I2C Controller

20.1 Introduction

This I2C module is a bus controller that can function as a master or a slave in a multi-master design. It supports an extremely wide clock frequency range from DC (almost) up to 400 Kb/s.

In master mode, a transfer can only be initiated by the processor writing the slave address into the I2C address register. The processor is notified of any available received data by a data interrupt or a transfer complete interrupt. If the HOLD bit is set, the I2C interface holds the SCL line Low after the data is transmitted to support slow processor service. The master can be programmed to use both normal (7-bit) addressing and extended (10-bit) addressing modes.

In slave monitor mode, the I2C interface is set up as a master and continues to attempt a transfer to a particular slave until the slave device responds with an ACK.

The HOLD bit can be set to prevent the master from continuing with the transfer, preventing an overflow condition in the slave.

A common feature between master mode and slave mode is the timeout (TO) interrupt flag. If at any point the SCL line is held Low by the master or the accessed slave for more than the period specified in the Timeout register, a timeout (TO) interrupt is generated to avoid stall conditions.

20.1.1 Features

The PS supports two I2C devices with these key features:

- I2C bus specification version 2
- Supports 16-byte FIFO
- Programmable normal and fast bus data rates
- Master mode
  - Write transfer
  - Read transfer
  - Extended address support
  - Support HOLD for slow processor service
  - Supports TO interrupt flag to avoid stall condition
- Slave monitor mode

- Slave mode
  - Slave transmitter
  - Slave receiver
  - Fully programmable slave response address
  - Supports HOLD to prevent overflow condition
  - Supports TO interrupt flag to avoid stall condition
- Software can poll for status or function as interrupt-driven device
- Programmable interrupt generation

20.1.2 System Block Diagram

The system viewpoint diagram for the I2C module is shown in Figure 20-1.

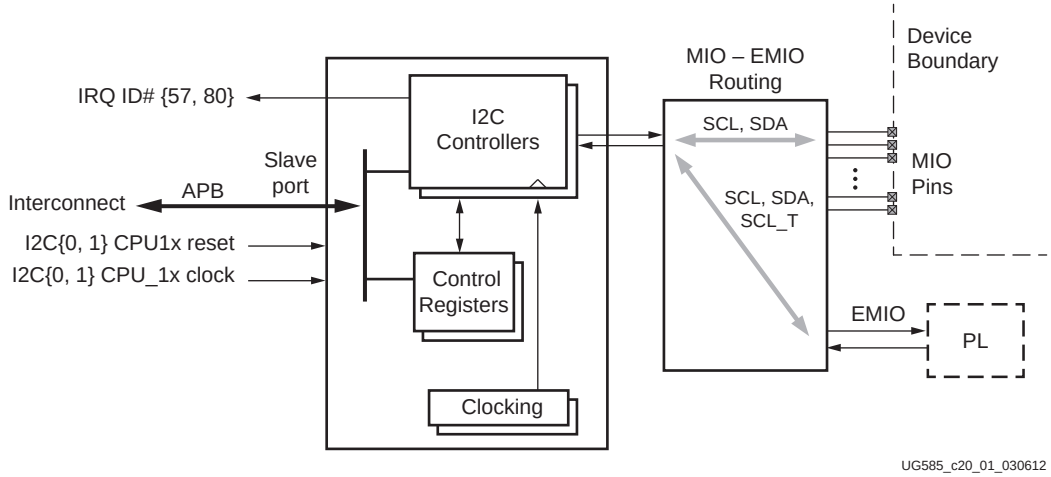

Figure 20-1: **I2C System Block Diagram**

20.1.3 Notices

The 7z010 CLG225 device supports 32 MIO pins as shown in section 2.5.4 MIO-at-a-Glance Table. This restricts the availability of the I2C signals on the MIO pins. As needed, I2C signals can be routed through the EMIO interface and passed-through to the PL pins. All 7z010 CLKG225 device restrictions are listed in section 1.1.3 Notices.

