



AXI Interconnect Core User Guide

UG-CORE-AXI-INTERCON-v1.7
June 2025
www.efinixinc.com



Contents

Introduction.....	3
Features.....	3
Device Support.....	3
Resource Utilization and Performance.....	4
Release Notes.....	4
Functional Description.....	5
Ports.....	6
Arbitration Modes.....	11
Address Decoders.....	12
AXI Interconnect Operations.....	13
IP Manager.....	16
Customizing the AXI Interconnect.....	17
AXI Interconnect Example Design.....	18
AXI Interconnect Testbench.....	19
Revision History.....	20

Introduction

The AXI Interconnect core manages traffic on the AXI interfaces where it allows you to connect one or more AXI memory-mapped masters to one or more AXI memory-mapped slaves. For example, when multiple masters are issuing AXI transaction requests simultaneously, the AXI Interconnect core determines which AXI master issues the AXI transaction.

The AXI Interconnect core targets applications such as multiple masters accessing HyperRAM controller using half duplex AXI mode or configurations on multi-slave devices using the same master device.

Use the IP Manager to select IP, customize it, and generate files. The AXI Interconnect core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.

Features

- Supports AXI3, AXI4, and AXI4-lite interfaces
- Supports N-to-1, 1-to-N, N-to-M (shared access mode) interconnect
- Supports 3 arbitration modes
 - Fixed priority
 - Round robin 1
 - Round robin 2
- Verilog HDL RTL and simulation testbench

Device Support

Table 1: AXI Interconnect Core Device Support

FPGA Family	Supported Device
Trion	All
Titanium	All
Topaz	All

Resource Utilization and Performance



Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and may change depending on the device resource utilization, design congestion, and user design.

Table 2: Titanium Resource Utilization and Performance

FPGA	Configuration	Logic and Adders	Flip-flops	Memory Blocks	DSP Blocks	f _{MAX} (MHz)	Efinity® Version
Ti60 F225 C4	1-to-8 with 32-bit data width	470	214	0	0	360	2022.1
	2-to-1 with 128-bit data width	910	720	0	0	315	

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes are available in the [Efinity page of the Support Center](#) under each Efinity software release version.



Note: You must be logged in to the Support Center to view the IP Core Release Notes.

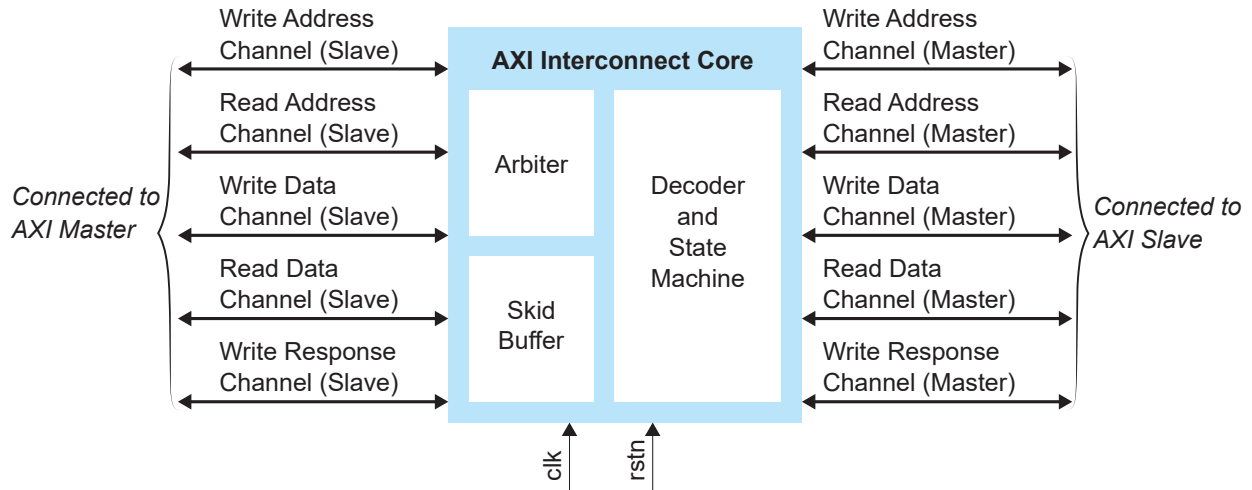
⁽¹⁾ Using Verilog HDL.

Functional Description

The AXI Interconnect core consists of the following blocks:

- *Arbiter*—Logic for 3 different arbitration modes
- *Skid Buffer*—Datapath for write data channel and read data channel
- *Address decoders*—Decodes address from the incoming AXI transaction and determines the targeted AXI slave transaction

Figure 1: AXI Interconnect System Block Diagram



Ports



Note: *M* represents the number of AXI master ports while *S* represents the number of AXI slave ports.

