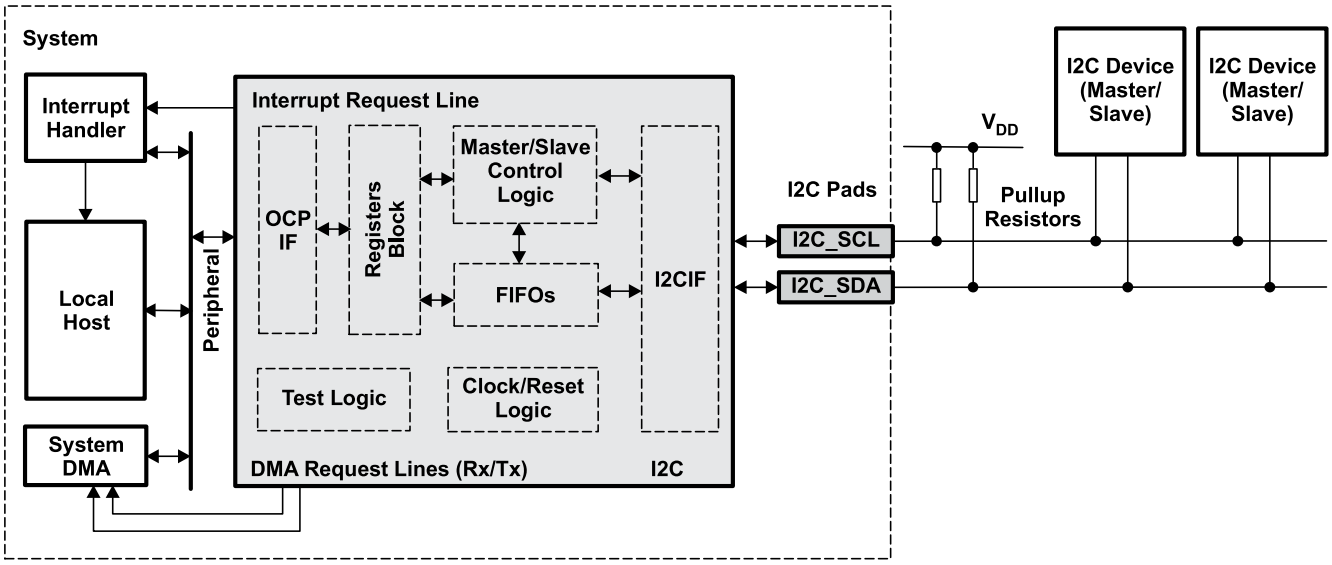
(1) This output signal is also used as a retiming input. The associated CONF_<module>_<pin>_RXACTIVE bit for the output clock must be set to 1 to enable the clock input back to the module.

## 21.3 Functional Description

### 21.3.1 Functional Block Diagram

Figure 21-3 shows an example of a system with multiple I2C compatible devices in which the I2C serial ports are all connected together for a two-way transfer from one device to other devices.

**Figure 21-3. I2C Functional Block Diagram**




The I2C peripheral consists of the following primary blocks:

- A serial interface: one data pin (I2C_SDA) and one clock pin (I2C_SCL).
- Data registers to temporarily hold receive data and transmit data traveling between the I2C_SDA pin and the CPU or the DMA controller.
- Control and status registers
- A peripheral data bus interface to enable the CPU and the DMA controller to access the I2C peripheral registers.
- A clock synchronizer to synchronize the I2C input clock (from the processor clock generator) and the clock on the I2C_SCL pin, and to synchronize data transfers with masters of different clock speeds.
- A prescaler to divide down the input clock that is driven to the I2C peripheral
- A noise filter on each of the two pins, I2C_SDA and I2C_SCL
- An arbitrator to handle arbitration between the I2C peripheral (when it is a master) and another master
- Interrupt generation logic, so that an interrupt can be sent to the CPU
- DMA event generation logic to send an interrupt to the CPU upon reception and data transmission of data.

### 21.3.2 I2C Master/Slave Contoller Signals

Data is communicated to devices interfacing with the I2C via the serial data line (SDA) and the serial clock line (SCL). These two wires can carry information between a device and others connected to the I2C bus. Both SDA and SCL are bi-directional pins. They must be connected to a positive supply voltage via a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open drain to perform the required wired-AND function.

An example of multiple I2C modules that are connected for a two-way transfer from one device to other devices is shown in Figure 21-4.

**Figure 21-4. Multiple I2C Modules Connected**

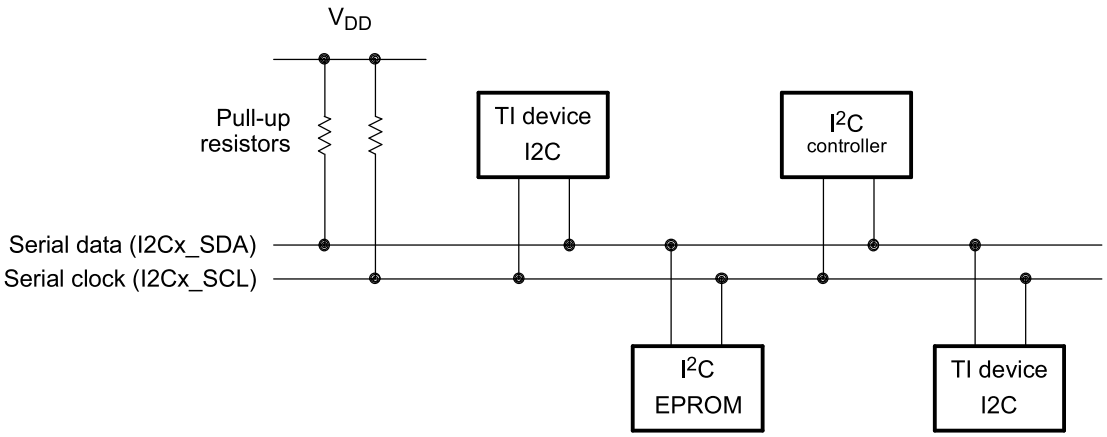



**Table 21-6. Signal Pads**

| Name | I2C Mode | |
|---|---|---|
| | Default Operating Mode | Description |
| I2C_SCL | In/ Out | I2C serial CLK line<br>Open-drain output buffer. Requires external pull-up resistor (Rp). |
| I2C_SDA | In/ Out | I2C serial data line<br>Open-drain output buffer. Requires external pull-up resistor (Rp). |