20.2 Functional Description

20.2.1 Block Diagram

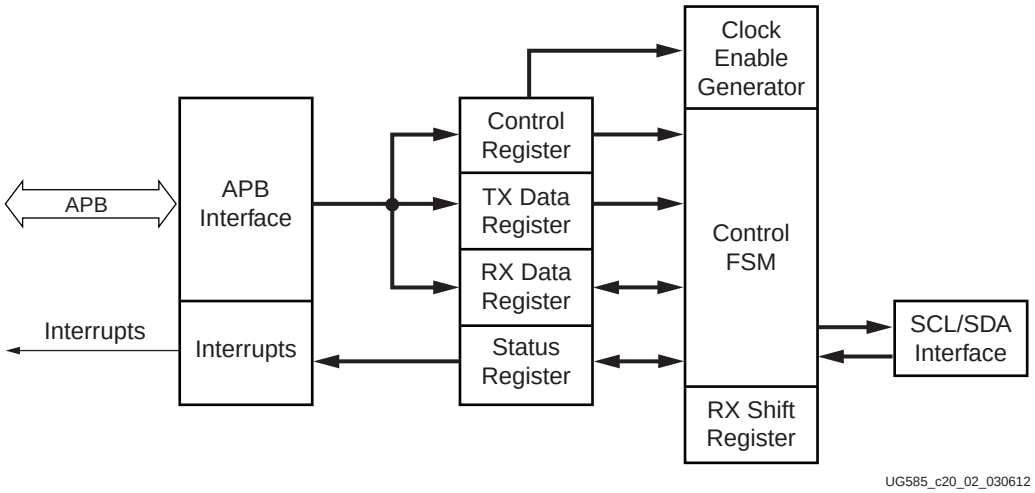

Figure 20-2: **I2C Peripheral Block Diagram**

20.2.2 Master Mode

An I2C transfer can only be initiated by the APB host, and two types of transfers can be performed:

- Write transfer, where the I2C becomes master transmitter
- Read transfer, where the I2C becomes master receiver

Write Transfer

To accomplish an I2C write transfer, the host must perform these steps:

1. Write to the control register to set up SCL speed and addressing mode.
2. Set the MS, ACKEN, and CLR_FIFO bits and clear the RW bit in the Control register.
3. If required, set the HOLD bit. Otherwise write the first byte of data to the I2C Data register.
4. Write the slave address into the I2C address register. This initiates the I2C transfer.
5. Continue to load the remaining data to be sent to the slave by writing to the I2C Data register. The data is pushed in the FIFO each time the host writes to the I2C Data register.

When all data is transferred successfully, the COMP bit is set in the interrupt status register. A data interrupt is generated whenever there are only two bytes left for transmission in the FIFO.

When all data is transferred successfully, If the HOLD bit is not set, the I2C interface generates a STOP condition and terminates the transfer. If the HOLD bit is set, the I2C interface holds the SCL line Low

after the data is transmitted. The host is notified of this event by a transfer complete interrupt (COMP bit set) and the TXDV bit in the status register is cleared. At this point, the host can proceed in three ways:

1. Clear the HOLD bit. This causes the I2C interface to generate a STOP condition.
2. Supply more data by writing to the I2C address register. This causes the I2C interface to continue with the transfer, writing more data to the slave.

If at any point the slave responds with a NACK, the transfer automatically terminates and a transfer NACK interrupt is generated (the NACK bit set). When a NACK is received the Transfer Size register indicates the number of bytes that still need to be sent minus one. Unless the very last byte written by the host into the FIFO was a NACK byte, TXDV remains High. In this case, the host must clear the FIFO by setting the CLR_FIFO bit in the Control register.

If at any point the SCL line is held Low by the master or the accessed slave for more than the period specified in the Timeout register, a TO interrupt is generated and the outstanding amount of data minus one is then read from the Transfer Size register.

Read Transfer

To accomplish an I2C read transfer, the host must perform these steps:

1. Write to the Control register to set up the SCL speed and addressing mode.
2. Set the MS, ACKEN, CLR_FIFO bits, and the RW bit in the Control register.
3. If the host wants to hold the bus after the data is received, it must also set the HOLD bit.
4. Write the number of requested bytes in the Transfer Size register.
5. Write the slave address in the I2C Address register. This initiates the I2C transfer.

