AXI IIC Bus Interface v2.1

LogiCORE IP Product Guide

PG090 (v2.1) December 18, 2024

AMD Adaptive Computing is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

Table of Contents

Chapter 1

Introduction

The AMD LogiCORE™ AXI IIC core connects to the AMBA® AXI specification and provides a low-speed, two-wire, serial bus interface to a large number of popular devices. This product specification defines the architecture, hardware (signal) interface, software (register) interface, and parameterization options for the AXI IIC core.

Features

- Compliant to industry standard I^2C protocol
- Register access through AXI4-Lite interface
- Master or slave operation
- Multi-master operation
- Software selectable acknowledge bit
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt with automatic mode switching from master to slave
- START and STOP signal generation and detection
- Repeated START signal generation
- Acknowledge bit generation and detection
- Bus busy detection
- Fast-Mode Plus 1 MHz, Fast Mode 400 kHz, or Standard Mode 100 kHz operation
- 7-bit or 10-bit addressing
- General call enable or disable
- Transmit and receive FIFOs – 16 bytes deep
- Throttling
- General purpose output, 1-bit to 8 bits wide
- Dynamic Start and Stop generation
- Filtering on the `scl` and `sda` signals to eliminate spurious pulses

IP Facts

AMD LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	AMD UltraScale+™ Families, AMD UltraScale™ Architecture, AMD Zynq™ 7000 SoC, 7 series, AMD Versal™ Adaptive SoC
Supported User Interfaces	AXI4-Lite
Resources	See Resource Use
Provided with Core	
Design Files	VHDL
Example Design	VHDL
Test Bench	VHDL
Constraints File	XDC delivered with IP generation.
Simulation Model	None
Supported S/W Driver[2]	Standalone and Linux
Tested Design Flows[3]	
Design Entry	AMD Vivado™ Design Suite
Simulation	For supported simulators, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 54435
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Support web page	

Notes:

1. For a complete list of supported devices, see the AMD Vivado™ IP catalog.
2. Standalone driver details can be found in the SDK directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available fromthe Wiki page.
3. For the supported versions of third-party tools, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

Chapter 2

Overview

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the Design Hubs page. You can also use the Design Flow Assistant to better understand the design flows and find content that is specific to your intended design needs. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Clocking
  - Resets
  - Chapter 4: Designing with the Core
  - Customizing and Generating the Core

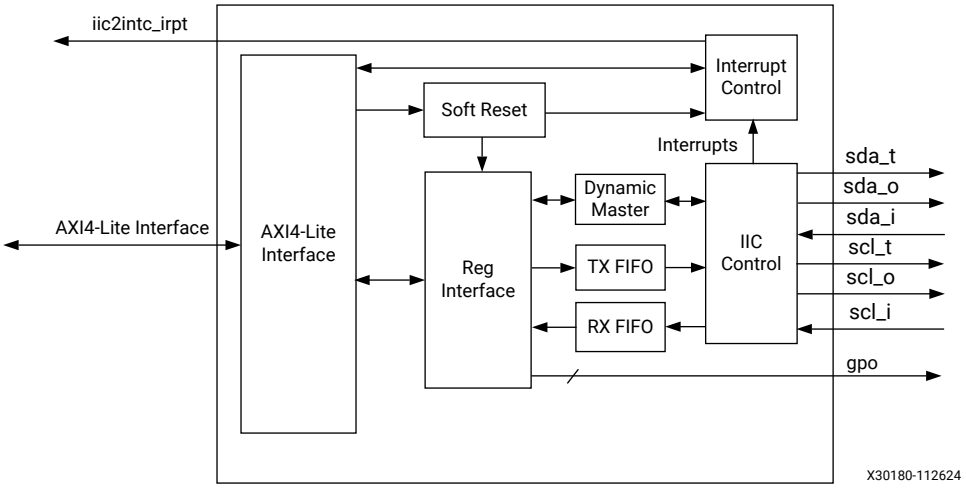
Functional Description

The AXI IIC core provides the transaction interface to the AXI4-Lite interface. This core does not provide explicit electrical connectivity to the IICbus. The design is expected to include bidirectional I/O buffers that implement open collector drivers for the `sda` and `scl` signals. You must also provide external pull-up devices to properly hold the bus at the logic 1 state when the driver is released.

TIP: *Pay attention to the Philips specification when setting the values of the pull-up devices (typically resistors). The values must meet the Philips specification, FPGA maximum ratings, and ratings of any devices on the bus itself.*

The following figure illustrates the top-level block diagram for the AXI IIC bus interface. The modules are described in the sections that follow.

Figure 1: **AXI IIC Bus Interface Top-Level Block Diagram**



- **AXI4-Lite Interface:** This module implements a 32-bit AXI4-Lite Slave interface for accessing AXI IIC registers. For additional details about the AXI4-Lite slave interface, see the *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155).
- **Interrupt Control:** This module gets the interrupt status from the AXI IIC and generates an interrupt to the host.
- **Registers Interface:** This module contains Control and Status registers. It also provides an option to access TX FIFO and RX FIFO. Registers are accessed through the AXI4-Lite interface.
- **TX and RX FIFO:** These FIFOs are used to store data before it is transmitted on the bus or sent to processor.
- **Dynamic Master:** This module controls the mode of the IIC block dynamically. This block works when start bit and a stop bit are written in the transmit FIFO.
- **Soft Reset:** This module allows you to reset the block using software.
- **IIC Control:** This module contains the state machine that controls the IIC interface. It interfaces with the Dynamic Master block to configure the core as Master or Slave.
- **Interrupt Control:** This block generates interrupts for various conditions based on the Interrupt Enable register settings.
- **Dynamic IIC Controller Logic:** The dynamic controller logic provides an interface to the AXI IIC controller that is simple to use. The dynamic logic supports only master mode and 7-bit addressing.

Multi-Master Operation

The AXI IIC core only participates in multi-master arbitration when the bus is initially free and the attempt is made. After the module issues the START, other masters can participate in addressing and the AXI IIC correctly relinquishes the bus if the requested address of the other master is lower than the address driven by AXI IIC. However, if the bus is not free, as indicated by `sda` being Low and `scl` being High (the START has occurred), when the request to acquire the bus is made, then the AXI IIC waits until the next bus free opportunity to arbitrate.

Signal Filtering

The Philips I^2C-bus specification indicates that 0 to 50 ns of pulse rejection can be applied when operating in Fast Mode (>100 kHz). You can specify the maximum amount allowed by the specification or more through the filtering parameters scl Inertial delay and sda Inertial delay. These parameters specify the amount of delay in clock cycles.

Some designs might not require any filtering and others (even those operating < 100 kHz) might require the maximum amount—and possibly more. It depends on many factors beyond the control of the core itself. It might be necessary for you to experiment to determine the optimum amount. If more than 50 ns of pulse rejection is required, you might need to more tightly constrain rise or fall times beyond what is required by the Philips specification to accommodate the additional delay occurring because of the filter operation.

Applications

This core is useful for interfacing to one or more I^2C-compliant devices (for example, System Management devices, Power Management devices, and Video and display devices).

Licensing and Ordering

This AMD LogiCORE™ IP module is provided at no additional cost with the AMD Vivado™ Design Suite under the terms of the End User License.

Information about other AMD LogiCORE™ IP modules is available at the Intellectual Property page. For information about pricing and availability of other AMD LogiCORE IP modules and tools, contact your local sales representative.

Product Specification

Standards

The AXI IIC core follows the Philips I^2C-bus Specification, version 2.1, January 2000, except for the following areas:

- High-speed mode (Hs-mode) is not currently supported by the AXI IIC core.
- 3-state buffers are used to perform the wired-AND function inherent in this bus structure.
- The AMD FPGA ratings must not be exceeded when interconnecting the AXI IIC core to other devices.

Performance

The AXI IIC core is characterized as per the benchmarking methodology described in the *Vivado Design Suite User Guide: Designing with IP* (UG896).

***Note*:**

Performance numbers for AMD UltraScale™ architecture and AMD Zynq™ 7000 SoC devices are expected to be similar to 7 series device numbers.

Table 1: **Maximum Frequencies**

Family	Speed Grade	F_{Max} (MHz)
		AXI4-Lite
Virtex 7	–1	180
Kintex 7		180
Artix 7		120
Virtex 7	–2	200
Kintex 7		200
Artix 7		140

Table 1: **Maximum Frequencies** *(cont'd)*

Family	Speed Grade	F_{Max} (MHz)
		AXI4-Lite
Virtex 7	–3	220
Kintex 7		220
Artix 7		160

Resource Use

Resource requirements for the AXI IIC core have been estimated for 7 series and Zynq 7000 devices (see the following table). These values were generated using the Vivado Design Suite.

***Note*:** Resources numbers for UltraScale architecture and Zynq 7000 devices are expected to be similar to 7 series device numbers.

Table 2: **Device Utilization – 7 Series and Zynq 7000 AP Devices**

Parameter Values		Device Resources		
scl Clock Frequency (in kHz)	Address Mode (in bits)	Registers	Slices	LUTs
100	7	231	141	317
100	10	238	141	314
400	7	231	141	317
400	10	238	141	314

I/O Signals

The AXI IIC core I/O signals are described in the following table.

Table 3: **I/O Signal Descriptions**

Signal Name	Interface	I/O	Initial State	Description
System Signals				
s_axi_aclk	System	I	–	AXI Clock
s_axi_aresetn	System	I	–	AXI Reset, active-Low.
iic2intc_irpt	System	O	0x0	System Interrupt output.
s_axi*	S_AXI	I	–	See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) for a description of AXI4 signals.

Table 3: **I/O Signal Descriptions** *(cont'd)*

Signal Name	Interface	I/O	Initial State	Description
IIC Signals				
sda_i	IIC	I	–	IIC Serial Data Input from 3-state buffer.
sda_o	IIC	O	0x0	IIC Serial Data Output to 3-state buffer.
sda_t	IIC	O	0x0	IIC Serial Data Output Enable to 3-state buffer.1
scl_i	IIC	I	–	IIC Serial Clock Input from 3-state buffer.
scl_o	IIC	O	0x0	IIC Serial Clock Output to 3-state buffer.
scl_t	IIC	O	0x0	IIC Serial Clock Output Enable to 3-state buffer.[1]
gpo	IIC	O	0x0	Configurable General Purpose Outputs.

Notes:

1. The sda_t and scl_t signals are the 3-state enable signals that control the data direction for the sda and scl signals.

Register Space

The following table specifies the name, address, and accessibility of each firmware addressable register from the three classes of registers within the AXI IIC core.

***Note*:** The AXI4-Lite write access register is updated by the 32-bit AXI Write Data (`*_wdata`) signal, and is not impacted by the AXI Write Data Strobe (`*_wstrb`) signal. For a Write access, both the AXI Write Address Valid (`*_awvalid`) and AXI Write Data Valid (`*_wvalid`) signals should be asserted together.

Table 4: **AXI IIC Core Register Map**

Address Space Offset[1]	Register Name	Description
01Ch	GIE	Global Interrupt Enable Register
020h	ISR	Interrupt Status Register
028h	IER	Interrupt Enable Register
040h	SOFTR	Soft Reset Register
100h	CR	Control Register
104h	SR	Status Register
108h	TX_FIFO	Transmit FIFO Register
10Ch	RX_FIFO	Receive FIFO Register
110h	ADR	Slave Address Register
114h	TX_FIFO_OCY	Transmit FIFO Occupancy Register
118h	RX_FIFO_OCY	Receive FIFO Occupancy Register
11Ch	TEN_ADR	Slave Ten Bit Address Register
120h	RX_FIFO_PIRQ	Receive FIFO Programmable Depth Interrupt Register
124h	GPO	General Purpose Output Register

Table 4: **AXI IIC Core Register Map** *(cont'd)*

Address Space Offset[1]	Register Name	Description
128h	TSUSTA	Timing Parameter Register
12Ch	TSUSTO	Timing Parameter Register
130h	THDSTA	Timing Parameter Register
134h	TSUDAT	Timing Parameter Register
138h	TBUF	Timing Parameter Register
13Ch	THIGH	Timing Parameter Register
140h	TLOW	Timing Parameter Register
144h	THDDAT	Timing Parameter Register

Notes:

1. Address Space Offset is relative to C_BASEADDR assignment.

Global Interrupt Enable (GIE)

The Global Interrupt Enable register, illustrated in the following figure and table, has a single defined bit, in the most significant bit that is used to globally enable the final interrupt (coalesced from the ISR) out to the system.