### 21.3.3 I2C Reset

The I2C module can be reset in the following three ways:

- A system reset (PIRSTNA = 0). A device reset causes the system reset. All registers are reset to power up reset values.
- A software reset by setting the SRST bit in the I2C_SYSC register. This bit has exactly the same action on the module logic as the system bus reset. All registers are reset to power up reset values.
- The I2C_EN bit in the I2C_CON register can be used to hold the I2C module in reset. When the system bus reset is removed (PIRSTNA = 1), I2C_EN = 0 keeps the functional part of I2C module in reset state and all configuration registers can be accessed. I2C_EN = 0 does not reset the registers to power up reset values.

**Table 21-7. Reset State of I2C Signals**

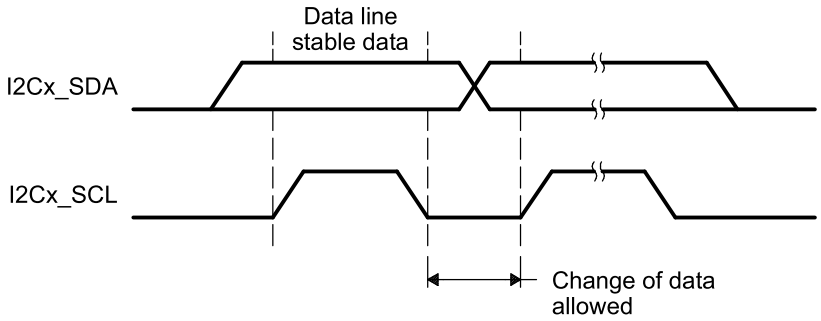
| Pin | I/O/Z [1] | System Reset | I2C Reset (I2C_EN = 0) |
|---|---|---|---|
| SDA | I/O/Z | High impedance | High impedance |
| SCL | I/O/Z | High impedance | High impedance |

[1] I = Input, O = Outpu, Z = High impedance

### 21.3.4 Data Validity

The data on the SDA line must be stable during the high period of the clock. The high and low states of the data line can only change when the clock signal on the SCL line is LOW.

**Figure 21-5. Bit Transfer on the I2C Bus**

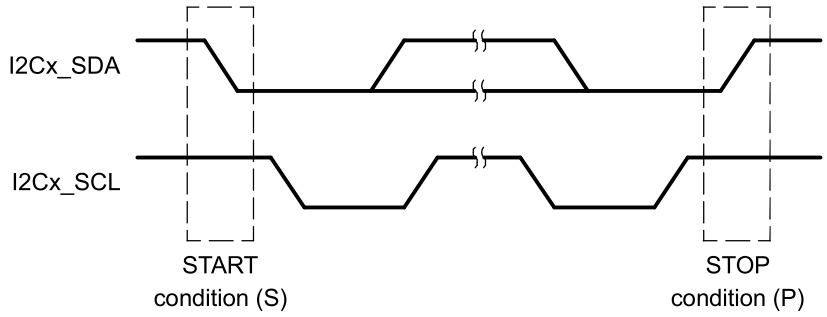


### 21.3.5 START & STOP Conditions

The I2C module generates START and STOP conditions when it is configured as a master.

- START condition is a high-to-low transition on the SDA line while SCL is high.
- STOP condition is a low-to-high transition on the SDA line while SCL is high.
- The bus is considered to be busy after the START condition (BB = 1) and free after the STOP condition (BB = 0).

**Figure 21-6. Start and Stop Condition Events**

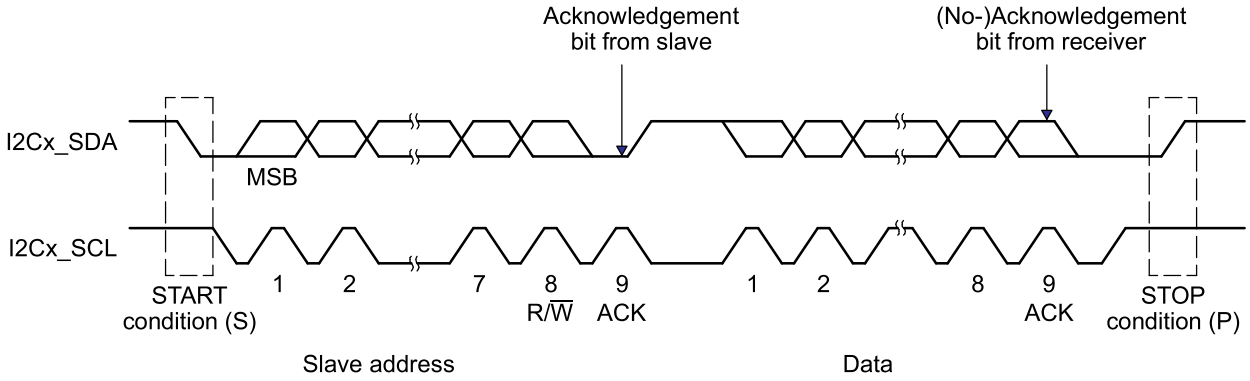



### 21.3.6 I2C Operation

#### 21.3.6.1 Serial Data Formats

The I2C controller operates in 8-bit word data format (byte write access supported for the last access). Each byte put on the SDA line is 8 bits long. The number of bytes that can be transmitted or received is restricted by the value programmed in the DCOUNT register. The data is transferred with the most significant bit (MSB) first. Each byte is followed by an acknowledge bit from the I2C module if it is in receiver mode.