The host is notified of any available received data in two ways:

1. If an outstanding transfer size is more than the FIFO size -2, a data interrupt is generated (DATA bit set) when there are two free locations available in the FIFO.
2. If an outstanding transfer size is less than FIFO size -2, a transfer complete interrupt is generated (COMP bit set) when the outstanding transfer size bytes are received.

In both cases, the RXDV bit in the status register is set.

The I2C interface automatically returns a NACK after receiving the last expected byte and terminates the transfer by generating a STOP condition. If the HOLD bit is set during a master read transfer, the I2C interface drives the SCL line Low.

If at any point the slave responds with NACK while the master transmits a slave address for a master read transfer, the transfer automatically terminates and a transfer NACK interrupt is generated (NACK bit is set). The outstanding amount of data can be read from the Transfer Size register.

If at any point the SCL line is held Low by the master or the accessed slave for more than the period specified in the Timeout register, a TO interrupt is generated.

20.2.3 Slave Monitor Mode

This mode is meaningful only when the module is in master mode and bit SLVMON in the control register is set. The host must set the MS and SLVMON bits and clear the RW bit in the Control register. Also, it must initialize the Slave Monitor Pause register.

The master attempts a transfer to a particular slave whenever the host writes to the I2C Address register. If the slave returns a NACK when it receives the address, the master waits for the time interval established by the Slave Monitor Pause register and attempts to address the slave again. The master continues this cycle until the slave responds with an ACK to its address or until the host clears the SLVMON bit in the Control register. If the addressed slave responds with an ACK, the I2C interface terminates the transfer by generating a STOP condition and a SLV_RDY interrupt.

20.2.4 Slave Mode

The I2C interface is set up as a slave by clearing the MS bit in the Control register. The I2C slave must be given a unique identifying address by writing to the I2C address register. The SCL speed must also be set up at least as fast as the fastest SCL frequency expected to be seen. When in slave mode, the I2C interface operates as either a slave transmitter or a slave receiver.

Slave Transmitter

The slave becomes a transmitter after recognizing the entire slave address sent by the master and when the R/W field in the last address byte sent is High. This means that the slave has been requested to send data over the I2C bus and the host is notified of this through an interrupt through an interrupt to the GIC (refer to Figure 20-1, page 606). At the same time, the SCL line is held Low to allow the host to supply data to the I2C slave before the I2C master starts sampling the SDA line. The host is notified of this event by setting the DATA interrupt flag.

At the same time, the SCL line is held Low to allow the host to supply data to the I2C slave before the I2C master starts sampling the SDA line.

The host must supply data for transmission through the I2C data register so that the SCL line is released and transfer continues. If it does not write to the I2C data register before the timeout period expires, an interrupt is generated and a TO interrupt flag is set.

After the host writes to the I2C data register, the transfer continues by loading data in the FIFO while the transfer is in progress. The amount of data loaded in the FIFO might be a known system parameter or communicated in advance through a higher level protocol using the I2C bus.

When there are only two valid bytes left in the FIFO for transmission, an interrupt is generated and the DATA interrupt flag is set. If the I2C master returns a NACK on the last byte transmitted from the FIFO, an interrupt is generated, and the COMP interrupt flag is set as soon as the I2C master generates a STOP condition.

The transfer must continue if the master acknowledges on the last byte sent out from the FIFO. At that moment, the DATA interrupt flag is set. The TXDV flag in the Status register is cleared because the FIFO is empty.

If the I2C master terminates the transfer before all of the data in the FIFO is sent by the slave, the host is notified, and the NACK interrupt flag is set while TXDV remains set and the Transfer Size register indicates the remaining bytes in the FIFO. The host must set the CLR_FIFO bit in the Control register to clear the FIFO and the TXDV bit.

Slave Receiver

The slave becomes a receiver after recognizing the entire slave address sent by the master and when the R/W bit in the first address byte is Low. This means that the master is about to send one or more data bytes to the slave over the I2C bus.