Table 3: Global

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
clk	Input	✓	✓	✓	Clock.
rst_n	Input	✓	✓	✓	Active low asynchronous reset.

Table 4: Write Address Channel (Slave)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
s_axi_awvalid[S*1:0]	Input	✓	✓	✓	Write address channel valid.
s_axi_awaddr[S*ADDR_WIDTH-1:0]	Input	✓	✓	✓	Write address channel address.
s_axi_awprot[S*3-1:0]	Input	✓	✓	✓	Write address channel protect.
s_axi_awid[S*ID_WIDTH-1:0]	Input	✓	✓	-	Write address channel transaction ID.
s_axi_awburst[S*2-1:0]	Input	✓	✓	-	Write address channel burst type.
s_axi_awlen[S*8-1:0]	Input	✓	✓	-	Write address channel burst length.
s_axi_awsz[S*3-1:0]	Input	✓	✓	-	Write address channel transfer size.
s_axi_awcache[S*4-1:0]	Input	-	✓	-	Write address channel cache encoding.
s_axi_awqos[S*4-1:0]	Input	-	✓	-	Write address channel Quality of Service (QoS).
s_axi_awuser[S*USER_WIDTH-1:0]	Input	-	✓	-	Write address channel user-defined signals.
s_axi_awlock[S*2-1:0]	Input	✓	-	-	Write address channel locked transaction. Tie to 0 when using AXI Lite.
s_axi_awready[S-1:0]	Output	✓	✓	✓	Write address channel ready.

Table 5: Write Data Channel (Slave)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
s_axi_wvalid[S-1:0]	Input	✓	✓	✓	Write channel valid.
s_axi_wdata[S*DATA_WIDTH-1:0]	Input	✓	✓	✓	Write channel data.
s_axi_wstrb[S*STRB_WIDTH-1:0]	Input	✓	✓	✓	Write channel strobe (Single bit represents data byte).
s_axi_wlast[S-1:0]	Input	✓	✓	-	Write channel last data beat. Tie to 1 when using AXI Lite.
s_axi_wuser[S*USER_WIDTH-1:0]	Input	-	✓	-	Write channel user-defined signals.
s_axi_wid[S*ID_WIDTH-1:0]	Input	✓	-	-	Write channel transaction ID. Tie to 0 when using AXI Lite.
s_axi_wready[S-1:0]	Output	✓	✓	✓	Write channel ready.

Table 6: Write Response Channel (Slave)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
s_axi_bready[S-1:0]	Input	✓	✓	✓	Write response channel ready.
s_axi_bresp[S*2-1:0]	Output	✓	✓	✓	Write response channel response.
s_axi_bvalid[S-1:0]	Output	✓	✓	✓	Write response channel valid.
s_axi_bid[S*ID_WIDTH-1:0]	Output	✓	✓	-	Write response channel transaction ID.
s_axi_buser[S*USER_WIDTH-1:0]	Output	-	✓	-	Write response channel user-defined ID.

Table 7: Read Address Channel (Slave)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
s_axi_arvalid[S-1:0]	Input	✓	✓	✓	Read address channel valid.
s_axi_araddr[S*ADDR_WIDTH-1:0]	Input	✓	✓	✓	Read address channel address.
s_axi_arprot[S*3-1:0]	Input	✓	✓	✓	Read address channel protect.
s_axi_arid[S*ID_WIDTH-1:0]	Input	✓	✓	-	Read address channel transaction ID.
s_axi_arburst[S*2-1:0]	Input	✓	✓	-	Read address channel burst type.
s_axi_arlen[S*8-1:0]	Input	✓	✓	-	Read address channel burst length.
s_axi_arsize[S*3-1:0]	Input	✓	✓	-	Read address channel transfer size.
s_axi_arcache[S*4-1:0]	Input	-	✓	-	Read address channel cache encoding.
s_axi_arqos[S*4-1:0]	Input	-	✓	-	Read address channel QoS.
s_axi_aruser[S*USER_WIDTH-1:0]	Input	-	✓	-	Read address channel user-defined signals.
s_axi_arlock[S*2-1:0]	Input	✓	-	-	Read address channel locked transaction.
s_axi_arready[S-1:0]	Output	✓	✓	✓	Read address channel ready.