**Figure 21-7. I2C Data Transfer**



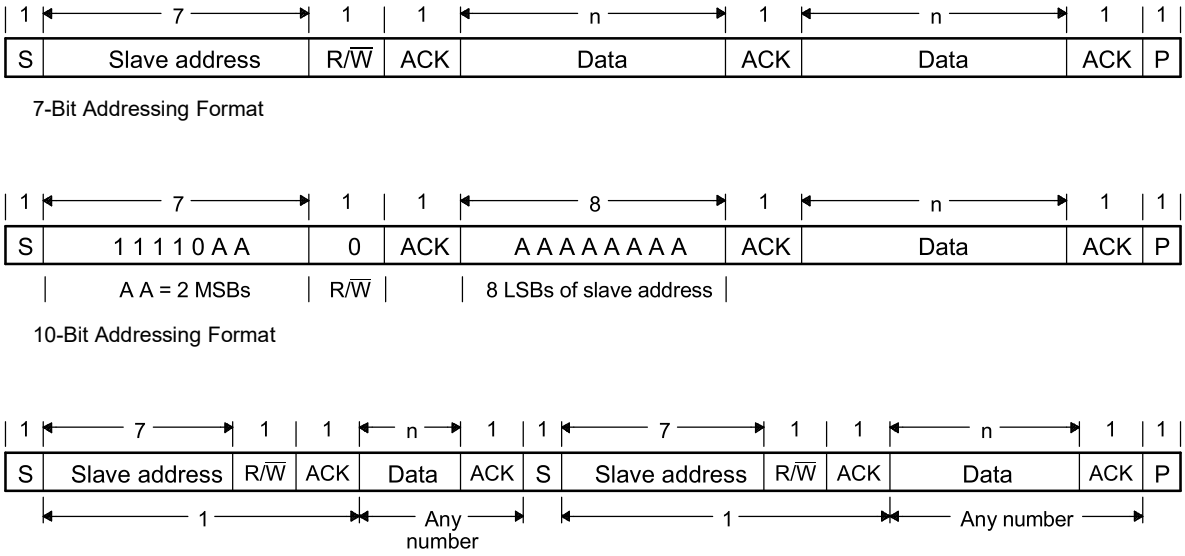

The I2C module supports two data formats, as shown in Figure 21-8:

- 7-bit/10-bit addressing format
- 7-bit/10-bit addressing format with repeated start condition

The first byte after a start condition (S) always consists of 8 bits. In the acknowledge mode, an extra bit dedicated for acknowledgment is inserted after each byte. In the addressing formats with 7-bit addresses, the first byte is composed of 7 MSB slave address bits and 1 LSB R/nW bit. In the addressing formats with 10-bit addresses, the first byte is composed of 7 MSB slave address bits, such as 11110XX, where XX is the two MSB of the 10-bit addresses, and 1 LSB R/nW bit, which is 0 in this case.

The least significant R/nW of the address byte indicates the direction of transmission of the following data bytes. If R/nW is 0, the master writes data into the selected slave; if it is 1, the master reads data out of the slave.

**Figure 21-8. I2C Data Transfer Formats**




### 21.3.6.2 Master Transmitter

In this mode, data assembled in one of the previously described data formats is shifted out on the serial data line SDA in synch with the self-generated clock pulses on the serial clock line SCL. The clock pulses are inhibited and SCL held low when the intervention of the processor is required (XUDF) after a byte has been transmitted.

### 21.3.6.3 Master Receiver

This mode can only be entered from the master transmitter mode. With either of the address formats (Figure 21-8 (a), (b), and (c)), the master receiver is entered after the slave address byte and bit R/W_ has been transmitted, if R/W_ is high. Serial data bits received on bus line SDA are shifted in synch with the self-generated clock pulses on SCL. The clock pulses are inhibited and SCL held low when the intervention of the processor is required (ROVR) after a byte has been transmitted. At the end of a transfer, it generates the stop condition.

### 21.3.6.4 Slave Transmitter

This mode can only be entered from the slave receiver mode. With either of the address formats (Figure 21-8 (a), (b), and (c)), the slave transmitter is entered if the slave address byte is the same as its own address and bit R/W_ has been transmitted, if R/W_ is high. The slave transmitter shifts the serial data out on the data line SDA in synch with the clock pulses that are generated by the master device. It does not generate the clock but it can hold clock line SCL low while intervention of the CPU is required (XUDF).

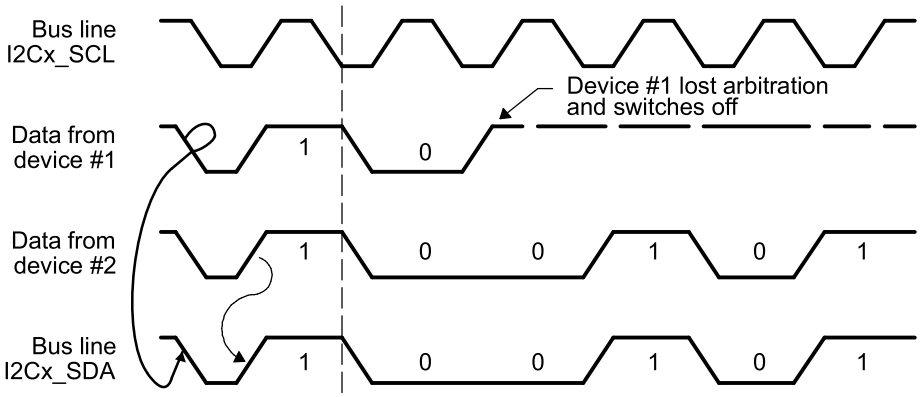
### 21.3.6.5 Slave Receiver

In this mode, serial data bits received on the bus line SDA are shifted-in in synch with the clock pulses on SCL that are generated by the master device. It does not generate the clock but it can hold clock line SCL low while intervention of the CPU is required (ROVR) following the reception of a byte.