After a byte is acknowledged by the I2C slave, the RXDV bit in the Status register is set, indicating that new data has been received. The host reads the received data through the I2C Data register. An interrupt is generated and the DATA interrupt flag is set when there are only two free locations left in the FIFO.

Whenever the I2C master generates a STOP condition, an interrupt is generated and the COMP interrupt flag is set. The Transfer Size register then contains the number of bytes received that are available in the FIFO. This number is decremented by the host on each read of the I2C Data register.

If the FIFO is full when one or more bytes are received by the I2C interface, an interrupt is generated and the RX_OVF interrupt flag is set. The last byte received is not acknowledged and the data in the FIFO is kept intact.

The HOLD bit can be set in the Control register to avoid overflow conditions when it is impossible to respond to interrupts in a reasonable time. If the HOLD bit is set, the I2C interface keeps the SCL line Low until the host clears resources for data reception. This prevents the master from continuing with the transfer, causing an overflow condition in the slave. The host clears resources for data reception by reading the data register.

If the HOLD bit is set and the I2C interface keeps the SCL line Low for longer than the timeout period, an interrupt is generated and the TO interrupt flag is set.

20.2.5 I2C Speed

The main clock used within the I2C interface is the clock enable signal (see Figure 20-3).

- In slave mode, the clock enable is used to extract synchronization information for correct sampling of the SDA line.
- In master mode, the clock enable is used to establish a time base for generating the desired SCL frequency.

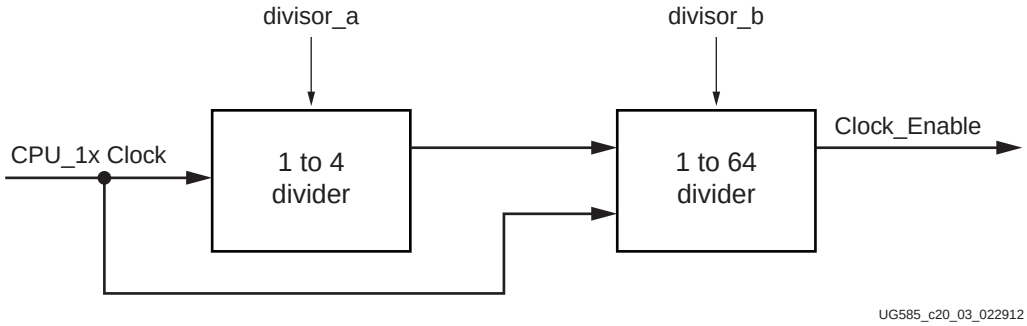

Figure 20-3: **I2C Clock Generator**

The frequency of the clock_enable signal is defined by the frequency of the CPU_1x clock and the values of divisor_a and divisor_b using Equation 20-1

$$FreqClock\_Enable = \frac{FreqCPU\_1x}{(divisor\_a + 1) \times (divisor\_b + 1)} \quad \textit{Equation 20-1}$$

Note: As seen from the above calculation, the SCL clock frequency range is limited by the cpu_1x clock. This means that for some cpu_1x clock rates there will be some SCL frequencies that are not possible.

See I2C register map in Appendix B, Register Details for details of register fields.

Table 20-1 lists the calculated values for standard and high speed SCL clock values. A programming example is provided in Section 20.3.2.

I2C SCL Clock = CPU_1X_Clock / (22 * (divisor_a + 1) * (divisor_b + 1))

Table 20-1: **Calculated Values for Standard and High Speed SCL Clock Values**

I2C SCL Clock	CPU_1X_Clock	divisor_a	divisor_b
100 KHz	111 MHz	2	16
400 KHz	111 MHz	0	12
100 KHz	133 MHz	0	60
400 KHz	133 MHz	2	4
100 KHz	166 MHz	3	16

20.2.6 Multi-Master Operation

In I2C multi master mode, the bus is shared with other masters. In this mode, the I2C clock (I2C_SCL)is driven by the device that acts as the master.

20.2.7 I2C0-to-I2C1 Connection

The I/O signals of the two I2C controllers in the PS are connected together when the slcr.MIO_LOOPBACK [I2C0_LOOP_I2C1] bit is set = 1. In this mode, the serial clocks are connected together and the serial data signals are connected together.