Table 8: Read Data Channel (Slave)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
s_axi_rready[S-1:0]	Input	✓	✓	✓	Read data channel ready.
s_axi_rid[S*ID_WIDTH-1:0]	Output	✓	✓	-	Read data channel transaction ID.
s_axi_rdata[S*DATA_WIDTH-1:0]	Output	✓	✓	✓	Read data channel data.
s_axi_rresp[S*2-1:0]	Output	✓	✓	✓	Read data channel response.
s_axi_rvalid[S-1:0]	Output	✓	✓	✓	Read data channel valid.
s_axi_rlast[S-1:0]	Output	✓	✓	-	Read data channel last data beat.
s_axi_ruser[S*USER_WIDTH-1:0]	Output	-	✓	-	Read data user-defined channel.

Table 9: Write Address Channel (Master)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_awvalid[M-1:0]	Output	✓	✓	✓	Write address channel valid.
m_axi_awaddr[M*ADDR_WIDTH-1:0]	Output	✓	✓	✓	Write address channel address.
m_axi_awprot[M*3-1:0]	Output	✓	✓	✓	Write address channel protect.
m_axi_awid[M*ID_WIDTH-1:0]	Output	✓	✓	-	Write address channel transaction ID.
m_axi_awburst[M*2-1:0]	Output	✓	✓	-	Write address channel burst type.
m_axi_awlen[M*8-1:0]	Output	✓	✓	-	Write address channel burst length.
m_axi_awsize[M*3-1:0]	Output	✓	✓	-	Write address channel transfer size.

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_awcache[M*4-1:0]	Output	-	✓	-	Write address channel cache encoding.
m_axi_awqos[M*4-1:0]	Output	-	✓	-	Write address channel QoS.
m_axi_awuser[M*USER_WIDTH-1:0]	Output	-	✓	-	Write address channel user-defined signals.
m_axi_awlock[M*2-1:0]	Output	✓	-	-	Write address channel locked transaction.
m_axi_awready[M-1:0]	Input	✓	✓	✓	Write address channel ready.

Table 10: Write Data Channel (Master)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_wvalid[M-1:0]	Output	✓	✓	✓	Write channel valid.
m_axi_wdata[M*DATA_WIDTH-1:0]	Output	✓	✓	✓	Write channel data.
m_axi_wstrb[M*STRB_WIDTH-1:0]	Output	✓	✓	✓	Write channel strobe (single bit represents data byte).
m_axi_wlast[M-1:0]	Output	✓	✓	-	Write channel last data beat.
m_axi_wuser[M*USER_WIDTH-1:0]	Output	-	✓	-	Write channel user-defined signals.
m_axi_wid[M*ID_WIDTH-1:0]	Output	✓	-	-	Write channel transaction ID.
m_axi_wready[M-1:0]	Input	✓	✓	✓	Write channel ready.

Table 11: Write Response Channel (Master)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_bready[M-1:0]	Output	✓	✓	✓	Write response channel ready.
m_axi_bresp[M*2-1:0]	Output	✓	✓	✓	Write response channel response.
m_axi_bvalid[M-1:0]	Output	✓	✓	✓	Write response channel valid.
m_axi_bid[M*ID_WIDTH-1:0]	Output	✓	✓	-	Write response channel transaction ID. Tie to 0 when using AXI Lite.
m_axi_buser[M*USER_WIDTH-1:0]	Input	-	✓	-	Write response channel user-defined ID.

Table 12: Read Address Channel (Master)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_arvalid[M-1:0]	Output	✓	✓	✓	Read address channel valid.
m_axi_araddr[M*ADDR_WIDTH-1:0]	Output	✓	✓	✓	Read address channel address.
m_axi_arprot[M*3-1:0]	Output	✓	✓	✓	Read address channel protect.
m_axi_arid[M*ID_WIDTH-1:0]	Output	✓	✓	-	Read address channel transaction ID.
m_axi_arburst[M*2-1:0]	Output	✓	✓	-	Read address channel burst type.
m_axi_arlen[M*8-1:0]	Output	✓	✓	-	Read address channel burst length.
m_axi_arsize[M*3-1:0]	Output	✓	✓	-	Read address channel transfer size.
m_axi_arcache[M*4-1:0]	Output	-	✓	-	Read address channel cache encoding.
m_axi_arqos[M*4-1:0]	Output	-	✓	-	Read address channel QoS.
m_axi_aruser[M*USER_WIDTH-1:0]	Output	-	✓	-	Read address channel user-defined signals.
m_axi_arlock[M*2-1:0]	Output	✓	-	-	Read address channel locked transaction.
m_axi_arready[M-1:0]	Input	✓	✓	✓	Read address channel ready.