### 21.3.7 Arbitration

If two or more master transmitters start a transmission on the same bus almost simultaneously, an arbitration procedure is invoked. The arbitration procedure uses the data presented on the serial bus by the competing transmitters. When a transmitter senses that a high signal it has presented on the bus has been overruled by a low signal, it switches to the slave receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration lost interrupt. Figure 21-9 shows the arbitration procedure between two devices. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

**Figure 21-9. Arbitration Procedure Between Two Master Transmitters**



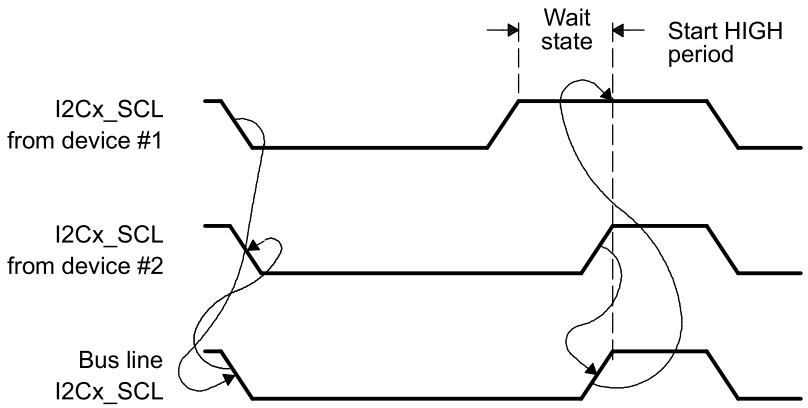

### 21.3.8 I2C Clock Generation and I2C Clock Synchronization

Under normal conditions, only one master device generates the clock signal, SCL. During the arbitration procedure, however, there are two or more master devices and the clock must be synchronized so that the data output can be compared. The wired-AND property of the clock line means that a device that first generates a low period of the clock line overrules the other devices. At this high/low transition, the clock generators of the other devices are forced to start generation of their own low period. The clock line is then held low by the device with the longest low period, while the other devices that finish their low periods must wait for the clock line to be released before starting their high periods. A synchronized signal on the clock line is thus obtained, where the slowest device determines the length of the low period and the fastest the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the WAIT-state. In this way a slave can slow down a fast master and the slow device can create enough time to store a received byte or to prepare a byte to be transmitted (Clock Stretching). Figure 21-10 illustrates the clock synchronization.

**Note:** If the SCL or SDA lines are stuck low, the Bus Clear operation is supported. If the clock line (SCL) is stuck low, the preferred procedure is to reset the bus using the hardware reset signal if your I2C devices have hardware reset inputs. If the I2C devices do not have hardware reset inputs, cycle power to the devices to activate the mandatory internal power-on reset (POR) circuit. If the data line (SDA) is stuck low, the master should send nine clock pulses. The device that held the bus low should release it sometime within those nine clocks. If not, use the hardware reset or cycle power to clear the bus.

**Figure 21-10. Synchronization of Two I2C Clock Generators**




### 21.3.9 Prescaler (SCLK/ICLK)

The I2C module is operated with a functional clock (SCLK) frequency that can be in a range of 12-100 MHz, according to I2C mode that must be used (an internal ~24 MHz clock (ICLK) is recommended in case of F/S operation mode e). Note that the frequency of the functional clock influences directly the I2C bus performance and timings.

The internal clock used for I2C logic - ICLK - is generated via the I2C prescaler block. The prescaler consists of a 4-bit register - I2C _PSC, and is used to divide the system clock (SCLK) to obtain the internal required clock for the I2C module.

### 21.3.10 Noise Filter

The noise filter is used to suppress any noise that is 50 ns or less, in the case of F/S mode of operation. It is designed to suppress noise with one ICLK. The noise filter is always one ICLK cycle, regardless of the bus speed. For FS mode (prescaler = 4, ICLK = 24 MHz), the maximum width of the suppressed spikes is 41.6 ns. To ensure a correct filtering, the prescaler must be programmed accordingly.

### 21.3.11 I2C Interrupts

The I2C module generates 12 types of interrupt: addressed as slave, bus free (stop condition detected), access error, start condition, arbitration-lost, noacknowledge, general call, registers-ready-for-access, receive and transmit data, receive and transmit draining. These 12 interrupts are accompanied with 12 interrupt masks and flags defined in the I2C_IRQENABLE_SET and respectively I2C_IRQSTATUS_RAW registers. Note that all these 12 interrupt events are sharing the same hardware interrupt line.