20.2.8 Status and Interrupts

The registers i2c.Interrupt_status_reg0, i2c.Intrpt_mask_reg0, i2c.Intrpt_enable_reg0 and i2c.Intrpt_disable_reg0 provide interrupt capability. See Table 20-2 for Interrupt and Status register names and bit assignments.

Table 20-2: **Interrupt and Status Register Names and Bit Assignments**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
i2c.Interrupt_status_reg0 i2c.Intrpt_mask_reg0 i2c.Intrpt_enable_reg0 i2c.Intrpt_disable_reg0															
X	X	X	X	X	X	ARB_LOST	X	RX_UNF	TX_OVF	RX_OVF	SLV_RDY	TO	NACK	DATA	COMP
i2c.Status_reg0															
X	X	X	X	X	X	X	BA	RXOVF	TXDV	RXDV	X	RX_RW	X	X	X

Interrupt Mask Register

Intrpt_mask_reg0 is a read-only interrupt mask register used to enable/disable individual interrupts in the i2c.interrupt_status_reg0 register:

- If the mask bit = 0, the interrupt is enabled.
- If the mask bit = 1, the interrupt is disabled.

This mask is controlled by the write-only Intrpt_enable_reg0 and Intrpt_disable_reg0 registers.

Each associated enable/disable interrupt bit should be set mutually exclusive (e.g., to enable an interrupt, write 1 to Intrpt_enable_reg0[x] and write 0 to Intrpt_disable_reg0[x]).

Interrupt Status Register

All the bits are sticky.

- Read: Reads interrupt status.
- Write: Write 1 to clear

Status Register

All bits present the raw status of the interface. Bits in this register dynamically change based on FIFO and other conditions.

20.3 Programmer's Guide

20.3.1 Start-up Sequence

1. **Reset controller:** Programming resets is described in section 20.4.2 Reset Controller.
2. **Configure I/O signal routing**: I2C signals SCL and SDA can be routed to either MIO or EMIO. Refer to Table 20-4. Example of I2C SCL and SDA signal routed to MIO is provided in section 20.5.1 Pin Programming.
3. **Configure Clocks:** The I2C clock architecture is described in section 20.4.1 Clocks.
4. **Controller Configuration:** Program I2C transfer parameters using i2c.Control_reg0 etc. Refer to section 20.4.2 Reset Controller.
5. **Configure Interrupts:** Interrupts help to control data in FIFO. Refer to section 20.2.8 Status and Interrupts and programming example in section 20.3.3 Configure Interrupts.
6. **Data Transfers:** Transfers in master mode and slave monitor mode can be referred in section 20.3.4 Data Transfers.

20.3.2 Controller Configuration

The user should choose interface mode, addressing mode, direction of transfer, timeout, and program the I2C bus speed before initiating an I2C transfer. Optionally, the user can clear FIFOs and hold the bus if required, in case of large data or combined transfers.

Example: Master Write Transfer

1. **Configure the Control parameters**. Write `0x0000_324E` to the i2c.Control_reg0 register:
   a. Select Master mode: Set i2c.Control_reg0[MS] = `1`.
   b. Set Direction of transfer as Master Transmitter: Set i2c.Control_reg0[RW] = `0`.
   c. Select Normal Addressing [7-bit] mode: Set i2c.Control_reg0[NEA] = `1`.
   d. Enable the transmission of ACK: Set i2c.Control_reg0[ACKEN] = `1`.

e. Clear the FIFOs: Set i2c.Control_reg0[CLR_FIFO] = 1.

f. Program the clock divisors:

- Set i2c.Control_reg0[divisor_a] = 0.
- Set i2c.Control_reg0[divisor_b] = 50.

These divisors generate an I2C SCL of 99 KHz using the CPU_1X clock of 111 MHz. For further details refer to section 20.2.5 I2C Speed.

2. **Configure Timeout.** Write 0x0000_00FF to the i2c.Time_out_reg0 register. Wait 255 SCL cycles when the SCL is held Low, before generating a timeout interrupt.