Table 13: Read Data Channel (Master)

Port	Direction	AXI3	AXI4	AXI4-Lite	Description
m_axi_rready[M-1:0]	Output	✓	✓	✓	Read data channel ready.
m_axi_rid[M*ID_WIDTH-1:0]	Output	✓	✓	-	Read data channel transaction ID. Tie to 0 when using AXI Lite.
m_axi_rdata[M*DATA_WIDTH-1:0]	Output	✓	✓	✓	Read data channel data.
m_axi_rresp[M*2-1:0]	Output	✓	✓	✓	Read data channel response.
m_axi_rvalid[M-1:0]	Output	✓	✓	✓	Read data channel valid.
m_axi_rlast[M-1:0]	Output	✓	✓	-	Read data channel last data beat. Tie to 1 when using AXI Lite.
m_axi_ruser[M*USER_WIDTH-1:0]	Input	-	✓	-	Read data user-defined channel.

Arbitration Modes

The AXI Interconnect core includes arbiter engines that grant the request to an AXI master when more than one AXI master issues a request. The AXI Interconnect supports three types of arbitration modes.

Fixed Priority

In this mode, the arbiter always prioritizes the most significant bit (MSB) ports as indicated by the bits shown in bold in the following example. Lower priority ports suffer from starvation.

Request	11100000	11100000	11100000	1110010	00001011	00000000	00001111	11111111
Grant	1 0000000	1 0000000	1 0000000	1 0000000	0000 1 000	00000000	0000 1 000	1 0000000

Round Robin 1

The arbitration starts from the MSB port. When more than two transaction requests are issued at the same time, the arbiter grants the request to the port sitting on the right-hand side nearest to the previously served port.

Example:

Request	11100000	11100000	11100000	1110010	00001011	00000000	00001111	11111111
Grant	1 0000000	0 1 000000	00 1 00000	000000 1 0	0000000 1	00000000	0000 1 000	00000 1 00

Round Robin 2

The arbitration starts from the MSB port. The arbitration goes through all connected master ports sequentially with a counter. When there is no request from the assigned master, the arbiter grants the port sitting on the right-hand side nearest to the assigned port. There are no counter increments when there are no requests.

Example:

Request	11100000	11100000	11100000	1110010	00001011	00000000	00001111	11111111
Counter	7	6	5	4	3	3	2	1
Grant	1 0000000	0 1 000000	00 1 00000	000000 1 0	0000 1 000	00000000	00000 1 00	000000 1 0

Address Decoders

The address decoder performs address decoding from the incoming AXI transaction address and determines the targeted AXI slave transaction. The AXI Interconnect core does not alter the targeted address content and forwards it entirely to the slave device.

The following table shows an example of AXI slave ports that have a 12-bit address space. The AXI Interconnect core decodes the address and routes the transaction request to the corresponding slave without discarding the address offset.

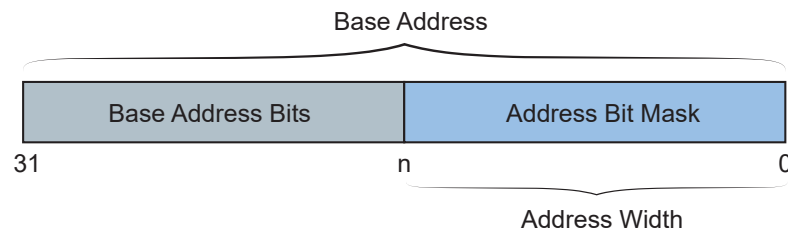
Table 14: Slave Base Address Setting

Original Address	Forwarded Address	Targeted Slave
AXI_S0	0x00000000	12
AXI_S1	0x00001000	12
AXI_S2	0x00002000	12
AXI_S3	0x00003000	12

Table 15: Slave Base Address Bits (Decoded Address)

Sent Address	Base Address Bits	Targeted Slave	Forwarded Address
0x00000FFF	0x00000	AXI_S0	0x00000FFF
0x00001FFF	0x00001	AXI_S1	0x00001FFF
0x00002FFF	0x00002	AXI_S2	0x00002FFF
0x00003FFF	0x00003	AXI_S3	0x00003FFF