- Addressed As Slave interrupt (AAS) is generated to inform the Local Host that an external master addressed the module as a slave. When this interrupt occurs, the CPU can check the I2C_ACTOA status register to check which of the 4 own addresses was used by the external master to access the module.
- Bus Free interrupt (BF) is generated to inform the Local Host that the I2C bus became free (when a Stop Condition is detected on the bus) and the module can initiate his own I2C transaction.
- Start Condition interrupt (STC) is generated after the module being in idle mode have detected (synchronously or asynchronously) a possible Start Condition on the bus (signalized with WakeUp).
- Access Error interrupt (AERR) is generated if a Data read access is performed while RX FIFO is empty or a Data write access is performed while TX FIFO is full.
- Arbitration lost interrupt (AL) is generated when the I2C arbitration procedure is lost.
- No-acknowledge interrupt (NACK) is generated when the master I2C does not receive acknowledge from the receiver.
- General call interrupt (GC) is generated when the device detects the address of all zeros (8 bits).
- Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed address, data, and command have been performed and the status bits have been updated. This

interrupt is used to let the CPU know that the I2C registers are ready for access.

- Receive interrupt/status (RRDY) is generated when there is received data ready to be read by the CPU from the I2C_DATA register (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to read the received data from the I2C_DATA register.
- Transmit interrupt/status (XRDY) is generated when the CPU needs to put more data in the I2C_DATA register after the transmitted data has been shifted out on the SDA pin (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to write the next transmitted data into the I2C_DATA register.
- Receive draining interrupt (RDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be transferred and to enable the draining mechanism (see Section 21.3.14.4, *Draining Feature*, for additional details).
- Transmit draining interrupt (XDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be written and to enable the draining mechanism (see Section 21.3.14.4, *Draining Feature*, for additional details).

When the interrupt signal is activated, the Local Host must read the I2C_IRQSTATUS_RAW register to define the type of the interrupt, process the request, and then write into these registers the correct value to clear the interrupt flag.

### 21.3.12 DMA Events

The I2C module can generate two DMA requests events, read (I2C_DMA_RX) and write (I2C_DMA_TX) that can be used by the DMA controller to synchronously read received data from the I2C_DATA or write transmitted data to the I2C_DATA register. The DMA read and write requests are generated in a similar manner as RRDY and XRDY, respectively.

The I2C DMA request signals (I2C_DMA_TX and I2C_DMA_RX) are activated according to the FIFO Management subsection.

### 21.3.13 Interrupt and DMA Events

I2C has two DMA channels (Tx and Rx).

I2C has one interrupt line for all the interrupt requests.

For the event and interrupt numbers, see the device-specific datasheet.

### 21.3.14 FIFO Management

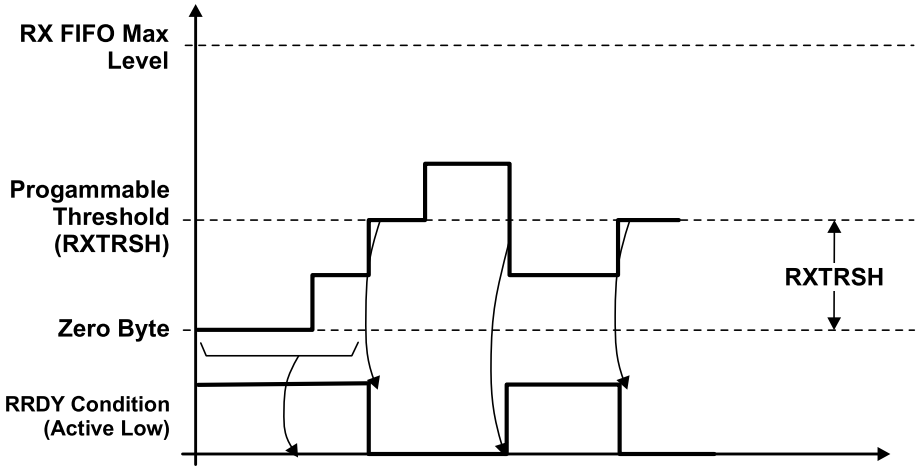
The I2C module implements two internal 32-bytes FIFOs with dual clock for RX and TX modes. The depth of the FIFOs can be configured at integration via a generic parameter which will also be reflected in I2C_IRQSTATUS_RAW.FIFODEPTH register.

#### 21.3.14.1 FIFO Interrupt Mode Operation

In FIFO interrupt mode (relevant interrupts enabled via I2C_IRQENABLE_SET register), the processor is informed of the status of the receiver and transmitter by an interrupt signal. These interrupts are raised when receive/transmit FIFO threshold (defined by I2C_BUF.TXTRSH or I2C_BUF.RXTRSH) are reached; the interrupt signals instruct the Local Host to transfer data to the destination (from the I2C module in receive mode and/or from any source to the I2C FIFO in transmit mode).

Figure 21-11 and Figure 21-12, respectively, illustrate receive and transmit operations from FIFO management point of view.

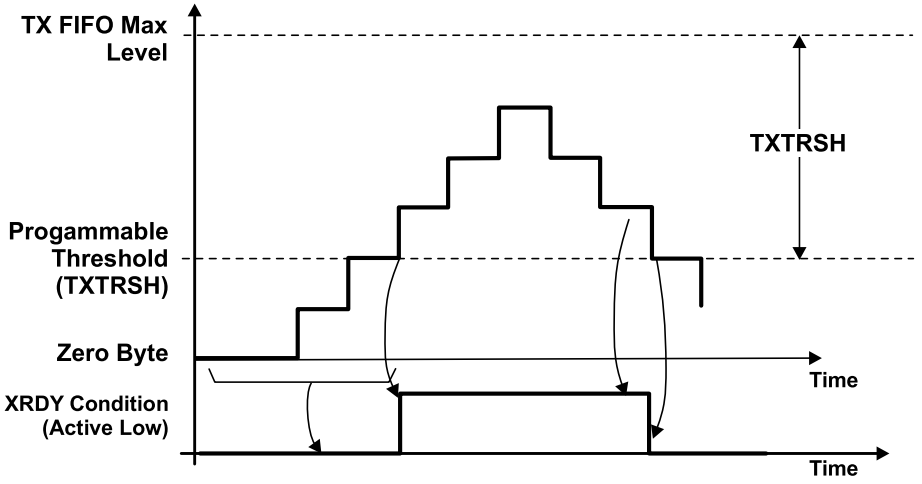
**Figure 21-11. Receive FIFO Interrupt Request Generation**




Note that in Figure 21-11, the RRDY Condition illustrates that the condition for generating a RRDY interrupt is achieved. The interrupt request is generated when this signal is active, and it can be cleared only by the CPU by writing a 1 in the corresponding interrupt flag. If the condition is still present after clearing the previous interrupt, another interrupt request will be generated.