Figure 2: **Global Interrupt Enable (GIE) Register**

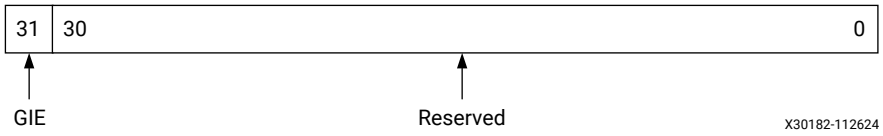


Table 5: **Global Interrupt Enable (GIE) Register (01Ch)**

Bits	Field Name	Default Value	Access Type	Description
31	GIE	0	R/W	Global Interrupt Enable 0 = All Interrupts disabled; no interrupt (even if unmasked in IER) possible from AXI IIC core 1 = Unmasked AXI IIC core interrupts are passed to processor
30:0	Reserved	N/A	N/A	Reserved

Interrupt Status Register (ISR)

Firmware uses the ISR, illustrated in the following figure, to determine which interrupt events from the AXI IIC core need servicing. The register uses a toggle on write method to allow firmware to clear selected interrupts by writing a 1 to the desired interrupt bit field position. This mechanism avoids the requirement on the User Interrupt Service routine to perform a Read/Modify/Write operation to clear a single bit within the register. An interrupt value of 1 means the interrupt has occurred. A value of 0 means that no interrupt occurred or it was cleared.

The following table and figure illustrate the interrupt to bit field mappings of the IPIER (interrupt enable) and IPISR (interrupt status) registers. The number in the parenthesis is the interrupt bit number.

Figure 3: **Interrupt Enable Register (IER) and Interrupt Status Register (ISR)**

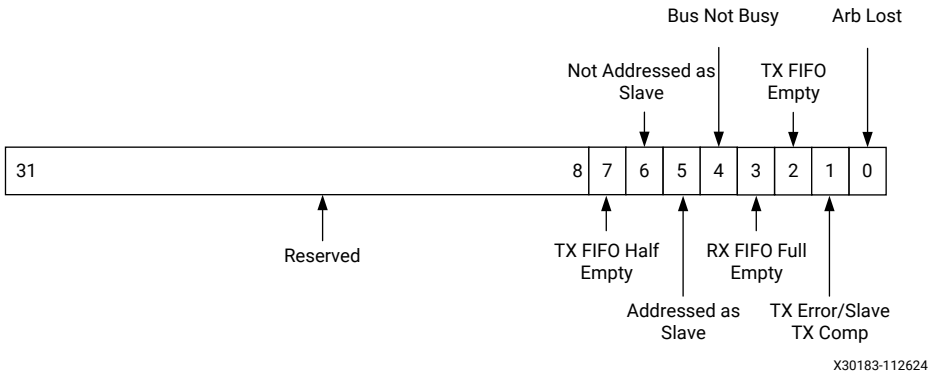


Table 6: **Interrupt Status Register (020h)**

Bits	Field Name	Default Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7	int(7)	1	Read/Toggle on Write	Interrupt(7) — Transmit FIFO Half Empty.
6	int(6)	1	Read/Toggle on Write	Interrupt(6) —Not Addressed As Slave.
5	int(5)	0	Read/Toggle on Write	Interrupt(5) — Addressed As Slave.
4	int(4)	1	Read/Toggle on Write	Interrupt(4) — IIC Bus is Not Busy.
3	int(3)	0	Read/Toggle on Write	Interrupt(3) — Receive FIFO Full.
2	int(2)	0	Read/Toggle on Write	Interrupt(2) — Transmit FIFO Empty.
1	int(1)	0	Read/Toggle on Write	Interrupt(1) — Transmit Error/Slave Transmit Complete.
0	int(0)	0	Read/Toggle on Write	Interrupt(0) — Arbitration Lost.

Interrupt(0): Arbitration Lost

Interrupt(0) is the Arbitration Lost interrupt. This interrupt is set when arbitration for the IIC bus is lost. Firmware must respond by first clearing the Control Register (CR) MSMS bit and then clearing this interrupt by writing a 1 to the Interrupt Status Register (ISR) INT(0) bit to toggle it. See also the TX_FIFO reset bit in the Control Register (CR).

Interrupt(1): Transmit Error/Slave Transmit Complete

Four possible events can cause this interrupt:

1. AXI IIC core operating as a master transmitter: Interrupt(1) implies an error. There are two possibilities:
   a. Either no slave was present at the transmitted address in which case the master transmitter recognizes a NOT ACKNOWLEDGE.
   b. The slave receiver issued a NOT ACKNOWLEDGE to signal that it is not accepting anymore data.

   In either case the MSMS bit in the Control Register (CR) transitions from 1 to 0 causing the AXI IIC core to initiate a stop condition, which implies that the bus is not busy.
2. AXI IIC core operating as a master receiver: Interrupt(1) implies a transmit complete. This interrupt is caused by setting the TXAK bit in the Control Register (CR) to 1 to indicate to the slave transmitter that the last byte has been transmitted. This bit is set soon after the NACK condition occurs on the line.
3. AXI IIC core operating as a slave transmitter: Interrupt(1) implies a transmit complete. This interrupt is caused by the master device to indicate to the IIC that the last byte has been transmitted.
4. AXI IIC core operating as a slave receiver: Interrupt(1) implies an error. This interrupt is caused by the IIC (setting Control Register (CR) field TXAK to 1).

Firmware must clear this interrupt by writing a 1 to the Interrupt Status Register (ISR), INT(1) bit to toggle it.

This interrupt occurs before INT(4), if INT(4) is also enabled. (The stop occurs later.)

Interrupt(2): Transmit FIFO Empty

The controller raises (sets) the interrupt flag and keeps it raised while a transmit throttle condition exists. After the flag has been raised and the transmit throttle condition is removed, then (and only then can) firmware lowers (clear) the flag by writing a 1 to the Interrupt Status Register (ISR), INT(2) bit to toggle the flag state. See Throttling for information on actions that must be taken to clear a transmit throttle condition. The usual cause for a transmit throttle condition is the transmit FIFO going empty.

IMPORTANT! *This interrupt is asserted when the IIC transmit pipeline is fully empty. That is, the TX FIFO is empty and the last byte of the Data has been completely transferred out.*

Interrupt(3): Receive FIFO Full

This interrupt is set when the Receive FIFO Programmable Depth Interrupt Register (RX_FIFO_PIRQ) is equal to the Receive FIFO Occupancy Register (RX_FIFO_OCY). Clearing this interrupt requires that the data receive FIFO be read. This bit is not set at the same time as Transmit Complete bit.

Interrupt(4): IIC Bus is Not Busy

Interrupt(4) is set when the IIC bus is not busy. The condition remains set as long as the bus is not busy and cannot be cleared while the condition is TRUE. Firmware must verify that the SR(BB) is asserted, indicating bus busy, before attempting to reset this interrupt bit.

A master that loses arbitration that wants to get back on this bus should immediately clear this bit.

If necessary, the slave should clear this bit after getting the AAS interrupt to know when the bus is not busy occurs. (A master could talk to several slaves before relinquishing the bus.)

Interrupt(5): Addressed as Slave

This interrupt is set when the AXI IIC core is addressed as a slave.

Interrupt(6): Not Addressed as Slave

This interrupt allows the detection of the end of receive data for a slave receiver when there has been no stop condition (repeated start). The interrupt occurs when a start condition followed by a non-matching slave address is detected. This interrupt is set when the AXI IIC core is not addressed as a slave.

Interrupt(7): Transmit FIFO Half Empty

This interrupt is set while the MSB of the TX_FIFO_OCY = 0.

Interrupt Enable Register (IER)

The Interrupt Enable register is described in the following table. Firmware uses the fields of this register to enable or disable interrupts needed to manage either the Standard Controller Logic Flow or the Dynamic Controller Logic Flow.

Table 7: **Interrupt Enable Register [1](028h)**

Bits	Field Name	Default Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7	int(7)	0	R/W	Interrupt(7) — Transmit FIFO Half Empty.
6	int(6)	0	R/W	Interrupt(6) — Not Addressed As Slave.
5	int(5)	0	R/W	Interrupt(5) — Addressed As Slave.
4	int(4)	0	R/W	Interrupt(4) — IIC Bus is Not Busy.
3	int(3)	0	R/W	Interrupt(3) — Receive FIFO Full.
2	int(2)	0	R/W	Interrupt(2) — Transmit FIFO Empty.
1	int(1)	0	R/W	Interrupt(1) — Transmit Error/Slave Transmit Complete.
0	int(0)	0	R/W	Interrupt(0) — Arbitration Lost.

Notes:

1. In any given bit position, 1 = Interrupt enabled, 0 = Interrupt masked.

Soft Reset Register (SOFTR)

Firmware can write to the SOFTR to initialize all of the AXI IIC core registers to their default states. To accomplish this, firmware must write the reset key (RKEY) value of 0xA to the least significant nibble of the 32-bit word. After recognizing a write of 0xA the proc_common soft_reset module issues a pulse four clocks long to reset the AXI IIC core. At the end of the pulse the SOFTR acknowledges the AXI transaction. That prevents anything further from happening while the reset occurs.

Writing any value to Bits[3:0] other than 0xA results in an AXI transaction acknowledge with an error status. The register is not readable.

Applying soft reset to the AXI IIC core also clears the bus busy (BB) status (Bit[2] of SR). When a read is issued to the SR immediately after reset, SR might not give the correct status of the IIC bus if the IIC bus is locked by another IIC device. Therefore, you should reset the external device after applying the soft reset to the AXI IIC core and before using the IIC bus again.

The SOFTR bit fields are shown in the following figure and described in the following table.

Figure 4: **Soft Reset Register (SOFTR)**

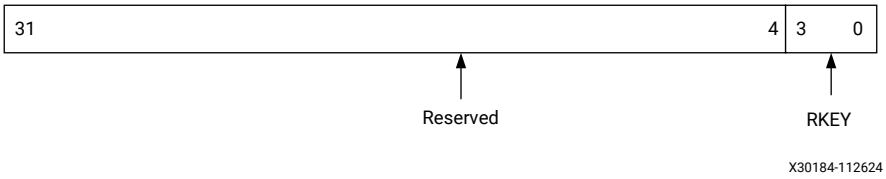


Table 8: **Soft Reset Register (040h)**

Bits	Field Name	Default Value	Access Type	Description
31:4	Reserved	N/A	N/A	Reserved
3:0	RKEY	N/A	W	Reset Key. Firmware must write a value of 0xA to this field to cause a soft reset of the Interrupt registers of AXI IIC controller. Writing any other value results in an AXI transaction acknowledgment with SLVERR and no reset occurs.

Control Register (CR)

Writing to the Control register configures the AXI IIC core operation mode and simultaneously allows for IIC transactions to be initiated.

Prior to setting master slave mode select (MSMS) to a 1, the TX_FIFO should contain the address of the AXI IIC core device. All the CR bits can be set at the same time as setting MSMS to a 1 to initiates a bus transaction.

When initiating a repeated start condition, the transmit FIFO must be empty. First, set the repeated start bit to a 1 and then write the address of the AXI IIC core device to the transmit FIFO. The rest of the FIFO can be filled with data, if required.

The EN field provides a way for the device driver to initialize interrupts prior to enabling the device to send/receive data.

The AXI IIC core Control register is shown in the following figure and described in the following table.

Figure 5: **Control (CR) Register**

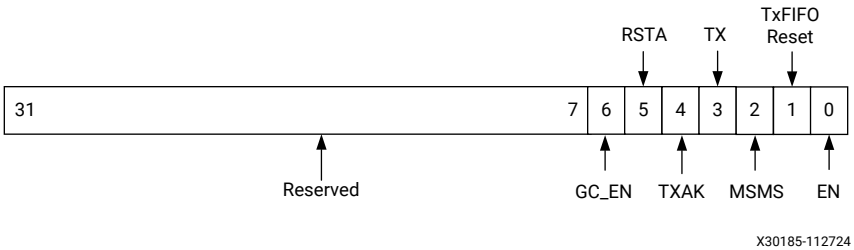


Table 9: **Control Register (100h)**

Bits	Field Name	Default Value	Access Type	Description
31:7	Reserved	N/A	N/A	Reserved
6	GC_EN	0	R/W	General Call Enable. Setting this bit High allows the AXI IIC to respond to a general call address. 0 = General Call Disabled 1 = General Call Enabled

Table 9: **Control Register (100h)** *(cont'd)*

Bits	Field Name	Default Value	Access Type	Description
5	RSTA	0	R/W	Repeated Start. Writing a 1 to this bit generates a repeated START condition on the bus if the AXI IIC bus interface is the current bus master. Attempting a repeated START at the wrong time, if the bus is owned by another master, results in a loss of arbitration. This bit is reset when the repeated start occurs. This bit must be set prior to writing the new address to the TX_FIFO or DTR.
4	TXAK	0	R/W	Transmit Acknowledge Enable. This bit specifies the value driven onto the sda line during acknowledge cycles for both master and slave receivers. 0 = ACK bit = 0 – acknowledge 1 = ACK bit = 1 – not-acknowledge Because master receiver indicates the end of data reception by not acknowledging the last byte of the transfer, this bit is used to end a master receiver transfer. As a slave, this bit must be set prior to receiving the byte to signal a not-acknowledge.
3	TX	0	R/W	Transmit/Receive Mode Select. This bit selects the direction of master/slave transfers. 0 = selects an AXI IIC receive 1 = selects an AXI IIC transmit This bit does not control the Read/Write bit that is sent on the bus with the address. The Read/Write bit that is sent with an address must be the LSB of the address written into the TX_FIFO.
2	MSMS	0	R/W	Master/Slave Mode Select. When this bit is changed from 0 to 1, the AXI IIC bus interface generates a START condition in master mode. When this bit is cleared, a STOP condition is generated and the AXI IIC bus interface switches to slave mode. When this bit is cleared by the hardware, because arbitration for the bus has been lost, a STOP condition is not generated (see also Interrupt(0): Arbitration Lost).
1	TX_FIFO Reset	0	R/W	Transmit FIFO Reset. This bit must be set to flush the FIFO if either (a) arbitration is lost or (b) if a transmit error occurs. 0 = transmit FIFO normal operation 1 = resets the transmit FIFO
0	EN	0	R/W	AXI IIC Enable. This bit must be set before any other CR bits have any effect. 0 = resets and disables the AXI IIC controller but not the registers or FIFOs 1 = enables the AXI IIC controller

Status Register (SR)

This register contains the status of the AXI IIC core interface. All bits are cleared upon reset.