Figure 2: Base Address Reference



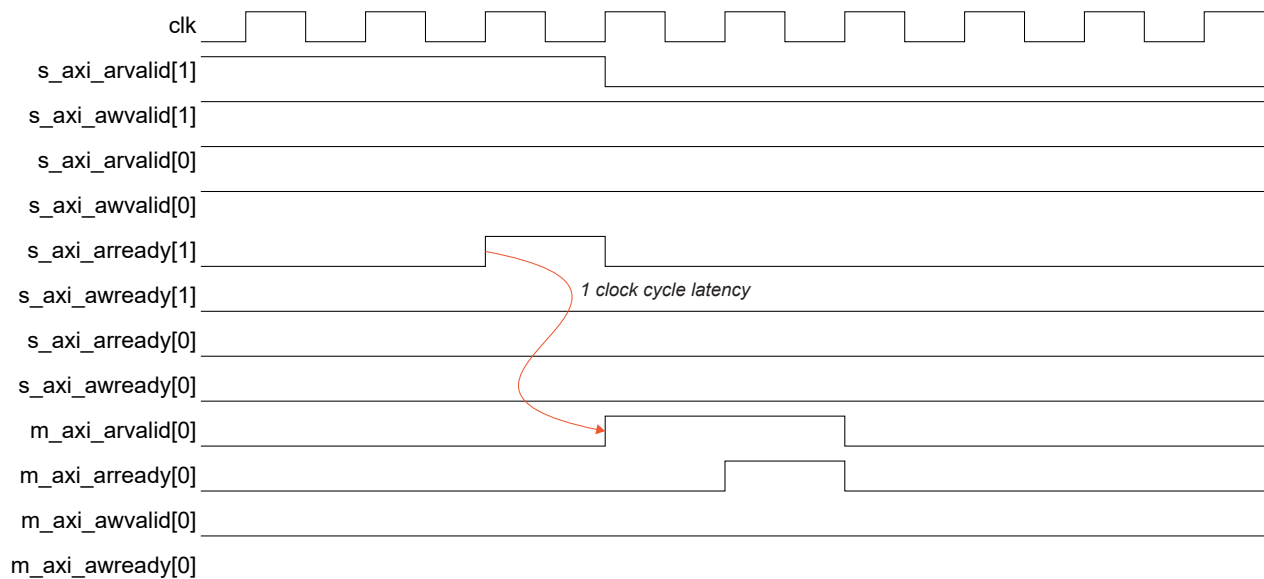
AXI Interconnect Operations

The following two waveforms illustrate a 2-to-1 interconnect example using fixed priority arbitration mode. There are three requests from two AXI master ports. The AXI Interconnect core grants read operation from port 1 (`s_axi_arvalid[1]`) because:

- Port 1 has a higher index (MSB)
- A read operation has a higher priority than a write operation

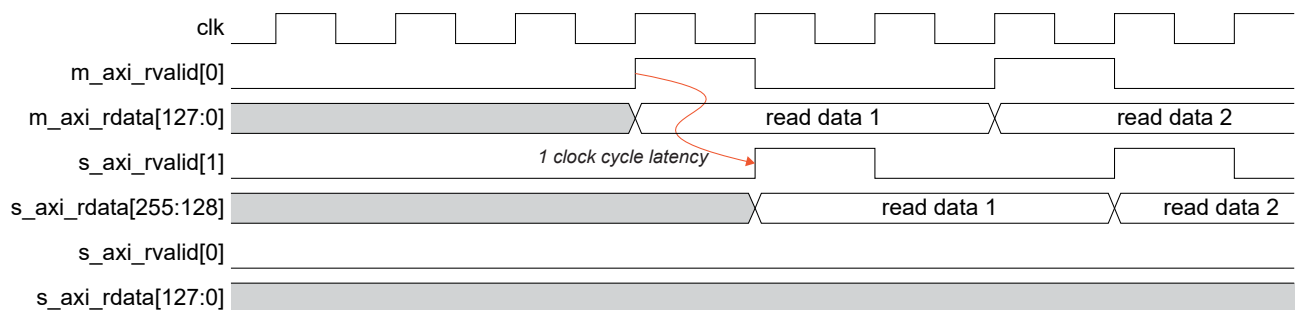
The ready signal of the address channel (`s_axi_arready[1]`) assertion indicates the request has been accepted. There is one clock cycle latency for the read request to be present on the AXI interconnect output.

Figure 3: Fixed Priority Arbitration 2-to-1 Interconnect Example Waveform (Request)