In receive mode, RRDY interrupt is not generated until the FIFO reaches its receive threshold. Once low, the interrupt can only be de-asserted when the Local Host has handled enough bytes to make the FIFO level below threshold. For each interrupt, the Local Host can be configured to read an amount of bytes equal with the value of the RX FIFO threshold + 1.

**Figure 21-12. Transmit FIFO Interrupt Request Generation**




Note that in Figure 21-12, the XRDY Condition illustrates that the condition for generating a XRDY interrupt is achieved. The interrupt request is generated when this condition is achieved (when TX FIFO is empty, or the TX FIFO threshold is not reached and there are still data bytes to be transferred in the TX FIFO), and it can be cleared only by the CPU by writing a 1 in the corresponding interrupt flag after transmitting the configured number of bytes. If the condition is still present after clearing the previous interrupt, another interrupt request will be generated.

Note that in interrupt mode, the module offers two options for the CPU application to handle the interrupts:

- When detecting an interrupt request (XRDY or RRDY type), the CPU can write/read one data byte to/from the FIFO and then clear the interrupt. The module will not reassert the interrupt until the

interrupt condition is not met.

- When detecting an interrupt request (XRDY or RRDY type), the CPU can be programmed to write/read the amount of data bytes specified by the corresponding FIFO threshold (I2C_BUF.TXTRSH + 1 or I2C_BUF.RXTRSH + 1). In this case, the interrupt condition will be cleared and the next interrupt will be asserted again when the XRDY or RRDY condition will be again met.

If the second interrupt serving approach is used, an additional mechanism (draining feature) is implemented for the case when the transfer length is not a multiple of FIFO threshold (see Section 21.3.14.4, *Draining Feature*).

In slave TX mode, the draining feature cannot be used, since the transfer length is not known at the configuration time, and the external master can end the transfer at any point by not acknowledging one data byte.

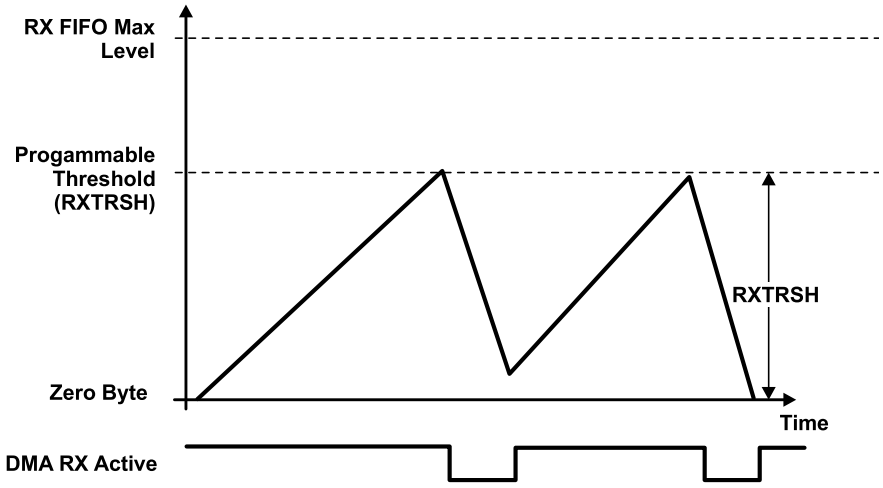
### 21.3.14.2 FIFO Polling Mode Operation

In FIFO polled mode (I2C_IRQENABLE_SET.XRDY_IE and I2C_IRQENABLE_SET.RRDY_IE disabled and DMA disabled), the status of the module (receiver or transmitter) can be checked by polling the XRDY and RRDY status registers (I2C_IRQSTATUS_RAW) (RDR and XDR can also be polled if draining feature must be used). The XRDY and RRDY flags are accurately reflecting the interrupt conditions mentioned in Interrupt Mode. This mode is an alternative to the FIFO interrupt mode of operation, where the status of the receiver and transmitter is automatically known by means of interrupts sent to the CPU.

### 21.3.14.3 FIFO DMA Mode Operation

In receive mode, a DMA request is generated as soon as the receive FIFO exceeds its threshold level defined in the threshold level register (I2C_BUF.RXTRSH +1). This request should be de-asserted when the number of bytes defined by the threshold level has been read by the DMA, by setting the I2C_DMARXENABLE_CLR.DMARX_ENABLE_CLEAR field.

**Figure 21-13. Receive FIFO DMA Request Generation**




In transmit mode, a DMA request is automatically asserted when the transmit FIFO is empty. This request should be de-asserted when the number of bytes defined by the number in the threshold register (I2C_BUF.TXTHRS+1) has been written in the FIFO by the DMA, by setting the I2C_DMATXENABLE_CLR. DMATX_ENABLE_CLEAR field. If an insufficient number of characters are written, then the DMA request will remain active. Figure 21-14 and Figure 21-15 illustrate DMA TX transfers with different values for TXTRSH.