20.3.3 Configure Interrupts

The interrupts are described in section 20.2.8 Status and Interrupts. The i2c.Intrpt_enable_reg0 register has bits to enable the interrupt mask and i2c.Intrpt_disable_reg0 has bits to forcefully disable the interrupts. Each pair of bits should be set mutually exclusive:

Example: Program Example to Configure Completion Interrupt

1. **Enable the completion interrupt**. Set the enable bit, clear the disable bit, and verify the mask value:
   a. Set i2c.Intrpt_enable_reg0[COMP] = 1.
   b. Clear i2c.Intrpt_disable_reg0[COMP] = 0.
   c. i2c.intrpt_mask_reg0[COMP] reads back = 1.
2. **Disable the completion interrupt**. Set the enable bit, clear the disable bit, and verify the mask value:
   a. Set i2c.Intrpt_disable_reg0[COMP] = 1.
   b. Clear i2c.Intrpt_enable_reg0[COMP] = 0.
   c. i2c.Intrpt_mask_reg0[COMP] reads back = 0.
3. **Monitor completion interrupt**. I2c.Interrupt_status_reg0 provides status of completion interrupt.:
   a. Read i2c.Interrupt_status_reg0 [COMP]. 1 indicates that an interrupt occurred.
4. **Clear completion interrupt**. Write 1 to the i2c.Interrupt_status_reg0[COMP] bit field.

20.3.4 Data Transfers

Transfers can be achieved in polled mode or interrupt driven mode. Limitation on data count while performing a master read transfer is 255 bytes. Below are examples of read and write transfer in Master mode and example for Slave Monitor mode. Refer to section 20.2.3 Slave Monitor Mode for details.

Example: Master Read Using Polled Method

1. **Set direction of transfer as read and clear the FIFOs**. Write 0x41 to i2c.Control_reg0.

2. **Clear interrupts**. Read and write back the read value to i2c.Interrupt_status_reg0.
3. **Write read data count to transfer size register and hold bus if required.** Write read data count value to i2c.Transfer_size_reg0. If read data count greater than FIFO depth**,** set i2c.Control_reg0 [HOLD] = 1.
4. **Write the slave address**. Write the address to the i2c.I2C_address_reg0 register.
5. **Wait for data to be received into the FIFO.** Poll on i2c.Status_reg0 [RXDV] = 1.
   a. If i2c.Status_reg0 [RXDV] = 0, and any of i2c.Interrupt_status_register [NACK], i2c.Interrupt_status_register [ARB_LOST], i2c.Interrupt_status_register [RX_OVF], i2c.Interrupt_status_register [RX_UNF] interrupts are set, then stop the transfer and report the error, otherwise continue to poll on i2c.Status_reg0 [RXDV].
   b. If i2c.Status_reg0 [RXDV] = 1, and if any of i2c.Interrupt_status_register [NACK], i2c.Interrupt_status_register [ARB_LOST], i2c.Interrupt_status_register [RX_OVF], i2c.Interrupt_status_register [RX_UNF] interrupts are set, then stop the transfer and report the error. Otherwise, go to step 6.
6. **Read data and update count.** Read data from FIFO until i2c.Status_reg0[RXDV] = 1. Decrement the read data count and if it is less than or equal to the FIFO depth, clear i2c.Control_reg0[HOLD].
7. **Check for Completion of transfer**. If total read count reaches zero, poll on i2c.Interrupt_status_reg0 [COMP] = 1. Otherwise continue from step 5.