The read-only SR is shown in the following figure and described in the following table.

Figure 6: **Status Register (SR)**

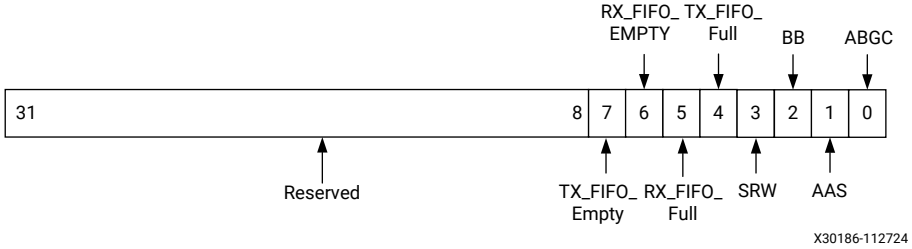


Table 10: **Status Register (104h)**

Bits	Field Name	Default Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7	TX_FIFO_Empty	1	R	Transmit FIFO empty. This bit is set High when the transmit FIFO is empty. Note: This bit goes High as soon as the TX FIFO becomes empty. At this moment, the last byte of Data might still be in output pipeline or might be partially transferred.
6	RX_FIFO_Empty	1	R	Receive FIFO empty. This is set High when the receive FIFO is empty.
5	RX_FIFO_Full	0	R	Receive FIFO full. This bit is set High when the receive FIFO is full. This bit is set only when all 16 locations in the FIFO are full, regardless of the compare value field of the RX_FIFO_PIRQ register.
4	TX_FIFO_Full	0	R	Transmit FIFO full. This bit is set High when the transmit FIFO is full.
3	SRW	0	R	Slave Read/Write. When the IIC bus interface has been addressed as a slave (AAS is set), this bit indicates the value of the read/write bit sent by the master. This bit is only valid when a complete transfer has occurred and no other transfers have been initiated. 0 = indicates master writing to slave 1 = indicates master reading from slave
2	BB	0	R	Bus Busy. This bit indicates the status of the IIC bus. This bit is set when a START condition is detected and cleared when a STOP condition is detected. 0 = indicates the bus is idle 1 = indicates the bus is busy
1	AAS	0	R	Addressed as Slave. When the address on the IIC bus matches the slave address in the Address register (ADR), the IIC bus interface is being addressed as a slave and switches to slave mode. If 10-bit addressing is selected this device only responds to a 10-bit address or general call if enabled. This bit is cleared when a stop condition is detected or a repeated start occurs. 0 = indicates not being addressed as a slave 1 = indicates being addressed as a slave
0	ABGC	0	R	Addressed By a General Call. This bit is set to 1 when another master has issued a general call and the general call enable bit is set to 1, CR(6) = 1.

Transmit FIFO (TX_FIFO)

This is the keyhole address for the FIFO that contains data to be transmitted on the IIC bus. In transmit mode, data written into this FIFO is output on the IIC bus. Attempting to write to a full FIFO is not recommended and results in that data byte being lost. Firmware must clear the FIFO prior to use in anticipation of it not being empty possibly due to abnormal IIC protocol, abnormal terminations or other normal controller actions such as dynamic mode reads.

The transmit FIFO (TX_FIFO) is shown in the following figure and described in the following table.

Figure 7: **Transmit FIFO (TX_FIFO)**

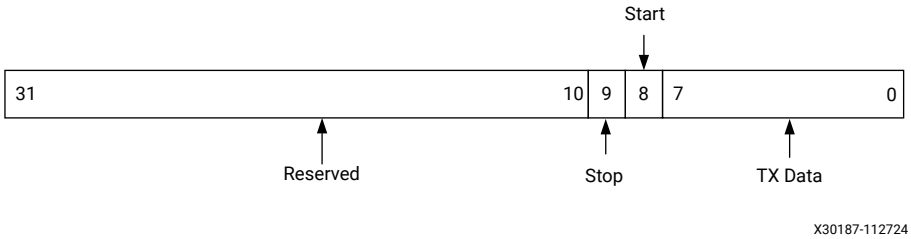


Table 11: **AXI IIC Transmit FIFO (108h)**

Bits	Field Name	Default Value	Access Type	Description
31:10	Reserved	N/A	N/A	Reserved
9	Stop	0	W	Stop. The dynamic stop bit can be used to send an IIC stop sequence on the IIC bus after the last byte has been transmitted or received.[2]
8	Start	0 [1]	W	Start. The dynamic start bit can be used to send a start or repeated start sequence on the IIC bus. A start sequence is generated if the MSMS = 0, a repeated start sequence is generated if the MSMS = 1.[2]
7:0	D7 to D0	Indeterminate [3]	W	AXI IIC Transmit Data. If the dynamic stop bit is used and the AXI IIC is a master receiver, the value is the number of bytes to receive.[3]

Notes:

1. The value that was available before the reset occurred still appears on the FIFO outputs.
2. A full description for the use of the dynamic stop and start bits is contained in the Dynamic Controller Logic Flow section. These bits are not readable.
3. Only Bits[7:0] can be read back.

Receive FIFO (RX_FIFO)

This FIFO contains the data received from the IIC bus. The received IIC data is placed in this FIFO after each complete transfer. The RX_FIFO_OCY must be equal to the RX_FIFO_PIRQ before throttling occurs. The receive FIFO is read only. Reading this FIFO when it is empty results in indeterminate data being read.

The receive FIFO is shown in the following figure and described in the following table.

Figure 8: **Receive FIFO (RX_FIFO)**

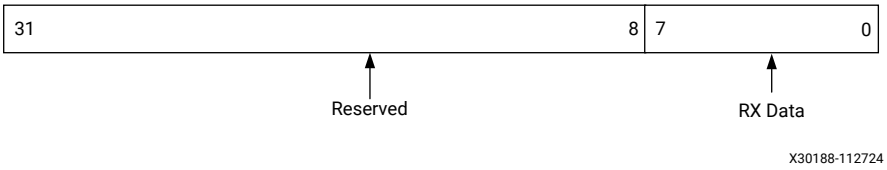


Table 12: **Receive FIFO (10Ch)**

Bits	Field Name	Default Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7:0	D7 to D0	Indeterminate[1]	R	IIC Receive Data

Notes:

1. The value that was available before the reset occurred appears on the FIFO outputs.

Slave Address Register (ADR)

You program the ADR register (and possibly the TEN_ADR register) to set the address at which the slave acknowledges an address transfer operation from the bus master. The slave address field of the ADR register contains all seven bits of the 7-bit address or the least significant seven bits of the 10-bit address the AXI IIC bus interface recognizes when operating as a slave.

The field layout of the register is shown in the following figure and described in the following table.

Figure 9: **Slave Address Register (ADR)**

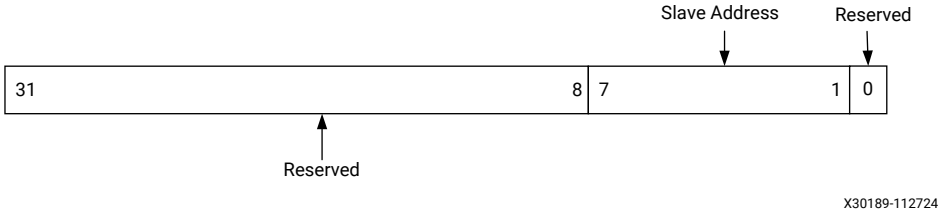


Table 13: **Slave Address Register (110h)**

Bits	Field Name	Default Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7:1	Slave Address	0	R/W	Address used by the IIC bus interface when in slave mode.
0	Reserved	N/A	N/A	Reserved

Slave 10-Bit Address Register (TEN_ADR)

Program the TEN_ADR register (and possibly the ADR register) to set the address at which the slave acknowledges and addresses transfer operation from the bus master. The slave address field of the TEN_ADR register contains the most significant three bits of the 10-bit address that the AXI IIC bus interface recognizes when operating as a 10-bit addressable slave. This register exists only if it configures the AXI IIC for 10-bit addressing by setting generic parameter Address mode to 10 bits in Vivado Integrated Design Environment.

The TEN_ADR register is shown in the following figure and described in the following table.

Figure 10: **10-Bit Address Register (TEN_ADR)**

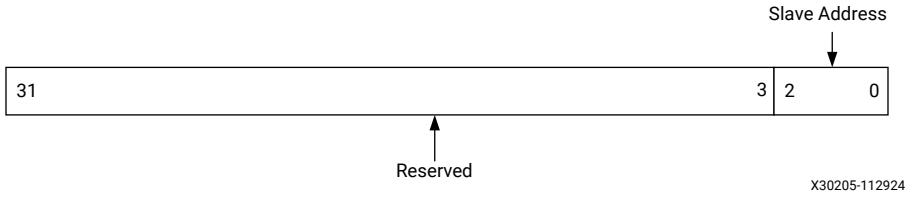


Table 14: **Slave 10-Bit Address Register (11Ch)**

Bits	Field Name	Default Value	Access Type	Description
31:3	Reserved	N/A	N/A	Reserved
2:0	MSB of Slave Address	0	R/W	Three MSBs of the 10-bit address used by the AXI IIC bus interface when in slave mode.

Transmit FIFO Occupancy Register (TX_FIFO_OCY)

This field contains the occupancy value for the transmit FIFO. The transmit FIFO Empty interrupt conveys that information. The value read is the occupancy value minus one, therefore reading all zeros indicates that the first location is filled and reading all ones implies that all 16 locations are filled. This register should not be used to determine whether TX FIFO is empty or not. Instead, AMD recommends using TX FIFO Empty Interrupt.

The TX_FIFO_OCY register is shown in the following figure and described in the following table.

Figure 11: **Transmit FIFO Occupancy Register (TX_FIFO_OCY)**

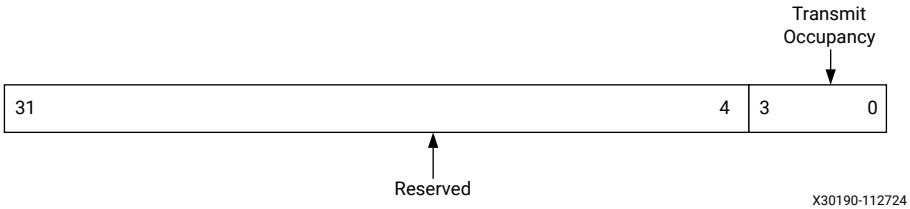


Table 15: **Transmit FIFO Occupancy Register (114h)**

Bits	Field Name	Default Value	Access Type	Description
31:4	Reserved	N/A	N/A	Reserved
3:0	Occupancy Value	0	R	Bit[3] is the MSB. A binary value of 1001 indicates that 10 locations are full in the FIFO.

Receive FIFO Occupancy Register (RX_FIFO_OCY)

This field contains the occupancy value for the receive FIFO. This register is read only. The value read is the occupancy value minus one, therefore reading all 0s implies that the first location is filled and reading all 1s implies that all 16 locations are filled. This register should not be used to determine whether FIFO is empty or not. AMD recommends using RX_FIFO Empty Bit from the Status register.

The RX_FIFO_OCY register is shown in the following figure and described in the following table.

Figure 12: **Receive FIFO Occupancy Register (RX_FIFO_OCY)**

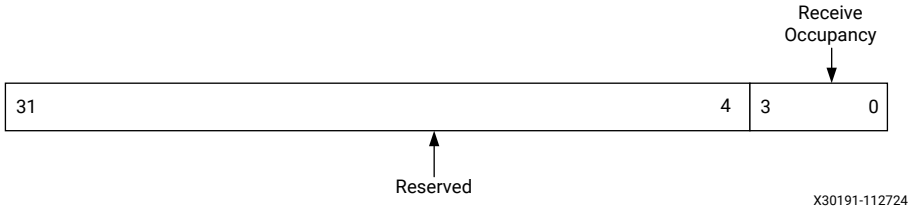


Table 16: **Receive FIFO Occupancy Register (118h)**

Bits	Field Name	Default Value	Access Type	Description
31:4	Reserved	N/A	N/A	Reserved
3:0	Occupancy Value	0	R	Bit[3] is the MSB. A binary value of 1001 implies that 10 locations are full in the FIFO.