**Figure 21-14. Transmit FIFO DMA Request Generation (High Threshold)**

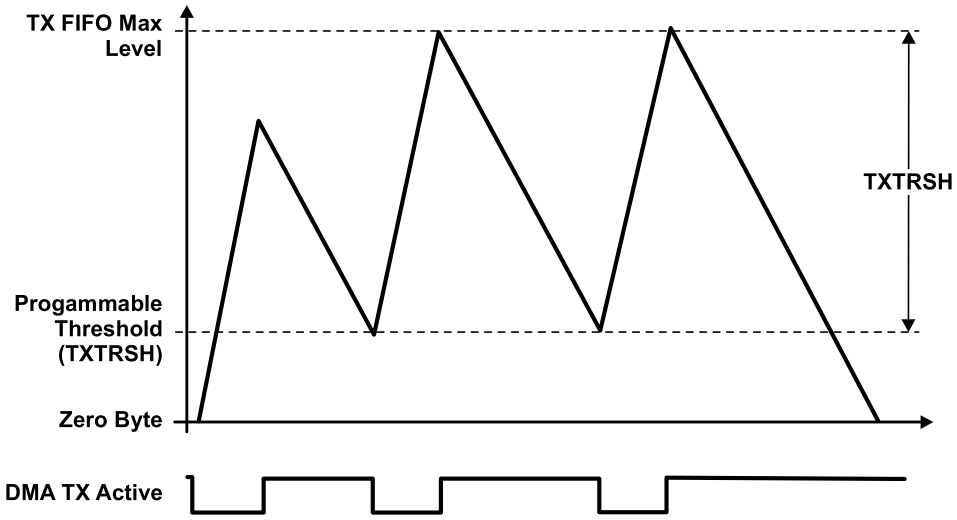
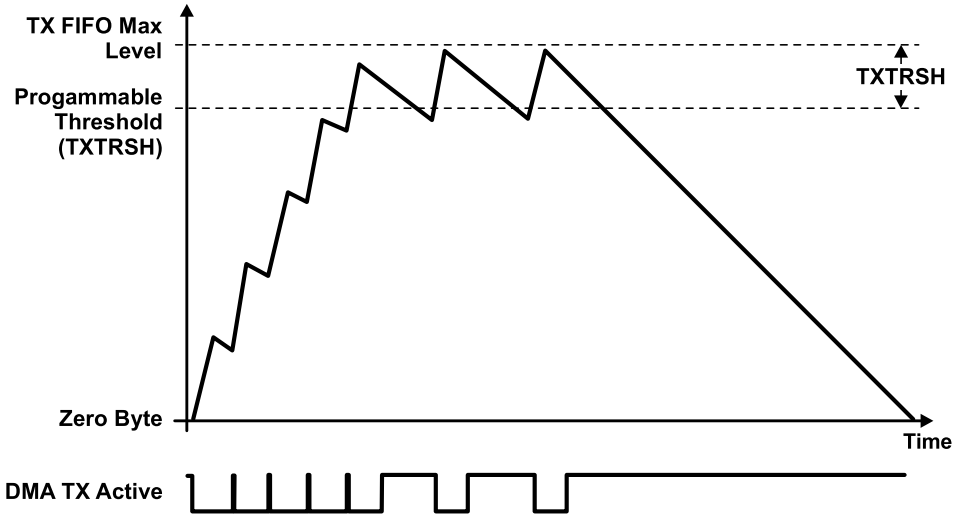



**Figure 21-15. Transmit FIFO DMA Request Generation (Low Threshold)**




Note that also in DMA mode it is possible to have a transfer whose length is not multiple of the configured FIFO threshold. In this case, the DMA draining feature is also used for transferring the additional bytes of the transfer (see Section 21.3.14.4, *Draining Feature*, for additional details).

According to the desired operation mode, the programmer must set the FIFO thresholds according to the following table (note that only the interface/OCP side thresholds can be programmed; the I2C side thresholds are default equals to 1). Note that the thresholds must be set consistent with the DMA channel length.

In I2C Slave RX Mode, the Local Host can program the RX threshold with the desired value, and use the FIFO draining feature at the end of the I2C transfer to extract from the FIFO the remaining bytes if the threshold is not reached (see Section 21.3.14.4, *Draining Feature*, for additional details).

Note that in I2C Slave TX Mode, the TX FIFO threshold should be set to 1 (I2C_BUF.TXTRSH=0, default value), since the length of the transfer may not be known at configuration time. In this way, the interrupt (or accordingly, DMA) requests will be generated for each byte requested by the remote I2C master to be transferred over the I2C bus. This configuration will prevent the I2C core to request additional data from the CPU or from the DMA controller (using IRQ or DMA), data that will eventually not be extracted from the FIFO by the external master (which can use not acknowledge at any time to end the transfer). If the TX threshold is not set to 1, the module will generate an interrupt or assert a DMA only when the external master requests a byte and the FIFO is empty. However, in this case the TX FIFO will require to be cleared at the end of the transfer.

The I2C module offers the possibility to the user to clear the RX or TX FIFO. This is achieved through I2C_BUF.RXFIFO_CLR and I2C_BUF.TXFIFO_CLR registers, which act like software reset for the FIFOs. In DMA mode, these bits will also reset the DMA state machines.

The FIFO clearing feature can be used when the module is configured as a transmitter, the external receiver responds with a NACK in the middle of the transfer, and there is still data in TX FIFO waiting to be transferred.

On the Functional (I2C) domain, the thresholds can always be considered equal to 1. This means that the I2C Core can start transferring data on the I2C bus whenever it has data in the FIFOs (FIFO is not empty).

#### 21.3.14.4 Draining Feature

The Draining Feature is implemented by the I2C core for handling the end of the transfers whose length is not a multiple of FIFO threshold value, and offers the possibility to transfer the remaining amount of bytes (since the threshold is not reached).