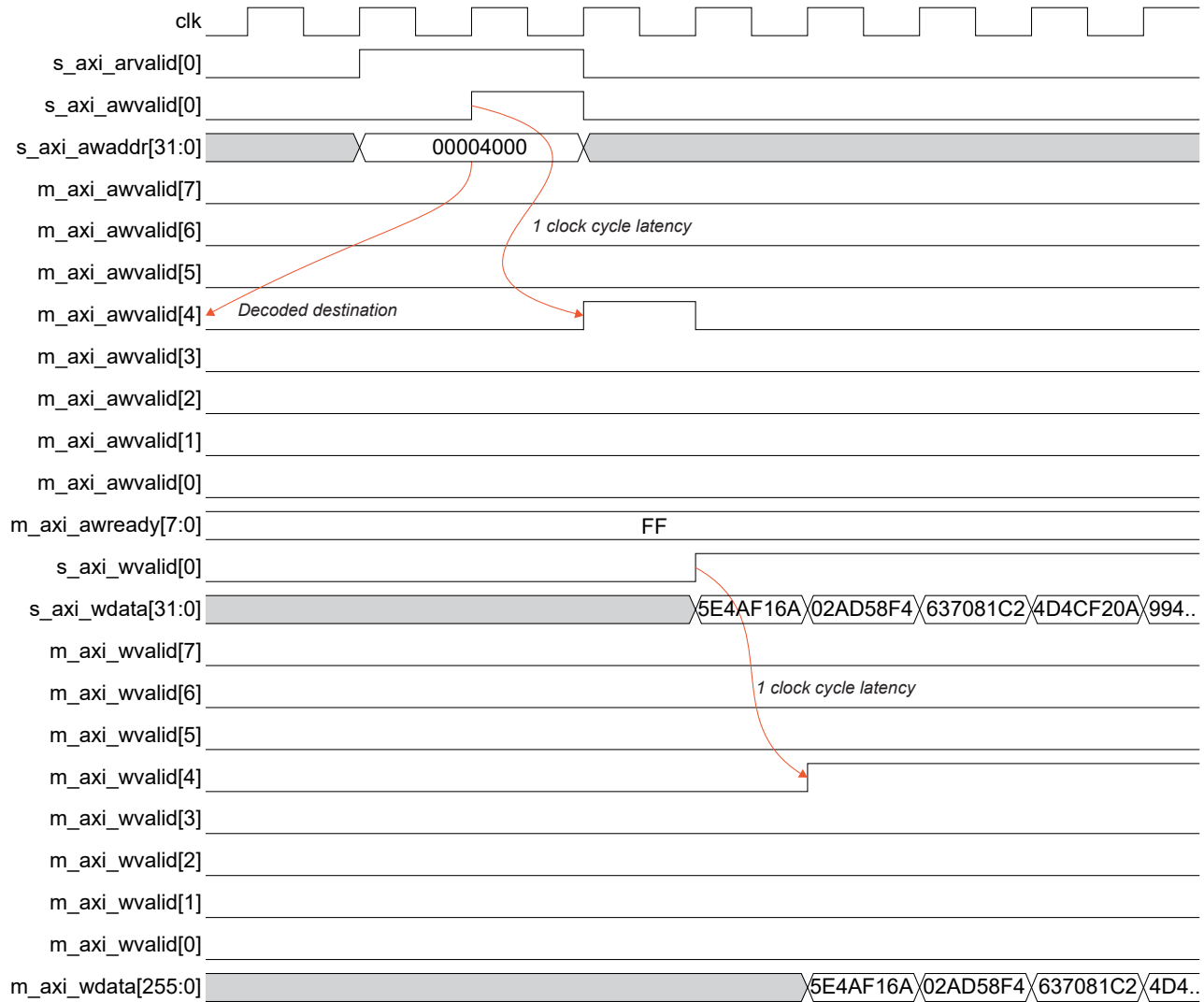
Read operations, there is one clock cycle latency for the data to be present on the AXI interconnect input.

Figure 4: Fixed Priority Arbitration 2-to-1 Interconnect Example Waveform (Read)