Receive FIFO Programmable Depth Interrupt Register (RX_FIFO_PIRQ)

This field contains the value which causes the receive FIFO Interrupt to be set. When this value is equal to the RX_FIFO_OCY value, the receive FIFO interrupt is set and remains set until the equality is no longer true. A read from the receive FIFO causes the IIC receive FIFO interrupt to be cleared. When the RX_FIFO_PIRQ is equal to the RX_FIFO_OCY, throttling also occurs to prevent the transmitter from transmitting.

The read/write RX_FIFO_PIRQ register is shown in the following figure and described in the following table.

Figure 13: **Receive FIFO Programmable Depth Interrupt Register (RX_FIFO_PIRQ)**

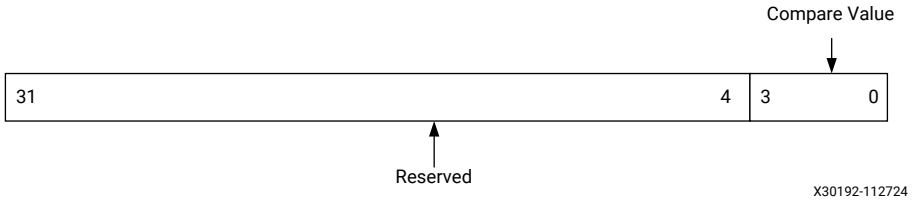


Table 17: **Receive FIFO Programmable Depth Interrupt Register (120h)**

Bits	Field Name	Default Value	Access Type	Description
31:4	Reserved	N/A	N/A	Reserved
3:0	Compare Value	0	R/W	Bit[3] is the MSB. A binary value of 1001 implies that when 10 locations in the receive FIFO are filled, the receive FIFO interrupt is set.

General Purpose Output Register (GPO)

The current value of the general-purpose output field of the GPO register is reflected continuously on the GPO I/O signal of the core. For example, the GPO signal of the core could be used to set an IIC memory device write protect.

If General Purpose Output Width is equal to one, the only bit populated in the register is GPO(0), the LSB. If General Purpose Output Width is equal to 8, then Bits[7:0] in the GPO are populated. Reading unpopulated bits results in indeterminate data.

The GPO register is shown in the following figure and described in the following table.

Figure 14: **General Purpose Output Register (GPO)**

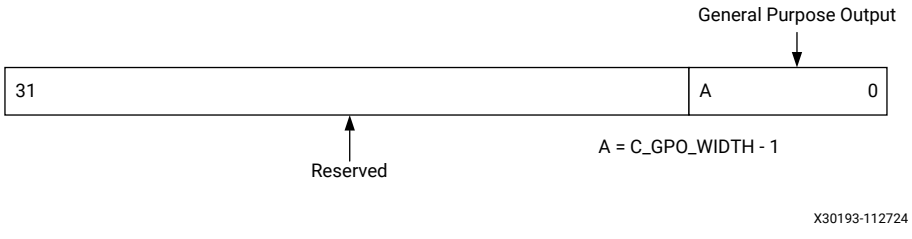


Table 18: **General Purpose Output Register (124h)**

Bits	Field Name	Default Value	Access Type	Description
31:C_GPO_WIDTH	Reserved	N/A	N/A	Reserved

Table 18: **General Purpose Output Register (124h)** *(cont'd)*

Bits	Field Name	Default Value	Access Type	Description
(C_GPO_WIDTH – 1):0	General Purpose Outputs	See Note[1]	R/W	GPO. The LSB (Bit[0]) is the first bit populated.

Notes:

1. This is the value that is specified during IP customization.

Timing Parameter TSUSTA Register (TSUSTA)

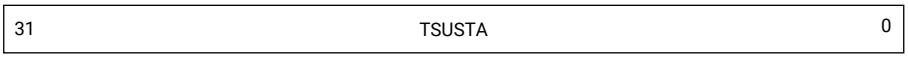
This value determines the setup time (in terms of number of core clock cycles) for a repeated START condition as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value takes into consideration the T_r (Rise Time) output of the I^2C timing specification.

For example, if T_{susta} is 4.7 µs and T_r is 1 µs, then this value should be programmed to get an output of 5.7 µs. If the AXI Clock frequency is 100 MHz (that is, 10 ns period), then this register should be programmed with a resultant of 570.

If the value does not work in your design environment, you can read this register value and update it.

The TSUSTA register is shown in the following figure and described in the following table.

Figure 15: **TSUSTA Register**



X30194-112724

Table 19: **TSUSTA Register (128h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	TSUSTA	See Note[1]	R/W	Setup time for a repeated START condition.

Notes:

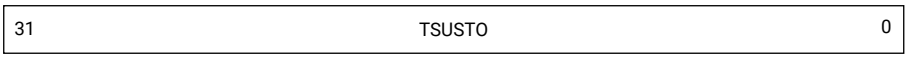
1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter TSUSTO Register (TSUSTO)

This value determines the setup time (in terms of number of core clock cycles) for a repeated STOP condition as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value takes into consideration the T_r (Rise Time) output of the I^2C timing specification. Also, it can be programmed similarly to the TSUSTA register. If the value does not work in your design environment, you can read this register value and update it.

The TSUSTO register is shown in th following figure and described in the following table.

Figure 16: **TSUSTO Register**



X30195-112724

Table 20: **TSUSTO Register (12Ch)**

Bits	Field Name	Default Value	Access Type	Description
31:0	TSUSTO	See Note[1]	R/W	Setup time for a repeated STOP condition.

Notes:

1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter THDSTA Register (THDSTA)

This value determines the hold time (in terms of number of core clock cycles) for a repeated START condition as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value takes into consideration the T_f (Fall Time) output of I^2C timing specification. Also, it can be programmed similarly to the TSUSTA register. If the value does not work in your design environment, you can read this register value and update it.

The THDSTA register is shown in the following figure and described in the following table.

Figure 17: **THDSTA Register**

X30196-112724

Table 21: **THDSTA Register (130h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	THDSTA	See Note[1]	R/W	Hold time for a repeated START condition.

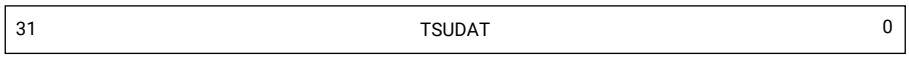
Notes:
1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter TSUDAT Register (TSUDAT)

This value determines the data setup time (in terms of number of core clock cycles) as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value takes into consideration the T_f (Fall Time) output of I^2C timing specification. Also, it can be programmed similarly to the TSUSTA register. If the value does not work in your design environment, you can read this register value and update it.

The TSUDAT register is shown in the following figure and described in the following table.

Figure 18: **TSUDAT Register**



X30197-112724

Table 22: **TSUDAT Register (134h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	TSUDAT	See Note[1]	R/W	Data setup time.

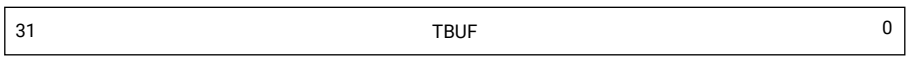
Notes:
1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter TBUF Register (TBUF)

This value determines the bus free time between a STOP and START condition (in terms of number of core clock cycles) as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value takes into consideration the T_f (Fall Time) output of I^2C timing specification. Also, it can be programmed similarly to the TSUSTA register. If the value does not work in your design environment, you can read this register value and update it.

The TBUF register is shown in the following figure and described in the following table.

Figure 19: **TBUF Register**



X30198-112724

Table 23: **TBUF Register (138h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	TBUF	See Note[1]	R/W	Bus free time between a STOP and START condition.

Notes:

1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.
2. This register programming is used as bus free time between STOP and START even in cases where the transactions generated are from different masters.

Timing Parameter THIGH Register (THIGH)

This value determines the High period of the scl clock (in terms of number of core clock cycles) as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value is calculated in the following method:

$$\text{Value} = ((\text{AXI Clock frequency in Hz})/(2 \times \text{IIC frequency in Hz})) - 7 - \text{SCL\_INERTIAL\_DELAY}$$

If the AXI frequency is 25 MHz, IIC frequency is 100 kHz and SCL_INERTIAL_DELAY is 0, then this equals 118. If the value does not work in your design environment, you can read this register value and update it.

The THIGH register is shown in the following figure and described in the following table.

Figure 20: **THIGH Register**

X30199-112724

Table 24: **THIGH Register (13Ch)**

Bits	Field Name	Default Value	Access Type	Description
31:0	THIGH	See Note[1]	R/W	High period of the scl clock.

Notes:

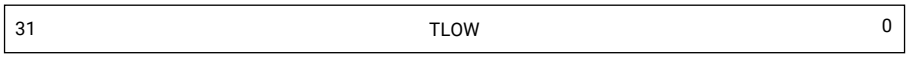
1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter TLOW Register (TLOW)

This value determines the Low period of the scl clock (in terms of number of core clock cycles) as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value is calculated similarly to the THIGH register. If the value does not work in your design environment, you can read this register value and update it.

The TLOW register is shown in the following figure and described in the following table.

Figure 21: **TLOW Register**



X30200-112724

Table 25: **TLOW Register (140h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	THIGH	See Note[1]	R/W	Low period of the scl clock.

Notes:

1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Timing Parameter THDDAT Register (THDDAT)

This value determines the data hold time (in terms of number of core clock cycles) as per the I^2C specification. It contains the default value based on the operating frequency of the core and the configured IIC frequency. This value is programmed similarly to the timing registers. If the value does not work in your design environment, you can read this register value and update it.

The THDDAT register is shown in the following figure and described in the following table.

Figure 22: **THDDAT Register**

X30201-112724

Table 26: **THDDAT Register (144h)**

Bits	Field Name	Default Value	Access Type	Description
31:0	THDDAT	See Note[1]	R/W	Data hold time.

Notes:

1. Depends on the mode of operation: Standard, Fast, or Fast-Mode Plus.

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

IIC Protocol and Electrical Characteristics

To understand and use the register-based software interface in the AXI IIC core, it is helpful to have a basic understanding of the IIC protocol and the electrical characteristics of the bus. For more details and timing diagrams, see the *Philips I^{2}C-bus Specification*.

Protocol for Address and Data Transfer

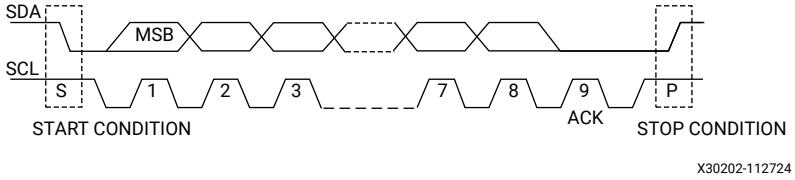
Each device on the bus has a unique 7-bit or 10-bit address, operates both as a transmitter and receiver, and additionally acts as a master or slave. A master device initiates data transfer on the bus and generates the clock signal for that transfer. The slaves respond to the address clocked into them by the master and either accept ("write") data from or provide ("read") data to the master.

The IIC protocol defines an arbitration procedure that ensures that if more than one master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted. The arbitration and clock synchronization procedures defined in the *Philips I^{2}C-bus Specification* are supported by the AXI IIC module.

Data transfers on the AXI IIC bus are initiated with a START condition and are terminated with a STOP condition. After reaching the bus free state, a master signals a START defined by a High-to-Low transition on sda while scl is High. Likewise, the master signals a STOP by a Low-to-High transition on the `sda` line while `scl` is High. Between the START and STOP conditions of the bus, data on the `sda` signal must be stable during the High period of the scl signal and must meet any required setup and hold times during the Low period of the `scl` signal.

The following figure illustrates how the definitions of: (a) the bus free state, and (b) the times when `sda` and `scl` can change relative to each other, ensure that the START and STOP conditions are not confused as data.

Figure 23: **Data Transfer on the IIC Bus**



Each transfer on the IIC bus consists of nine clock pulses on scl to move eight bits of data and one acknowledge bit. Master and slave transmitters send data with the most significant bit first (MSB).

After providing data for the eight clock periods, the master or slave transmitter releases the sda line during the acknowledgement clock period to permit the receiver to transfer a 1-bit acknowledgment.

If a slave-receiver issues a not-acknowledge (by releasing the sda signal during the acknowledgement clock period) this indicates that the slave-receiver was unable to accept the prior eight bits transferred (consisting of address or data bits). After a byte of data is transferred, the slave (receiver | transmitter) has the unique capability to throttle the transfer by keeping the `scl` line in its Low state by actively pulling the `scl` line Low for an arbitrary period of time. This ability allows it time to determine internally what value it should place on the sda line for the acknowledgement.

If the master-receiver signals a not-acknowledge, this indicates to the slave-transmitter that this byte was the last byte of the transfer.

Standard communication on the bus between a master and a slave is composed of four parts:

- START
- Slave Address
- Data Transfer
- STOP

The IIC protocol defines a transfer format for both 7-bit and 10-bit addressing.