Note that this feature prevents the CPU or the DMA controller to attempt more FIFO accesses than necessary (for example, to generate at the end of a transfer a DMA RX request having in the FIFO less bytes than the configured DMA transfer length). Otherwise, an Access Error interrupt will be generated (see I2C_IRQSTATUS_RAW.AERR interrupt).

The Draining mechanism will generate an interrupt (I2C_IRQSTATUS_RAW.RDR or I2C_IRQSTATUS_RAW.XDR) at the end of the transfer informing the CPU that it needs to check the amount of data left to be transferred (I2C_BUFSTAT.TXSTAT or RXSTAT) and to enable the Draining Feature of the DMA controller if DMA mode is enabled (by re-configuring the DMA transfer length according to this value), or perform only the required number of data accesses, if DMA mode is disabled.

In receiving mode (master or slave), if the RX FIFO threshold is not reached but the transfer was ended on the I2C bus and there is still data left in the FIFO (less than the threshold), the receive draining interrupt (I2C_IRQSTATUS_RAW.RDR) will be asserted to inform the local host that it can read the amount of data in the FIFO (I2C_BUFSTAT.RXSTAT). The CPU will perform a number of data read accesses equal with RXSTAT value (if interrupt or polling mode) or re-configure the DMA controller with the required value in order to drain the FIFO.

In master transmit mode, if the TX FIFO threshold is not reached but the amount of data remaining to be written in the FIFO is less than TXTRSH, the transmit draining interrupt (I2C_IRQSTATUS_RAW.XDR) will be asserted to inform the local host that it can read the amount of data remained to be written in the TX FIFO (I2C_BUFSTAT.TXSTAT). The CPU will need to write the required number of data bytes (specified by TXSTAT value) or re-configure the DMA controller with the required value in order to transfer the last bytes to the FIFO.

Note that in master mode, the CPU can alternatively skip the checking of TXSTAT and RXSTAT values since it can obtain internally this information (by computing DATACOUNT modulo TX/RXTHRSH).

The draining feature is disabled by default, and it can be enabled using I2C_IRQENABLE_SET.XDR_IE or I2C_IRQENABLE_SET.RDR_IE registers (default disabled) only for the transfers with length not equal with the threshold value.

### *21.3.15 How to Program I2C*

#### 21.3.15.1 Module Configuration Before Enabling the Module

1. Program the prescaler to obtain an approximately 12-MHz I2C module clock (I2C_PSC = x; this value is to be calculated and is dependent on the System clock frequency).
2. Program the I2C clock to obtain 100 Kbps or 400 Kbps (SCLL = x and SCLH = x; these values are to be calculated and are dependent on the System clock frequency).
3. Configure its own address (I2C_OA = x) - only in case of I2C operating mode (F/S mode).
4. Take the I2C module out of reset (I2C_CON:I2C_EN = 1).

#### 21.3.15.2 Initialization Procedure

1. Configure the I2C mode register (I2C_CON) bits.
2. Enable interrupt masks (I2C_IRQENABLE_SET), if using interrupt for transmit/receive data.
3. Enable the DMA (I2C_BUF and I2C_DMA/RX/TX/ENABLE_SET) and program the DMA controller) - only in case of I2C operating mode (F/S mode), if using DMA for transmit/receive data.

#### 21.3.15.3 Configure Slave Address and DATA Counter Registers

In master mode, configure the slave address (I2C_SA = x) and the number of byte associated with the transfer (I2C_CNT = x).

#### 21.3.15.4 Initiate a Transfer

Poll the bus busy (BB) bit in the I2C status register (I2C_IRQSTATUS_RAW). If it is cleared to 0 (bus not busy), configure START/STOP (I2C_CON: STT / I2C_CON: STP condition to initiate a transfer) - only in case of I2C operating mode (F/S mode).

#### 21.3.15.5 Receive Data

Poll the receive data ready interrupt flag bit (RRDY) in the I2C status register (I2C_IRQSTATUS_RAW), use the RRDY interrupt (I2C_IRQENABLE_SET.RRDY_IE set) or use the DMA RX (I2C_BUF.RDMA_EN set together with I2C_DMARXENABLE_SET) to read the received data in the data receive register (I2C_DATA). Use draining feature (I2C_IRQSTATUS_RAW.RDR enabled by I2C_IRQENABLE_SET.RDR_IE)) if the transfer length is not equal with FIFO threshold.

#### 21.3.15.6 Transmit Data

Poll the transmit data ready interrupt flag bit (XRDY) in the I2C status register (I2C_IRQSTATUS_RAW), use the XRDY interrupt (I2C_IRQENABLE_SET.XRDY_IE set) or use the DMA TX (I2C_BUF.XDMA_EN set together with I2C_DMATXENABLE_SET) to write data into the data transmit register (I2C_DATA). Use draining feature (I2C_IRQSTATUS_RAW.XDR enabled by I2C_IRQENABLE_SET.XDR_IE)) if the transfer length is not equal with FIFO threshold.

### *21.3.16 I2C Behavior During Emulation*

To configure the I2C to stop during emulation suspend events (for example, debugger breakpoints), set up the I2C and the Debug Subsystem:

1. Set I2C_SYSTEST.FREE=0. This will allow the Suspend_Control signal from the Debug Subsystem (Chapter 27) to stop and start the I2C. Note that if FREE=1, the Suspend_Control signal is ignored and the I2C is free running regardless of any debug suspend event. This FREE bit gives local control from a module perspective to gate the suspend signal coming from the Debug Subsystem.
2. Set the appropriate xxx_Suspend_Control register = 0x9, as described in Section 27.1.1.1, *Debug Suspend Support for Peripherals*. Choose the register appropriate to the peripheral you want to suspend during a suspend event.