Example: Master Write Using Polled Method

1. **Set Direction of transfer as write and Clear the FIFO's**. Write 0x40 to i2c.Control_reg0.
2. **Clear Interrupts**. Read and write back the read value to i2c.Interrupt_status_reg0.
3. **Calculate the space available in FIFO**. Subtract i2c.Transfer_size_reg0 value from FIFO depth.
4. **Fill the data into FIFO**. Write the data to i2c.I2C_data_reg0 based on the count obtained in step 3.
5. **Write the Slave Address**. Write the address to i2c.I2C_address_reg0 register.
6. **Wait for TX FIFO to be empty**. Poll on i2c.Status_reg0 [TXDV] = 0.
   a. If i2c.Status_reg0 [TXDV] = 1, any of i2c.Interrupt_status_register [NACK], i2c.Interrupt_status_register [ARB_LOST], i2c.Interrupt_status_register [RX_OVF], i2c.Status_register [TX_OVF] are set, then stop the transfer and report the error otherwise continue to poll.
   b. If i2c.Status_reg0 [TXDV] = 0, repeat step 3, 4 and 6 until there is no further data.
7. **Wait for completion of transfer**. Check for i2c.Interrupt_status_reg0 [COMP] = 1.

Example: Master Read Using Interrupt Method

1. **Set direction of transfer as read and clear the FIFO's**. Write 0x41 to i2c.Control_reg0.
2. **Clear interrupts**. Read and write back the read value to i2c.Interrupt_status_reg0.
3. **Enable Timeout, NACK, Rx overflow, Arbitration lost, DATA, Completion interrupts**. Write 0x22F to i2c.Intrpt_en_reg0.
4. **Write read data count to transfer size register and hold bus if required.** Write read data count value to i2c.Transfer_size_reg0. If read data count greater than FIFO depth**,** set i2c.Control_reg0 [HOLD].

5. **Write the slave address**. Write the address to the i2c.I2C_address_reg0 register.
6. **Wait for data to be received into FIFO**.
   a. If read data count is greater than FIFO depth, wait for i2c.Interrupt_status_reg0 [DATA] = 1. Read 14 bytes from FIFO. Decrement the read data count by 14 and if it is less than or equal to the FIFO depth, clear i2c.Control_reg0[HOLD]
   b. Otherwise, wait for i2c.Interrupt_status_reg0 [COMP] = 1 and read data from the FIFO based on the read data count.
7. **Check for completion of transfer**. Check if read count reaches zero. Otherwise repeat from step 6.

Example: Master Write Using Interrupt Method

1. **Set direction of transfer as write and clear the FIFO's**. Write 0x40 to i2c.Control_reg0.
2. **Clear Interrupts**. Read and write back the read value to i2c.Interrupt_status_reg0.
3. **Enable Timeout, NACK, Tx Overflow, Arbitration lost, DATA, Completion interrupts**. Write 0x24F to the i2c.Intrpt_en_reg0 register.
4. **Enable bus HOLD logic**. Set i2c.Control_reg0 [HOLD] if the write data count is greater than the FIFO depth.
5. **Calculate the space available in FIFO**. Subtract the i2c.Transfer_size_reg0 value from the FIFO depth.
6. **Fill the data into FIFO**. Write the data to i2c.I2C_data_reg0 based on the count obtained in step 5.
7. **Write the slave address**. Write the address to the i2c.I2C_address_reg0 register.
8. **Wait for data to be sent**. Check for i2c.Interrupt_status_reg0 [COMP] to be set.
   a. If further data is to be written, repeat steps 5, 6 and 8.
   b. If there is no further data, set i2c.Control_reg0 [HOLD] = 0.
9. **Wait for completion of transfer**. Check for i2c.Interrupt_status_reg0 [COMP] to be set.

Example: Slave Monitor Mode

Slave monitor mode helps to monitor when the slave is in the busy state. The slave ready interrupt occurs only when slave is not busy. This can be done only in master mode:

1. **Select slave monitor mode and clear the FIFOs.** Write 0x60 to i2c.Control_reg0.
2. **Clear interrupts**. Read and write back the read value to i2c.Interrupt_status_reg0.
3. **Enable Interrupts**. Set i2c.Intrpt_en_reg0 [SLV_RDY] = 1.
4. **Set slave monitor delay**. Set i2c.Slave_mon_pause_reg0 with 0xF.
5. **Write the Slave Address**. Write the address to the i2c.I2C_address_reg0 register.
6. **Wait for slave to be ready**. Poll on i2c.Interrupt_status_reg0 [SLV_RDY] = 1.

20.3.5 Register Overview

An overview of the I2C registers is provided in Table 20-3.