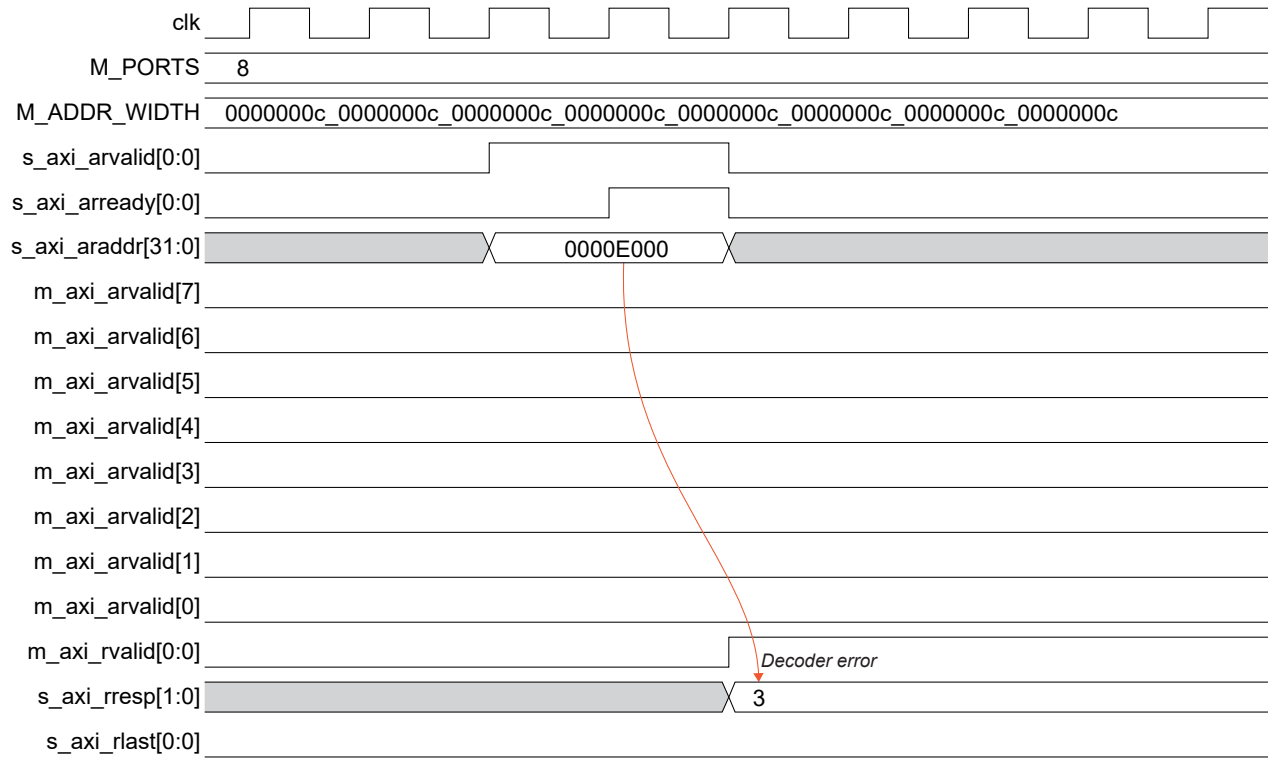
The following waveform illustrates the AXI interconnect of a 1-to-8 operation. The AXI interconnect decodes the slave destination from the write/read address channel and routes the transaction to the corresponding destination. For example, each of the connected slaves has a 12-bit address range. Address 'h4000 indicates that the transaction request is targeting slave port 4.

Figure 5: 1-to-8 Interconnect Example Waveform



The following waveform illustrates a transaction address request with an invalid range. Each of the connected slaves has a 12-bit address range. There are eight destination slaves configured in the AXI interconnect where the valid decoded address range is 0h0 to 0h7fff. However, the master side is driving a transaction address to hE000, which is out of the valid address range. The AXI Interconnect core denies the transaction by returning the decoder error.

Figure 6: Invalid Transaction Address Example Waveform



IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinity® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinity development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.



Note: Not all Efinity IP cores include an example design or a testbench.

Generating the AXI Interconnect Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **AXI Infrastructure > AXI Interconnect** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.



Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the AXI Interconnect* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinity® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.



Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tpl.v**—Verilog HDL instantiation template.
- **<module name>_tpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Customizing the AXI Interconnect

The core has parameters so you can customize its function. You set the parameters in the **General** tab of the core's IP Configuration window.

Table 16: AXI Interconnect Core Parameters (General Tab)

Parameters	Options	Description
Protocol	AXI3, AXI4, AXI4-LITE	Defines the AXI Interface Protocol Default: AXI4
Arbitration Mode	PRIORITY, ROUND_ROBIN_1, ROUND_ROBIN_2	Defines the Arbitration Mode Default: PRIORITY
Number of Slave Interfaces	1 - 8	Defines the number of slave interface connected to the master port. Default: 2
Number of Master Interfaces	1 - 8	Defines the number of master interface connected to the slave port. Default: 1
AXI Address Width	12 - 64	Defines the address channel address width. Default: 32
AXI Data Width	32, 64, 128, 256, 512	Defines the write/read channel data width. Default: 32
AXI ID Width	1 - 32	Defines the ID transaction width. Default: 1
AXI User Width	1 - 32	Defines the user define signal width. Default: 1

Table 17: AXI Interconnect Core Parameters (AXI Tab)

The number of AXI_S rows depends on the Number of Slave Interfaces parameter you set.

Parameters	Options	Description
Destination Address Space (Address Width)	12 - 32	Defines the slave address width. You can configure multiple slaves with different address widths by concatenating the parameter. Example: Slave 0 and slave 1 have an address width of 12, slave 2 has an address width of 20, and slave 3 has an address width of 24. The correct parameter setting is {32'd24, 32'd20, 32'd12, 32'd12}. Default: 12
Destination Address Offset (Base Address)	h00001000 - hFFFFFF00	Starting offset address for each slave. Example: The starting offset address for each slave are: <ul style="list-style-type: none"> slave 0 (12-bit) : 00000000 slave 1 (12-bit) : 00001000 slave 2 (20-bit) : 00100000 slave 3 (24-bit) : 01000000 The correct parameter setting is, {32'h01000000, 32'h00100000, 32'h00001000, 32'h00000000}