- For 7-bit addressing, a slave address is sent after the START condition. This address is seven bits long followed by an eighth bit which is the read/write bit. A High indicates a request for data (read) and a Low indicates a data transmission (write).

  Only the slave with the calling address that matches the address transmitted by the master (that won arbitration) responds by sending back an acknowledge bit by pulling the sda line Low on the ninth clock.

- For 10-bit addressing, two bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition. The first five bits (Bits[7:3]) notify the slave that this is a 10-bit transfer followed by the next two bits (Bits[2:1]), which set the slave address Bits[9:8], and the LSB bit (Bit[8]) is the R/W bit. The second byte transferred sets Bits[7:0] of the slave address.

  After successful slave addressing is achieved, the data transfer proceeds byte-by-byte in the direction as specified by the read/write bit.

The master can terminate the communication by generating a STOP signal to free the bus when the receiver signals a not-acknowledge (signaled by releasing sda during the acknowledgement clock period). However, the master can generate a START signal without generating a STOP signal first. This is called a repeated START. A repeated start allows the master to change the direction of data transfer or address a different slave without giving up the bus.

Electrical Issues

An IIC bus consists of two wires named serial data (`sda`) and serial clock (`scl`), which carry information between the devices connected to the bus. The 400 pF maximum signal load capacitance limits the maximum number of devices connectable to the same bus.

Both `sda` and `scl` transport data bidirectionally between connected devices using wired-AND electrical connectivity. To implement the wired-AND, each device uses an open-collector/open-drain output that only sinks current to ground to pull the signal to logic 0. Electrically that means it must not drive a logic 1 on to either bus signal but can only release or float the output. When no device asserts a logic 0 onto the bus, external pull-up devices (typically resistors) bring the signal state High. This method creates a source of confusion because no device can actually set the state of scl or sda to its High (logic 1) state.

IMPORTANT! *You must pay careful attention to the value of these pull-ups to guarantee that the implementation (consisting of the AXI IIC core, the AMD FPGA, and other devices on the bus) does not violate the IIC timing parameters. Selecting the value of pull-up resistors for a particular application is beyond the scope of the AXI IIC and this document.*

You should consider using the small additional amount of logic necessary for filtering of `scl` and `sda` by specifying non-zero values for the parameters `scl` Inertial delay and `sda` Inertial delay. Reliability of the system can increase substantially.

When all devices on the bus release their drivers and both `sda` and `scl` are High for a specified period of time the bus is considered to be in the bus free state.

Interrupts

The AXI IIC core driver firmware has eight unique interrupt conditions available to manage IIC data transfers. It generates a single `iic2intc_irpt` signal.

Throttling

The *Philips I²C-bus Specification* permits devices to throttle (suspend) data transmission on the bus by holding the scl line Low for an indefinite period of time. The AXI IIC core controller uses this throttling mechanism to prevent either a receive overrun (RX_FIFO full) or a transmit underflow (TX_FIFO empty) by holding the `scl` line Low after the acknowledge bit has been sent.

Throttling is independent of master or slave operation and depends upon transmit or receive operation only. However initiation of stops and repeated starts can only be accomplished at times when throttling is occurring and only the AXI IIC core acting as a master can initiate such actions. When the AXI IIC core is addressed as a slave, the throttling mechanism gives firmware time to either gather data for transmission from the TX_FIFO, or find a place to store data received into the RX_FIFO.

Transmit throttling occurs when the transmit FIFO goes empty (except when a stop condition is pending). When one or more bytes are written into the TX_FIFO, the transmit throttle condition is removed. The automatic throttling provides firmware with time to handle the interrupt processing necessary for data transfer without manually having to manage the low-level `scl` signaling details. To prevent the throttle condition from re-appearing when the TX_FIFO goes empty firmware must set up the master to issue a stop condition by resetting the CR (MSMS) bit *while the bus is throttled and prior* to writing the very last byte to be transmitted to the TX_FIFO.

To switch transmission to a new device while throttled a repeated start can be issued. Firmware does this by setting the RSTA bit in the CR, *then* writing the device address into the TX_FIFO. The controller recognizes this sequence, issues the repeated start, retrieves the address byte from the TX_FIFO and outputs it onto the bus. If no more data was placed into the TX_FIFO the controller immediately throttles again.

Receive throttling is done when the RX_FIFO_OCY matches the value set in the RX_FIFO_PIRQ. The throttle condition is momentarily removed when a byte from the receive FIFO is read, thus allowing the transmitter to send the next byte. The slow rate of IIC transmissions should permit firmware to completely empty the receive FIFO prior to the receipt of the next byte but it is not necessary to do so.

When the AXI IIC core is a master and a receive throttle condition exists, the core generates a stop condition if the CR MSMS bit is changed from a 1 to a 0. This allows single byte reads from a slave device. The CR TXAK must be set equal to 1 to not-acknowledge the byte if desired.

When the AXI IIC core is a master, is in a receive throttle condition and the transmit FIFO is empty, setting the repeated start in the CR causes a transmit throttle condition to occur. That would raise the Interrupt(2): Transmit FIFO Empty flag.

Programming Sequence

Standard Controller Logic Flow

This section briefly discusses setting the AXI IIC registers to initiate and complete bus transactions.

IIC Master Transmitter with a Repeated Start

1. Write the IIC device address to the TX_FIFO.
2. Write data to TX_FIFO.
3. Write to CR to set MSMS = 1 and TX = 1.
4. Continue writing data to TX_FIFO.
5. Wait for transmit FIFO empty interrupt. This implies the IIC has throttled the bus.
6. Write to CR to set RSTA = 1.
7. Write IIC device address to TX_FIFO.
8. Write all data except last byte to TX_FIFO.
9. Wait for transmit FIFO empty interrupt. This implies the IIC has throttled the bus.
10. Write to CR to set MSMS = 0. The IIC generates a STOP condition at the end of the last byte.
11. Write last byte of data to TX_FIFO.

IIC Master Receiver with a Repeated Start

1. Write the IIC peripheral device addresses for the first slave device to the TX_FIFO. Write the RX_FIFO_PIRQ to the total message length (call it M) minus two. It is assumed that the message < the maximum FIFO depth of 16 bytes.
2. Set CR MSMS = 1 and CR TX = 0.
3. Wait for the receive FIFO interrupt indicating M – 1 bytes have been received.
4. Set CR TXAK = 1.

   TXAK causes the AXI IIC core to not-acknowledge the next byte received indicating to the slave transmitter that the master receiver accepts no further data. TXAK is set before reading data from the RX_FIFO, because as soon as a read from the RX_FIFO has occurred, the throttle condition is removed and the opportunity to set the bit is lost.
5. Read all M – 1 data bytes from the RX_FIFO. Set the RX_FIFO_PIRQ to 0 so that the last byte, soon to be received, causes the receive FIFO full interrupt to be raised.

6. Clear the receive FIFO full interrupt now because after a single byte is retrieved from the RX_FIFO the throttle condition is removed by the controller and the interrupt flag can be lowered (cleared).
7. Wait for the receive FIFO full interrupt.
8. The controller is throttled again with a full RX_FIFO. Set CR RSTA = 1. Write the peripheral IIC device address for a new (or same) IIC slave to the TX_FIFO.
9. Read the final byte of data (of the first message) from the RX_FIFO.

   This terminates the throttle condition so the receive FIFO full interrupt can be cleared at this time. It also permits the controller to issue the IIC restart and transmit the new slave address available in the TX_FIFO. Also set the Receive FIFO Programmable Depth Interrupt register to be 2 less than the total second message length (call it N) in anticipation of receiving the message of N – 1 bytes from the second slave device.
10. Wait for the receive FIFO full interrupt.
11. Set TXAK = 1. Write the RX_FIFO_PIRQ to 0, read the message from the RX_FIFO and clear the receive FIFO full interrupt.
12. Wait for the receive FIFO full interrupt (signaling the last byte is received).
13. Set MSMS = 0 in anticipation of giving up the bus through generation of an IIC Stop.
14. Read the final data byte of the second message from the RX_FIFO. This clears the throttle condition and makes way for the controller to issue the IIC Stop.

IIC Slave Receiver

1. In the control register, set MSMS to 0.
2. Set the TX field to 0.
3. Set CR EN = 1 to enable the AXI IIC core. If the IIC needs to recognize a general call then set EN = 1 and GC_EN = 1.
4. Write the slave address and R/$\overline{W}$ bit to the ADR.

   In 7-bit mode, a slave address of 0x7F should be written as 0xFE to the ADR. The eighth bit is the Read Not Write bit which is 0 in the case of a master transmit (IE Write) to slave. S0, 111 1111 0 = FE.
5. Write 0x0 to the RX_FIFO_PIRQ Compare Value. That causes an interrupt when one byte of data (not address) has been received. Because the address transmitted on the IIC bus is not stored in the receive FIFO, this interrupt is not caused by receiving either a 7-bit address or a 10-bit address.
6. Wait for addressed as slave interrupt AAS.
7. After an AAS interrupt has occurred, determine if the IIC slave is to receive or transmit data by reading Status register Bit[4].
8. Clear not addressed as slave interrupt NAS.

9. If the IIC is a slave receiver, there are two basic choices for the slave receiver interrupt processing.
    a. Set the Receive FIFO Interrupt register to 0x0 and wait for either a Not Addressed as Slave NAS interrupt (no data was sent) or RX_FIFO_PIRQ interrupt. In this mode, an interrupt occurs for every byte of data received plus a NAS for the end of the transmission.
    b. Set the Receive FIFO Interrupt register to 0xF and wait for either a Not Addressed as Slave NAS (some amount of data less than 16 bytes was set) or RX_FIFO_PIRQ interrupt. In this mode if the RX_FIFO_PIRQ interrupt occurs then 16 bytes of data exists in the FIFO to handle. AMD recommends that the software read the RX_FIFO_OCY, though it is not required. NAS can occur without a RX_FIFO_PIRQ interrupt. That means the RX_FIFO_OCY should be read to indicate how many bytes of data must be handled. In this mode there is one RX_FIFO_PIRQ interrupt for every 16 bytes of data plus a NAS for the end of the transmission. If less than 16 bytes of data is sent, NAS is the only interrupt.
10. In either choice above, clear the active interrupts after the data has been handled and wait for the next interrupt.
11. After the NAS interrupt has been received, handle the data and clear AAS.
12. Wait for the AAS interrupt.

IIC Slave Transmitter

If the AXI IIC is a slave transmitter, the following interrupt processing is available for use:

1. Ensure the Transmit Error/Slave Transmit Complete interrupt is cleared.
2. After the AXI IIC has been addressed as a slave transmitter, the IIC transmits the first byte of data in the transmit FIFO. If no data exists in the transmit FIFO, the IIC throttles the bus until data is written into the transmit FIFO.
3. If the protocol allows knowledge as to how much data the slave must transmit, fill up the FIFO and use the transmit FIFO empty or transmit FIFO half empty interrupts to keep the transmit FIFO full. Wait for the transmit error/slave transmit complete interrupt.
4. It is possible to write one byte of data at a time to the FIFO, then wait for a transmit FIFO empty interrupt which means the master wants more data, or wait for the Transmit error/ slave transmit complete interrupt which indicates that the master has received the required data.
5. When transmit error/slave transmit complete has occurred, the NAS also occurs because the master has to either send a stop or send a repeated start.
6. When the NAS interrupt has been received, handle the data and clear AAS.
7. Wait for the AAS interrupt.

Dynamic Controller Logic Flow

For initialization, both the RX_FIFO and TX_FIFO should be empty, and the AXI IIC should be enabled by setting CR EN = 1.

Start and Repeated Start Sequence

When sending bytes of data over the IIC bus, the TX_FIFO is filled first with the 7-bit device address of the IIC peripheral and the read/write bit, and any required data. To wake up the dynamic logic, the device address is written to the TX_FIFO as a 16-bit word (10 bits are used by the AXI IIC) with the start bit set (Bit[8]). Then if a read is to be performed, write the receive byte count to the TX_FIFO else put the data to be written to it. When the dynamic logic detects that data is available in the TX_FIFO and that the start bit is set, the AXI IIC does the following:

1. Check the CR to see if MSMS is already set:
   a. If MSMS is not set, then set the MSMS bit to create a start sequence.
   b. If MSMS is already set, then set the RSTA in the CR to create a repeated start sequence.
2. Transmit the 7-bit address and R/W bit contained in the TX_FIFO.
3. Check the least significant bit contained with the 7-bit address to determine if this is a read or write operation on the IIC bus.
4. Get the next byte in the TX_FIFO
5. If a read access is occurring on the IIC bus, use this value as a receive byte counter. When this counter reaches zero, CR TXAK is forced High. This causes a not-acknowledge to be generated during reception of the last byte and signals the IIC slave device to stop transmitting read data.
6. If a write access is occurring, the contents of the TX_FIFO are sent out on the `sda` bus.