Table 20-3: **I2C Register Overview**

Function	Register Names	Overview
Configuration	Control_reg0	Configure the operating mode
Data	I2C_address_reg0 I2C_data_reg0 Transfer_size_register0 Slave_mon_pause_reg0 Time_out_reg0 Staus_reg0	Transfer data and monitors status.
Interrupt Processing	Interrupt_status_reg0 Interrupt_mask_reg0 Interrupt_enable_reg0 Interrupt_disable_reg0	Enable/disable interrupt detection, mask interrupt set to the interrupt controller, read raw interrupt status.

20.4 System Functions

20.4.1 Clocks

The controller, I/O interface and APB interconnect are driven by CPU_1X clock. This clock comes from the PS clock subsystem.

PS Clock Subsystem

CPU_1x Clock

Refer to section 25.2 CPU Clock, for general clock programming information.

Operating Restrictions

The clock operating restrictions are described in section 20.1.3 Notices.

20.4.2 Reset Controller

The controller reset bits are generated by the PS, see Chapter 26, Reset System.

Example: Controller Reset

1. **Assert Controller Reset**: Set slcr.I2C_RST_CTRL[I2Cx_CPU1X_RST] bit = 1.
2. **De-assert Controller Reset**: Clear slcr.I2C_RST_CTRL[I2Cx_CPU1X_RST] bit = 0.

20.5 I/O Interface

20.5.1 Pin Programming

The I2C SCL and SDA signals can be routed to one of many sets of MIO pins or to the EMIO interface. All of the I2C signals are listed in Table 20-2. The routing of the SCL and SDA signals are described in section 2.5 PS-PL MIO-EMIO Signals and Interfaces.

Example: Route I2C 0 SCL and SDA Signals to MIO Pins 50, 51

In this example, the I2C 0 SCL and SDA signals are routed through MIO pins 50 and 51. Many other pin options are possible.

1. **Configure MIO pin 50 for the SCL signal**. Write `0x0000_1240` to the slcr.MIO_PIN_50 register:
   a. Route I2C 0 SCL signal to pin 50.
   b. 3-state controlled by I2C (set TRI_ENABLE = `0`).
   c. LVCMOS18 (refer to the register definition for other voltage options).
   d. Slow CMOS edge (benign setting).
   e. Enable internal pull-up resistor.
   f. Disable HSTL receiver.
2. **Configure MIO pin 51 for the SDA signal**. Write `0x0000_1240` to the slcr.MIO_PIN_51 register:
   a. Route I2C 0 SDA signal to pin 51.
   b. 3-state controlled by the I2C (set TRI_ENABLE = `0`).
   c. LVCMOS18 (refer to the register definition for other voltage options).
   d. Slow CMOS drive edge.
   e. Enable internal pull-up resistor.
   f. Disable HSTL receiver.

20.5.2 MIO-EMIO Interfaces

Table 20-4 identifies the interface signals to the I2C controller. The MIO pins and any restrictions based on device version are shown in the MIO table in section 2.5.4 MIO-at-a-Glance Table.

Table 20-4: **I2C MIO Pins and EMIO Signals**

I2C Interface	Default Controller Input Value	MIO Pins		EMIO Signals	
		Numbers	I/O	Name	I/O
I2C 0, Serial Clock	0	10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50	IO	EMIOI2C0SCLI	I
				EMIOI2C0SCLO	O
				EMIOI2C0SCLTN	O

Table 20-4: **I2C MIO Pins and EMIO Signals *(Cont'd)***

I2C Interface	Default Controller Input Value	MIO Pins		EMIO Signals	
		Numbers	I/O	Name	I/O
I2C 0, Serial Data	~	11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51	IO	EMIOI2C0SDAI	I
				EMIOI2C0SDAO	O
				EMIOI2C0SDATN	O
I2C 1, Serial Clock	0	12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52	IO	EMIOI2C1SCLI	I
				EMIOI2C1SCLO	O
				EMIOI2C1SCLTN	O
I2C 1, Serial Data	~	13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53	IO	EMIOI2C1SDAI	I
				EMIOI2C1SDAO	O
				EMIOI2C1SDATN	O