AXI Interconnect Example Design

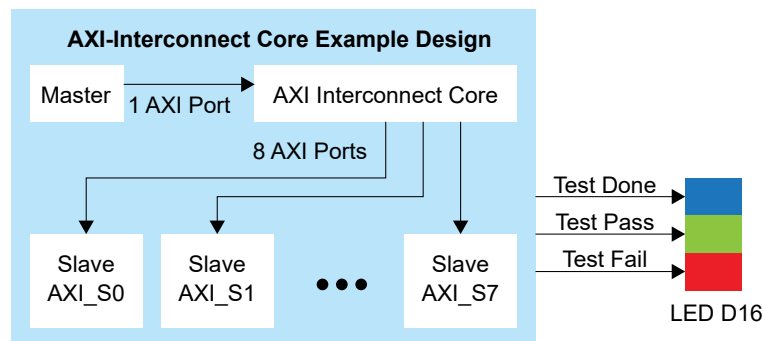
You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board. To generate example design, the **Example Design Deliverables Option** signal must be enabled.



Important: Efinix tested the example design generated with the default parameter options only.

The example designs target the Titanium Ti60 F225 Development Board. The design implements an AXI Interconnect in the FPGA and demonstrates a single master port access to eight different slave ports. The read and write data is routed by the AXI Interconnect core according to addresses assigned.

Figure 7: AXI Interconnect Example Design



The master issues eight 128-burst of AXI write data each targeting 8 different slaves. Each slave consists of 128-depth data FIFO making each data transaction to fill up the available FIFO. The master issues read operation to all slaves. Then the master compares read data with the write data from each slave accordingly. The development board LEDs output the following:

Table 18: Example Design Output

Output	Result	Description
LED D16 Blue	Test Done	Indicates the test is completed.
LED D16 Green	Test Pass	Indicates the written and read data are matched.
LED D16 Red	Test Fail	Indicates the written and read data are not matched.

Table 19: Slave Ports Address Range

Base Address	Address Width	Base Address Bits Value	Address Start	Address End
0x00000000	28	0x0	0x00000000	0x0FFFFFFF
0x10000000	24	0x10	0x10000000	0x10FFFFFFF
0x11000000	12	0x11000	0x11000000	0x11000FFF
0x11100000	20	0x111	0x11100000	0x111FFFFFFF
0x20000000	28	0x2	0x20000000	0x2FFFFFFF
0x30000000	28	0x3	0x30000000	0x3FFFFFFF
0x40000000	24	0x40	0x40000000	0x40FFFFFFF
0x41000000	20	0x410	0x41000000	0x410FFFFFFF

AXI Interconnect Testbench

You can choose to generate the testbench when generating the core in the IP Manager Configuration window. To generate testbench, the **Testbench Deliverables Option** signals must be enabled.



Note: You must include all **.v** files generated in the **/Testbench** directory in your simulation.



Important: Efinix tested the testbench generated with the default parameter options only.

Efinix provides a simulation script for you to run the testbench quickly using the Modelsim software. To run the Modelsim testbench script, run `vsim -do modelsim.do` in a terminal application. You must have Modelsim installed on your computer to use this script.

The testbench instantiates the IP core and also the simplified AXI master and slave models. The AXI master performs write and read transaction requests to eight different slave destinations. The AXI Interconnect routes the transaction request to the corresponding slave destination. All the written data is read back through read the data channel and compared throughout the simulation.

Revision History

Table 20: Revision History

Date	Document Version	IP Version	Description
June 2025	1.7	5.4	Added description in table Write Address Channel (Slave), Write Data Channel (Slave), Write Response Channel (Master), and Read Data Channel (Master) in Ports topic. (DOC-2562)
November 2024	1.6	5.3	Added Topaz in Device Support. (DOC-2176) Added IP Version in Revision History. (DOC-2185)
July 2024	1.5	-	Added Device Support and release notes sections. (DOC-1234) Added important note in Example Design and Testbench regarding using default parameters options only. (DOC-1781) Replaced table Address Decoder Example with Slave Base Address Setting. Added table Base Address Bits (Decoded Address) and figure Base Address Reference. Updated table Slave Ports Address Range. (DOC-1890)
February 2023	1.4	-	Added note about the resource and performance values in the resource and utilization table are for guidance only.
December 2022	1.3	-	Added New in Version topic.
August 2022	1.2	-	Updated for Efinity IP Manager. Added example design section.
March 2022	1.1	-	Update M_ADDR_WIDTH and added M_BASE_ADDR parameters. (DOC-759)
February 2022	1.0	-	Initial release.