Stop Sequence

For the AXI IIC core to release the IIC bus, by clearing the MSMS bit in the CR, Bit[9] must be set in the TX_FIFO with the last byte to be sent for a write access. For a read access, Bit[9] must be set when the second word is written to the TX_FIFO. The least significant eight bits of the second word contain the number of bytes to receive. If the stop bit is never set, the AXI IIC continues to own the IIC bus.

Pseudo Code for Dynamic IIC Accesses

AMD recommends verifying that the TX_FIFO is not full or does not overflow with the writing of new data. For read accesses, you should reset the RX_FIFO or check that SR(RX_FIFO_Empty) = 1.

Initialization

1. Set the RX_FIFO depth to maximum by setting RX_FIFO_PIRQ = 0x0F.
2. Reset the TX_FIFO.
3. Enable the AXI IIC, remove the TX_FIFO reset, disable the general call.

Read Four Bytes from an IIC Device Addressed as 0x34

1. Check all FIFOs empty and bus not busy by reading the Status register.
2. Write 0x135 to the TX_FIFO (set start bit, device address to 0x34, read access).
3. Write 0x204 to the TX_FIFO (set stop bit, four bytes to be received by the AXI IIC).
4. Wait for RX_FIFO not empty.
    a. Read the RX_FIFO byte.
    b. If the fourth byte is read, then exit; otherwise, continue checking RX_FIFO not empty.

Write Four Bytes (0x89, 0xAB, 0xCD, 0xEF) to an IIC Slave EEPROM Device Addressed as 0x34

Place the data at EEPROM address 0x33:

1. Check all FIFOs empty and bus not busy by reading the SR.
2. Write 0x134 to the TX_FIFO (set the start bit, the device address, write access).
3. Write 0x33 to the TX_FIFO (EEPROM address for data).
4. Write 0x89 to the TX_FIFO (byte 1).
5. Write 0xAB to the TX_FIFO (byte 2).
6. Write 0xCD to the TX_FIFO (byte 3).
7. Write 0x2EF to the TX_FIFO (stop bit, byte 4).

Read Four Bytes from an IIC Slave EEPROM Device Addressed as 0x34

The data is at EEPROM address 0x33. First, a write access is necessary to set the EEPROM address, then a repeated start follows with the read accesses:

1. Check all FIFOs empty and bus not busy by reading the Status register.
2. Write 0x134 to the TX_FIFO (set start bit, device address to 0x34, write access).
3. Write 0x33 to the TX_FIFO (EEPROM address for data).
4. Write 0x135 to the TX_FIFO (set start bit for repeated start, device address 0x34, read access).
5. Write 0x204 to the TX_FIFO (set stop bit, four bytes to be received by the AXI IIC).

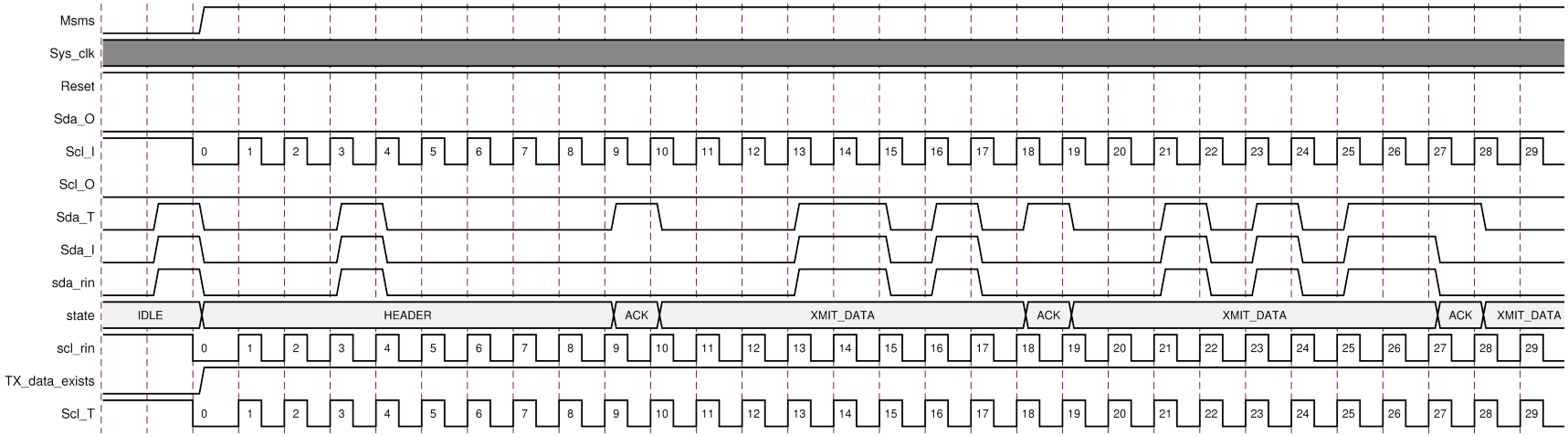
6. Wait for RX_FIFO not empty.
   a. Read the RX_FIFO byte.
   b. If the fourth byte is read, exit; otherwise, continue checking RX_FIFO not empty.

Timing Diagrams

The timing diagrams in this section depict the functionality of the core.

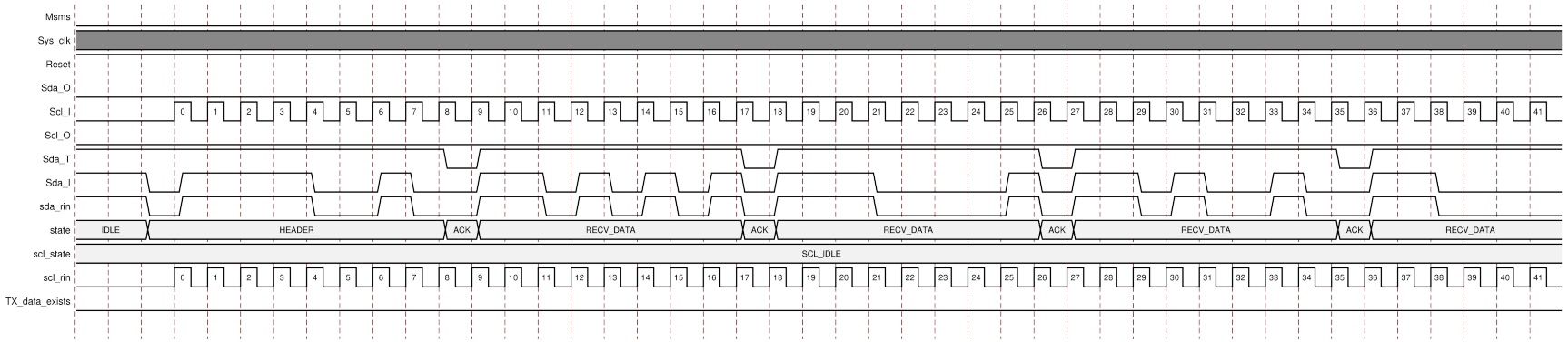
The following figure shows the waveforms for when the AXI IIC core is working as a master transmitting data.

Figure 24: **Master Mode – Transmitting Data**



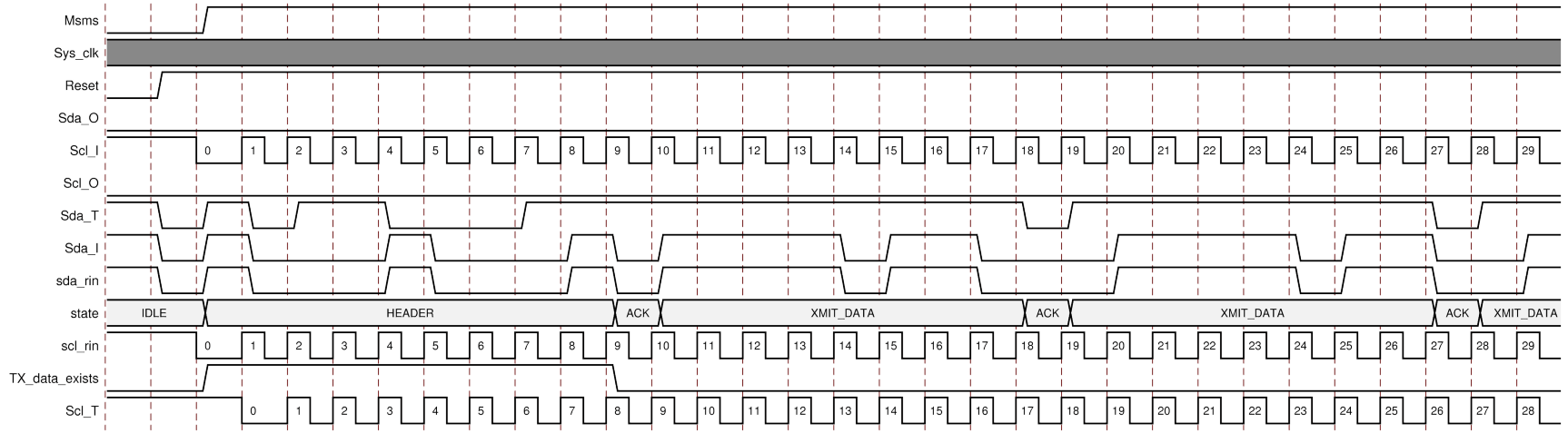
The following figure shows the waveforms for when the AXI IIC core is working as a slave receiving data.

Figure 25: **Slave Mode – Receiving Data**



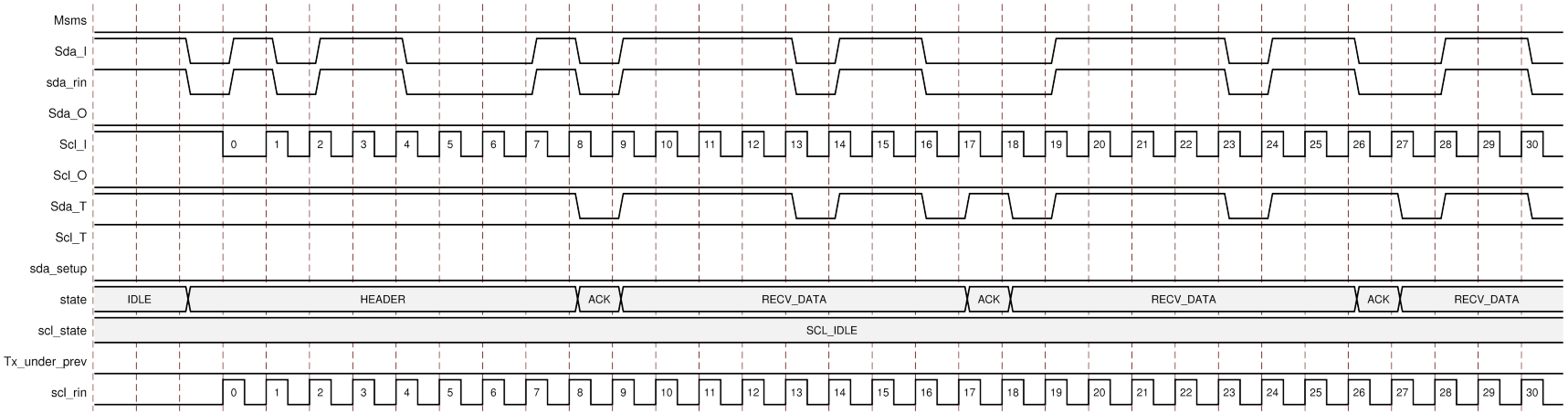
The following figure shows the waveforms for when the AXI IIC core is working as a master receiving data.

Figure 26: **Master Mode – Receiving Data**



The following figure shows the waveforms for when the AXI IIC core is working as a slave transmitting data.

Figure 27: **Slave Mode – Transmitting Data**



Clocking

The AXI IIC core works on the AXI clock. For maximum supported values see Table 1.

Resets

The AXI IIC core works on the `s_axi_aresetn`, which is active-Low and should be synchronous to `s_axi_aclk`.

Chapter 5

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard AMD Vivado™ design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using AMD tools to customize and generate the core in the AMD Vivado™ Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Figure 28: **Vivado Customize IP Dialog Box**

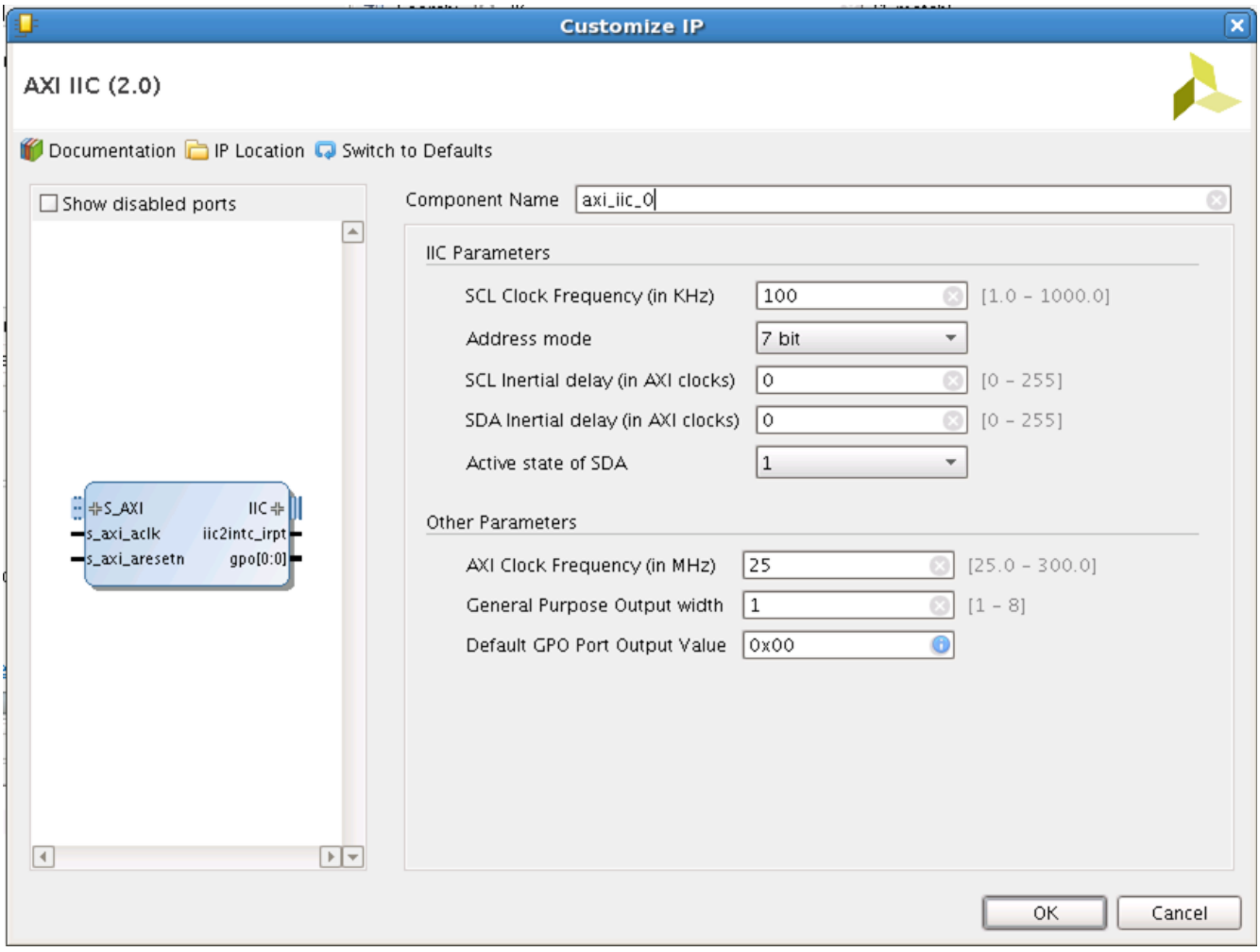
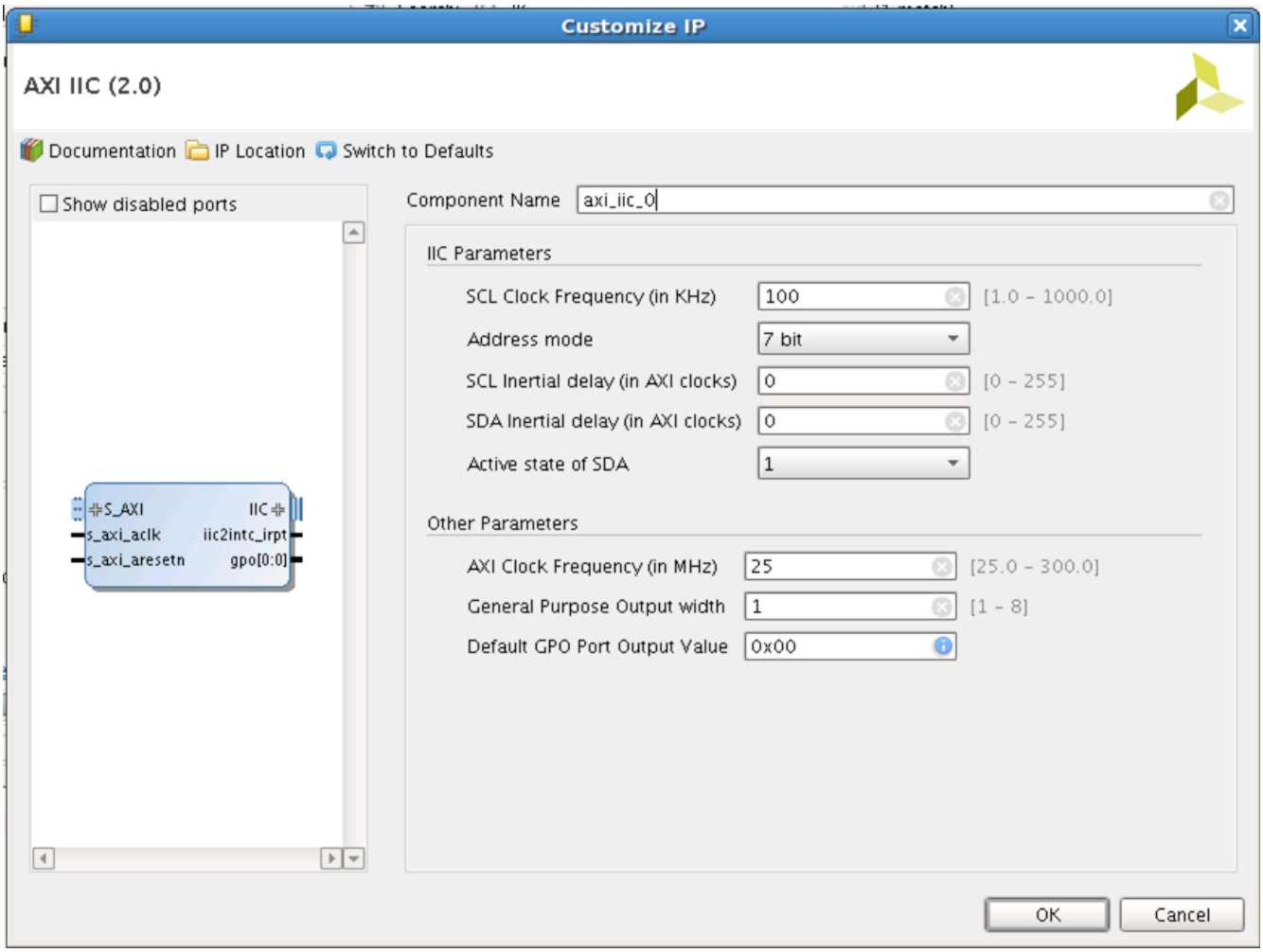


Figure 29: **Vivado IP Integrator**



IIC Parameters

- **SCL Clock Frequency (in KHz):** Determines the approximate frequency of the master mode generated SCL clock signal (Hz).
  - For SCL Clock Frequency < 100,000, the appropriate timing specifications for Standard Mode operation are used.
  - For SCL Clock Frequency > 100,000 and SCL Clock Frequency < 400,000, the specifications for Fast Mode operation are used.
  - For SCL Clock Frequency > 400,000, the timing specifications for Fast-Mode Plus operation are used. See the *Philips Semiconductors I^2C-bus Specification* for details.

IMPORTANT! *SCL clock is generated from the AXI clock. The SCL clock frequency can vary and is not guaranteed to be exactly the same as what was requested. The SCL clock period can differ from the requested value by up to four clock period time of AXI clock. This does not affect the I^2C operation.*

- **Address Mode:** Can be configured either with 7-bit or 10-bit addressing mode.
- **SCL Inertial Delay (in AXI Clocks):** Configures the width of the pulse rejection on the SCL signal.
- **SDA Inertial Delay (in AXI Clocks):** Configures the width of the pulse rejection on the SDA signal.
- **Active State of SDA:** Used during transmit throttling when the AXI IIC acts as a master transmitter. In this case, the AXI IIC master transmitter drives the SDA line with the Active State of SDA value.

Other Parameters

- **AXI Clock Frequency (in MHz):** The AXI clock frequency must be at least 25 MHz and 25 times faster than the SCL clock frequency.

> **IMPORTANT!** *When using this IP in IP integrator, this value is auto set based on the connected clock.*

- **General Purpose Output Width:** This is the width of the General Purpose Output port.
- **Default GPO Port Output Value:** This is the value driven on General Purpose Output port after the IP is out of reset. The value is 8-bit wide and depending on the GPO port width the LSB bits of the programmed value are driven onto GPO port.

User Parameters

The following table shows the relationship between the fields in the AMD Vivado™ IDE and the user parameters (which can be viewed in the Tcl Console).

Table 27: **GUI Parameter to User Parameter Relationship**

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
SCL Clock Frequency (in KHz)	IIC_FREQ_KHZ	100
Address Mode Allowed values include: • 7-bit • 10 bit	TEN_BIT_ADR	7-bit
SCL Inertial delay (in AXI Clocks) Allowed range is from 0 to 255.	C_SCL_INERTIAL_DELAY	0
SDA Inertial delay (in AXI Clocks) Allowed range is from 0 to 255.	C_SDA_INERTIAL_DELAY	0
Active State of SDA Allowed values are 0 and 1.	C_SDA_LEVEL	1
AXI Clock Frequency (in MHZ) Allowed range is from 25 to 300.	AXI_ACLK_FREQ_MHZ	25

Table 27: **GUI Parameter to User Parameter Relationship** *(cont'd)*

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
General Purpose Output Width Allowed range is from 1 to 8.	C_GPO_WIDTH	1
Default GPO Port Output Value Allowed range from 0x00 to 0xFF.	C_DEFAULT_VALUE	0x00

Notes:

1. Parameter values are listed in the table where the GUI parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Constraining the Core

This section contains information about constraining the core in the AMD Vivado™ Design Suite.

Required Constraints

Necessary XDC constraints are delivered along with IP generation.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

The AXI IIC can operate on `s_axi_aclk` frequency of 25 to 500 MHz.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

This section contains information about simulating IP in the Vivado Design Suite.

For comprehensive information about AMD Vivado™ simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Synthesis and Implementation

This section contains information about synthesis and implementation in the Vivado Design Suite.

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

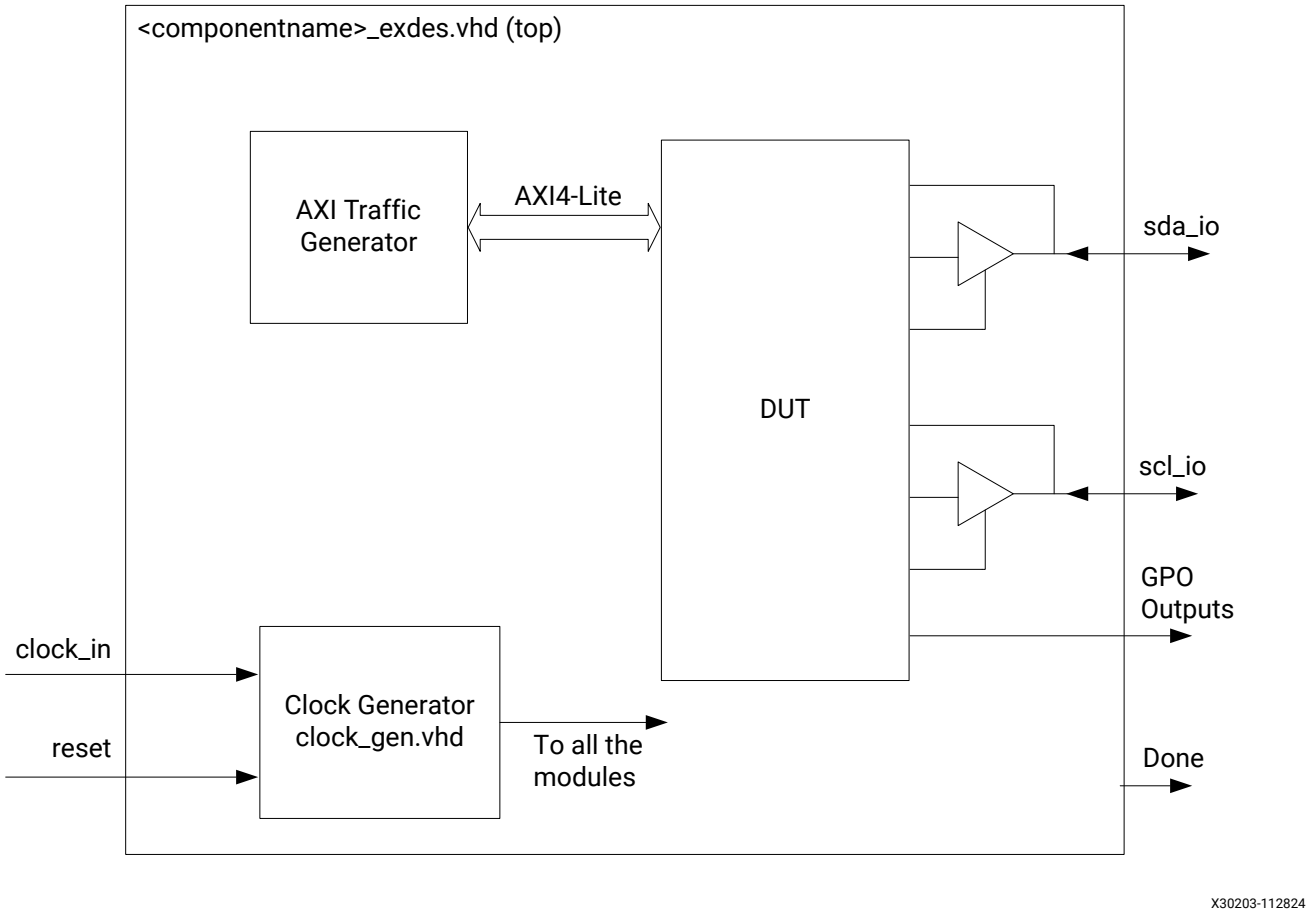
Example Design

This section contains information about the example design provided in the AMD Vivado™ Design Suite.

Overview

The top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in the following figure. This includes clock generator (MMCME2) and AXI Traffic Generator module.

Figure 30: **AXI IIC Example Design Block Diagram**



This example design demonstrates transactions on AXI4-Lite interfaces of the DUT to execute a repeated start read sequence using AXI IIC.

- **Clock Generator:** MMCME2 is used to generate the clocks for the example design. The example design assumes the input clocks as 200 MHz. The clock generator module generates the clock frequency specified in the customization Vivado IDE.
- **AXI Traffic Generator (ATG):** This module (IP) is configured in System Test Mode. All the AXI_IIC related AXI4-Lite transactions are stored in the coe/mif file. For more information on AXI Traffic Generator, see the *AXI Traffic Generator LogiCORE IP Product Guide* (PG125). The ATG automatically starts the AXI4-Lite transaction after coming out of reset. The example design demonstrates the programming sequence required to perform a repeated start read operation. Upon successful completion of AXI IIC operation, the ATG writes "FF" to theGeneral Purpose Output Register (GPO). For the example design status, you can connect the GPO outputs of the AXI IIC to the LED. On successful completion, two or more LEDs glow (Done and GPO output). In case of a failure, only one LED would glow.

Implementing the Example Design

After following the steps described in Customizing and Generating the Core to generate the core, implement the example design as follows:

1. Right-click the core in the Hierarchy window, and select **Open IP Example Design**.
2. A new window pops up, asking you to specify a directory for the example design. Select a new directory or keep the default directory.
3. A new project is automatically created in the selected directory and it is opened in a new Vivado window.
4. Provide the location constraints as per the board.
5. In the Flow Navigator (left-side pane), click **Run Implementation** and follow the directions.

Example Design Directory Structure

In the current project directory, a new project with name `<component_name>_example` is created and the files are generated in `<component_name>_example/<component_name>_example.srcs/` directory. This directory and its subdirectories contain all the source files that are required to create the AXI IIC controller example design.

The following table shows the files delivered as part of the example design.

Table 28: **Example Design Directory**

Name	Description
<component_name>_exdes.vhd	Top-level HDL file for the example design.
clock_gen.vhd	Clock generation module for example design.
atg_addr.coe	COE file of address. This file contains the AXI IIC register address.
atg_data.coe	COE file of data. This file contains the data to be written/read from the AXI IIC registers.
atg_mask.coe	COE file to mask certain reads.
atg_ctrl.coe	COE file that contains control information of ATG.

The following table shows the test bench that is delivered to simulate the example design.

Table 29: **Simulation Directory**

Name	Description
<component_name>_exdes_tb.vhd	Test bench for example design.

The following table shows the XDC file that is delivered to implement the example design.

Table 30: **Constraints Directory**

Name	Description
<component_name>_exdes.xdc	Top-level constraints file for example design.

The XDC has all the necessary constraints needed to run the example design on the KC705 board. All I/O constraints are commented in the XDC file.

IMPORTANT! *Uncomment the XDC constraints before implementing the design for KC705 board.*

Simulating the Example Design

Using the AXI IIC example design (delivered as part of the AXI IIC), you can quickly simulate and observe the behavior of the AXI IIC.

Setting Up the Simulation

The AMD simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) for assistance compiling AMD simulation models and setting up the simulator environment. To switch simulators, click Simulation Settings in the Flow Navigator (left pane). In the Simulation options list, change Target Simulator.

Simulation Results

The simulation script compiles the AXI IIC example design and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully.

If the test passes, then the following message is displayed:

```
Test Completed Successfully
```

If the test fails or does not complete, then the following message is displayed:

```
Test Failed!! Test Timed Out.
```

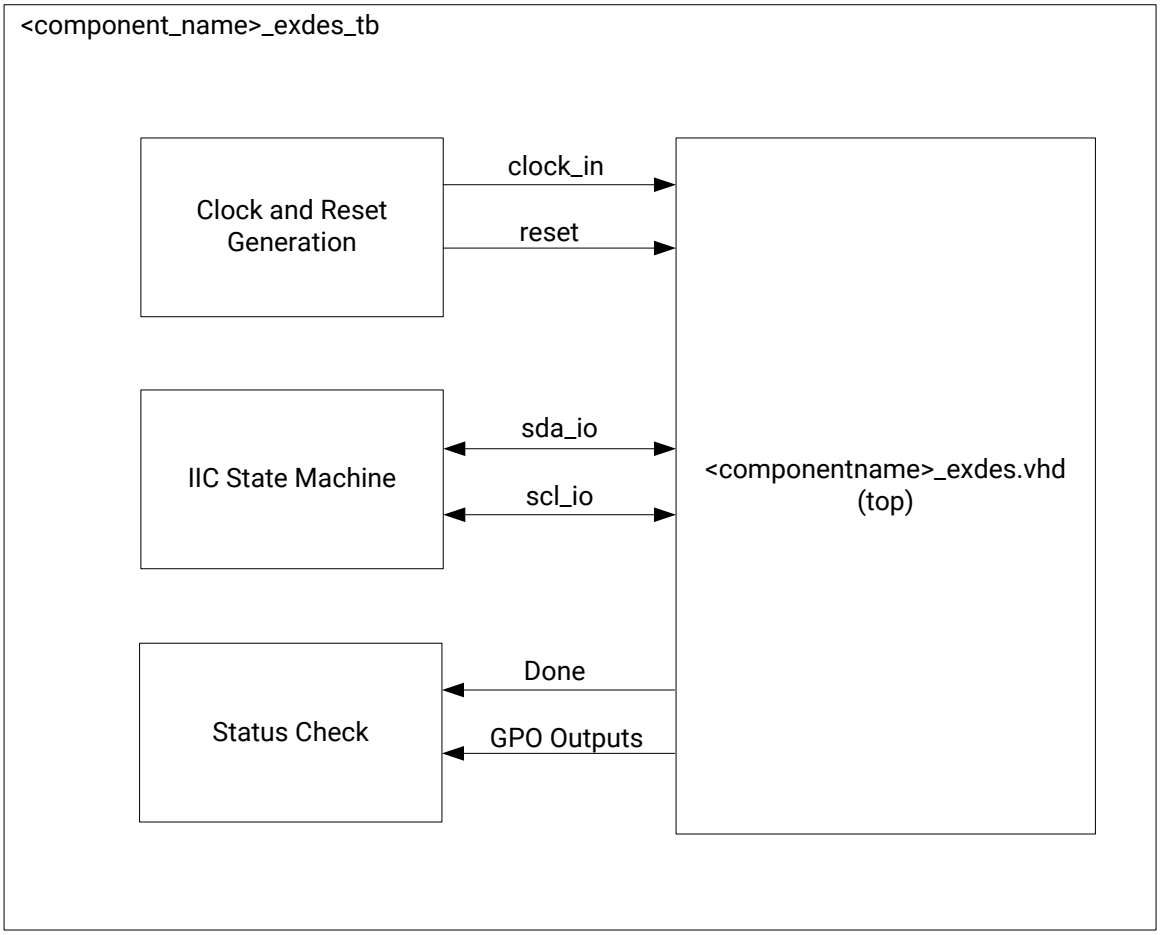
Test Bench

This chapter contains information about the test bench provided in the AMD Vivado™ Design Suite.

The following figure shows the test bench for the AXI IIC example design. The top-level test bench generates a top-level 200 MHz clock and drives initial reset to the example design. The example test bench also acts as an IIC Slave for the repeated start read sequence.

Note: This test bench is configured only for the particular IIC transaction that is being carried out in the example design. This test bench does not work for any other custom IIC transaction.

Figure 31: **AXI IIC Example Design Test Bench**



Appendix A

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the AMD Vivado™ Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are include

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911).

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Appendix B

Debugging

This appendix includes details about resources available on the AMD Support website and debugging tools.

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Finding Help with AMD Adaptive Computing Solutions

To help in the design and debug process when using the core, the Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Community Forums are also available where members can learn, participate, share, and ask questions about AMD Adaptive Computing solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the AMD Adaptive Support web page or by using the AMD Adaptive Computing Documentation Navigator. Download the Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with an AMD Adaptive Computing product. Answer Records are created and maintained daily to ensure that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main AMD Adaptive Support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for AXI IIC Core

AR: 54435.

Technical Support

AMD Adaptive Computing provides technical support on the Community Forums for this AMD LogiCORE™ IP product when used as described in the product documentation. AMD Adaptive Computing cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the Community Forums.

Debug Tools

There are many tools available to address AXI IIC core design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The AMD Vivado™ Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in AMD devices.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Hardware Debug

For hardware issues, see Electrical Issues.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and `output s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and aclk inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or the Vivado Design Suite debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.

Appendix C

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
2. *Vivado Design Suite: AXI Reference Guide* (UG1037)
3. *Vivado Design Suite User Guide: Getting Started* (UG910)
4. *AXI Traffic Generator LogiCORE IP Product Guide* (PG125)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155)
9. *ISE to Vivado Design Suite Migration Guide* (UG911)
10. *Vivado Design Suite User Guide: Designing with IP* (UG896)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/18/2024 Version 2.1	
IIC Slave Receiver	Updated Control register information.
08/20/2021 Version 2.1	
General updates	Fixed RTL to initiate read on IIC only after getting the value for the number of bytes to receive in dynamic mode.
10/05/2016 version 2.0	
General updates	• Added a note in Register Space. • Added the Automotive Applications Disclaimer in Please Read: Important Legal Notices.

Section	Revision Summary
04/06/2016 Version 2.0	
General updates	• Updated Standards section. • Added note #2 in Table 2-23: TBUF Register (138h).
11/18/2015 Version 2.0	
General updates	Added support for UltraScale+ families.
11/19/2014 Version 2.0	
General updates	• Document updates only for revision change. • Updated #2 description in Interrupt(1): Transmit Error/Slave Transmit Complete section. • Updated description in Interrupt(3): Receive FIFO Full section. • Updated Table 2-18: General Purpose Output Register. • Updated descriptions for Timing Parameter TSUSTA Register (TSUSTA) to Timing Parameter THDDAT Register (THDDAT) sections.
10/01/2014 Version 2.0	
General updates	• Document updates only for revision change. • Updated Bits[31:7] in Table 2-9: Control Register. • Updated Figs. 4-1 to 4-2. • Added Default GPO Port Output Value parameter. • Added User Parameter table in Design Flow Steps chapter.
04/02/2014 Version 2.0	
General updates	• Updated bullet lists in Functional Description section. • Added description in Bit[7] in Control register. • Updated descriptions and access type to W in Transmit FIFO register. • Updated description in Slave 10-Bit Address register. • Updated descriptions in Transmit FIFO Occupancy and Receive FIFO Occupancy registers. • Updated offsets for TX_FIFO start bit in Pseudo Code for Dynamic IIC Accesses section. • Updated description in Resets section. • Added a Design Flow Steps chapter. • Updated Example Design Directory Structure section.
12/18/2013 Version 2.0	
General updates	Added UltraScale support.

Section	Revision Summary
10/02/2013 Version 2.0	
General updates	• Revision number advanced to 2.0 to align with core version number 2.0. • Updated Table 2-1 Maximum Frequencies. • Updated Resource Utilization section. • Added Important note in Interrupt(2): Transmit FIFO Empty section. • Updated Bit[7] in Status register. • Added IP integrator. • Added Simulation, Synthesis, Example Design, and Test Bench chapters. • Updated Migrating Appendix.
03/20/2013 Version 2.0	
General updates	• Updated core v2.0 for Vivado Design Suite only. • Updated Fig. 1-1 and Overview section. • Updated Max. Frequencies table. • Updated Resource Estimation tables • Updated I/O Signal table. • Updated address offset in AXI IIC Register Map. • Updated Timing Diagrams section. • Updated Fig. 4-1 GUI • Added Debug section. • Updated to Questa SIM.
10/16/2012 Version 1.0	
General updates	Initial Xilinx release of core documentation as a product guide. Replaces LogiCORE IP AXI IIC Bus Interface Data Sheet, DS756. Updated for XPS 14.3 and Vivado Design Suite v2012.3.

Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES,

ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright