

AXI Interconnect v2.1

LogiCORE IP Product Guide

Vivado Design Suite

PG059 December 20, 2017

Table of Contents

IP Facts

Chapter 1: Overview

AXI Infrastructure Cores	5
Feature Summary	6
Applications	9
AXI Interconnect Core Limitations	9
Licensing and Ordering	10

Chapter 2: Product Specification

Use Models	12
Standards	17
Latency	17
Maximum Performance	20
Resource Utilization	34
Port Descriptions	52
Register Space	70

Chapter 3: Designing with the Core

AXI Interconnect Core Functionality	71
Design Parameters	96
Clocking	108
Resets	108

Chapter 4: Design Flow Steps

Customizing and Generating the Core	110
Constraining the Core	152
Simulation	154
Synthesis and Implementation	155

Chapter 5: Example Design

Appendix A: Upgrading

Migration from CORE Generator System AXI Interconnect v1.x Core	160
Migration from XPS AXI Interconnect v1.x Core	161
Upgrading in the Vivado Design Suite	161

Appendix B: Debugging

Finding Help on Xilinx.com	163
Debug Tools	164

Appendix C: Definitions, Acronyms, and Abbreviations

Appendix D: Additional Resources and Legal Notices

Xilinx Resources	168
References	168
Revision History	169
Please Read: Important Legal Notices	171

Introduction

The Xilinx® LogiCORE™ IP AXI Interconnect core connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices.

Note: The AXI Interconnect core is intended for memory-mapped transfers only. For AXI4-Stream transfers, see the *AXI4-Stream Infrastructure IP Suite LogiCORE IP Product Guide* (PG085) [Ref 1].

Features

The AXI Interconnect core is comprised of multiple LogiCORE IP instances (infrastructure cores). Each of the AXI4 memory-mapped infrastructure cores that comprise the AXI Interconnect core are fully described in this document. The following features apply to the AXI Interconnect core in general and to all infrastructure cores described in this document unless otherwise noted:

- AXI protocol compliant. Can be configured to support AXI3, AXI4, and AXI4-Lite protocols
- Interface data widths:
 - AXI4 and AXI3: 32, 64, 128, 256, 512, or 1,024 bits
 - AXI4-Lite: 32 or 64 bits
- Address width: Up to 64 bits
- USER width (per channel): Up to 1,024 bits
- ID width: Up to 32 bits
- Support for Read-only and Write-only masters and slaves, resulting in reduced resource utilization

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family ⁽¹⁾	UltraScale+™, UltraScale™, 7 Series FPGAs, Zynq®-7000
Supported User Interfaces	AXI4, AXI4-Lite, AXI3
Resources	See Table 2-16 through Table 2-26 .
Provided with Core	
Design Files	Verilog and VHDL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows⁽²⁾	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide .
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

Overview

The AXI Interconnect core can only be added to a Vivado® IP integrator block design in the Vivado Design Suite. The Interconnect IP core represents a hierarchical design block containing multiple LogiCORE™ IP instances (infrastructure cores) that become configured and connected during your system design session. Each of the infrastructure cores can also be added directly to a block design (outside of the AXI Interconnect core) or selected directly from the Vivado IP Catalog and configured for use in an HDL design.

The AXI Interconnect core allows any mixture of AXI master and slave devices to be connected to it, which can vary from one another in terms of data width, clock domain and AXI sub-protocol (AXI4, AXI3, or AXI4-Lite). When the interface characteristics of any connected master or slave device differ from those of the crossbar switch inside the interconnect, the appropriate infrastructure cores are automatically inferred and connected within the interconnect to perform the necessary conversions.

AXI Infrastructure Cores

The following IP cores, described in this document, can be included within each instance of the AXI Interconnect core, depending on the configuration of AXI Interconnect core and its connectivity in the IP integrator block design:

- **AXI Crossbar** connects one or more similar AXI memory-mapped masters to one or more similar memory-mapped slaves.
- **AXI Data Width Converter** connects one AXI memory-mapped master to one AXI memory-mapped slave having a wider or narrower datapath.
- **AXI Clock Converter** connects one AXI memory-mapped master to one AXI memory-mapped slave operating in a different clock domain.
- **AXI Protocol Converter** connects one AXI4, AXI3 or AXI4-Lite master to one AXI slave of a different AXI memory-mapped protocol.
- **AXI Data FIFO** connects one AXI memory-mapped master to one AXI memory-mapped slave through a set of FIFO buffers.
- **AXI Register Slice** connects one AXI memory-mapped master to one AXI memory-mapped slave through a set of pipeline registers, typically to break a critical timing path.
- **AXI MMU** provides address range decoding and remapping services for AXI Interconnect.

Feature Summary

AXI Crossbar

- Each instance of the AXI Interconnect core contains one AXI Crossbar instance (provided it is configured with more than one SI or more than one MI).
- The Slave Interface (SI) of the AXI Crossbar core can be configured to comprise 1-16 SI slots to accept transactions from up to 16 connected master devices. The Master Interface (MI) can be configured to comprise 1-16 MI slots to issue transactions to up to 16 connected slave devices.
- Selectable Interconnect Architecture
 - Crossbar mode (Performance optimized)
 - Shared-Address, Multiple-Data (SAMMD) crossbar architecture.
 - Parallel crossbar pathways for Write data and Read data channels. When more than one Write or Read data source has data to send to different destinations, data transfers can occur independently and concurrently, provided AXI ordering rules are met.
 - Sparse crossbar datapaths according to configured connectivity map, resulting in reduced resource utilization.
 - One shared Write address arbiter, plus one shared Read address arbiter. Arbitration latencies typically do not impact data throughput when transactions average at least three data beats.
 - Crossbar mode is available only when AXI Crossbar is configured for AXI4 or AXI3 protocol.
 - Shared Access mode (Area optimized)
 - Shared write data, shared read data, and single-shared address pathways.
 - Issues one outstanding transaction at a time.
 - Minimizes resource utilization.
- Supports multiple outstanding transactions (crossbar mode)
 - Supports connected masters with multiple reordering depth (ID threads).
 - Supports up to 32-bit wide ID signals with varying ID width per connected master.
 - Supports write response re-ordering, Read data re-ordering, and Read Data interleaving.
 - Configurable Write and Read transaction acceptance limits for each connected master.

- Configurable Write and Read transaction issuing limits for each connected slave.
- Optional single-thread mode (per connected master) reduces thread control logic by allowing one or more outstanding transactions from only one thread ID at a time.
- “Single-Slave per ID” method of cyclic dependency (deadlock) avoidance

For each ID thread issued by a connected master, the Interconnect allows one or more outstanding transactions to only one slave device for Writes and one slave device for Reads, at a time.

- Fixed priority and round-robin arbitration
 - 16 configurable levels of static priority.
 - Round-robin arbitration is used among all connected masters configured with the lowest priority setting (priority 0), when no higher priority master is requesting.
 - Any SI slot that has reached its acceptance limit, or is targeting an MI slot that has reached its issuing limit, or is trying to access an MI slot in a manner that risks deadlock, is temporarily disqualified from arbitration, so that other SI slots can be granted arbitration.
- Supports TrustZone security for each connected slave as a whole
 - If configured as a secure slave device, only secure AXI accesses are permitted.
 - Any non-secure accesses are blocked and the AXI Interconnect core returns a `decerr` response to the connected master.

AXI Data Width Converter

- SI data width: 32, 64, 128, 256, 512 or 1,024 bits.
- MI data width: 32, 64, 128, 256, 512 or 1,024 bits (must be different than SI data width).
- When upsizing, data is packed (merged) when permitted by address channel control signals (CACHE modifiable bit is asserted).
- When downsizing, burst transactions are split into multiple transactions if the maximum burst length would otherwise be exceeded.
- When upsizing, the IP core can optionally perform FIFO buffering and clock frequency conversion (synchronous or asynchronous) in a resource-efficient manner.

AXI Clock Converter

- Synchronous integer-ratio (N:1 and 1:N) conversion for $2 \leq N \leq 16$.
- Asynchronous clock conversion (uses more storage and incurs more latency than synchronous conversion).

AXI Protocol Converter

- AXI4 or AXI3 to AXI4-Lite protocol conversion:
 - Stores AWID and ARID values received on the SI and restores them as BID/RID during response transfers.
 - Converts burst transaction into a series of AXI4-Lite single-beat transfers.
- AXI4 to AXI3 protocol conversion:
 - Splits burst transactions of more than 16 beats from connected AXI4 masters into multiple transactions of no more than 16 beats.
 - Split transactions are single-threaded to enforce in-order merging of response transfers.
- All other conversions tie-off unused signals and incur no intervening logic.

AXI Register Slice

- Individually configurable for each of the 5 AXI channels.
- Facilitates timing closure by trading-off frequency versus latency.
- One latency cycle per register-slice by default.
- Able to propagate AXI traffic with no loss in data throughput (without bubble cycles) under all AXI handshake conditions.
- Optional pipelining to cross Super Logic Region (SLR) in Stacked Silicon Interconnect (SSI) devices. See *Large FPGA Methodology Guide: Including Stacked Silicon Interconnect (SSI) Technology* [Ref 15] for more details.

AXI Data FIFO

- Individually configurable for Write and Read datapaths.
- 32-deep LUT-RAM based.
- 512-deep block RAM based.
- Optional packet FIFO operation to avoid full/empty stalls in the middle of bursts.

AXI MMU

- Defines up to 256 address ranges that represent the address space accessible by a connected endpoint master device.

Applications

Interconnect is general-purpose, and is typically deployed in all systems using AXI memory-mapped transfers.

AXI Interconnect Core Limitations

These limitations apply to the AXI Interconnect core and all applicable infrastructure cores:

- The AXI Interconnect core does not support discontinued AXI3 features:
 - Atomic locked transactions. This feature was retracted by the AXI4 protocol. A locked transaction is changed to a non-locked transaction and propagated by the MI.
 - Write interleaving. This feature was retracted by AXI4 protocol. AXI3 master devices must be configured as if connected to a slave with a Write interleaving depth of one.
- AXI4 Quality of Service (QoS) signals do not influence arbitration priority in AXI Crossbar. QoS signals are propagated from SI to MI.
- AXI Interconnect cores do not support low-power mode or propagate the AXI C channel signals.
- AXI Interconnect cores do not time out if the destination of any AXI channel transfer stalls indefinitely. All connected AXI slaves must respond to all received transactions, as required by AXI protocol.
- AXI Interconnect (AXI Crossbar core) provides no address remapping.
- AXI Interconnect sub-cores do not include conversion or bridging to non-AXI protocols, such as APB.
- AXI Interconnect cores do not have clock-enable (`ac1ken`) inputs. Consequently, the use of `ac1ken` is not supported among memory-mapped AXI interfaces in Xilinx systems.

Note: The `ac1ken` signal is supported for Xilinx AXI4-Stream interfaces.

Licensing and Ordering

This Xilinx® LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the [Xilinx End User License](#). Information about this and other Xilinx LogiCORE IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

Product Specification

Figure 2-1 shows the top-most AXI Interconnect core block diagram. Inside the AXI Interconnect core, a Crossbar core routes traffic between the Slave Interfaces (SI) and Master Interfaces (MI). Along each pathway connecting a SI or MI to the Crossbar, an optional series of AXI Infrastructure cores (couplers) can perform various conversion and buffering functions. The couplers include: Register Slice, Data FIFO, Clock Converter, Data Width Converter and Protocol Converter.

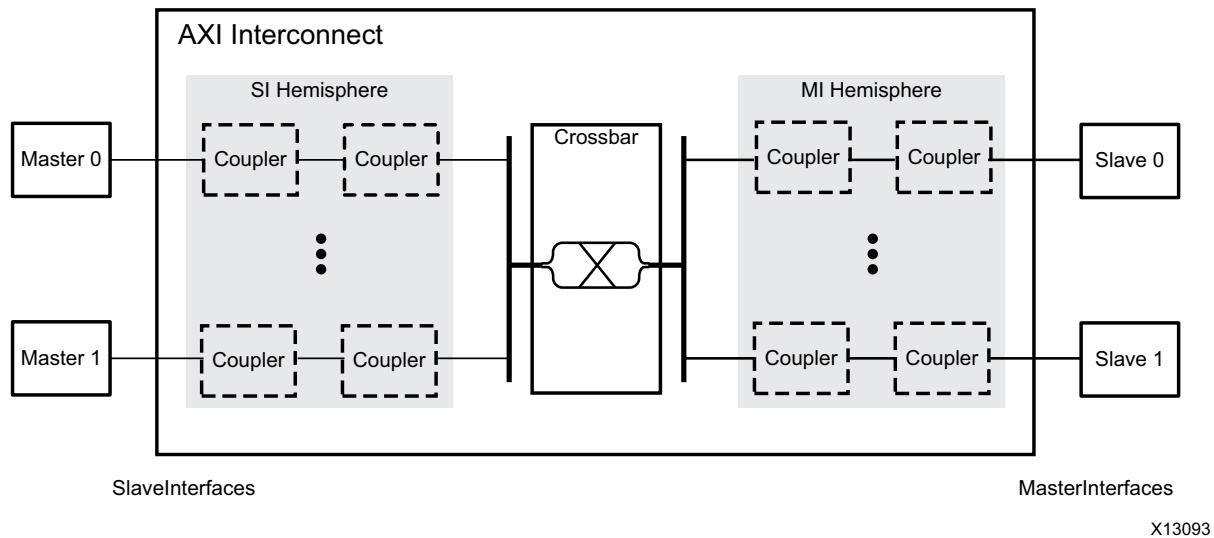


Figure 2-1: AXI Interconnect Core Diagram

The AXI Interconnect core can be configured to have up to 16 Slave Interfaces (SI) and up to 16 Master Interfaces (MI). Each SI connects to one AXI master device to accept Write and Read transaction requests. Each MI connects to one AXI slave device and issues transactions to slave devices. At the center is the *crossbar* core that routes traffic on all the AXI channels between the SI and MI. Along each of the pathways between an SI and the crossbar, or between the crossbar and an MI, there can be one or more infrastructure cores that perform various conversion and storage functions.

Note: When configured to have one Slave Interface, the AXI Interconnect supports up to 64 Master Interfaces.

The crossbar effectively splits the AXI Interconnect core down the middle between the SI-related functional units (*SI hemisphere*) and the MI-related units (*MI hemisphere*).

Use Models

The AXI Interconnect core connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices. Each connected master could be a core that originates AXI transactions (endpoint master) or a master interface of an upstream AXI Interconnect core being cascaded. Each connected slave could be the final target of AXI transactions (endpoint slave) or a slave interface of a downstream AXI Interconnect core being cascaded. A connected master or slave could also be any of the AXI Infrastructure conversion/storage cores, although these functions are typically performed inside the AXI Interconnect core to avoid clutter in the top-level design.

Each AXI Interconnect core can be configured to perform one the following general connectivity patterns:

- [N-to-1 Interconnect](#)
- [1-to-N Interconnect](#)
- [N-to-M Interconnect \(Crossbar Mode\)](#)
- [N-to-M Interconnect \(Shared Access Mode\)](#)

The Interconnect can also be configured to connect one master to one slave, in which case the IP integrator will automate the instantiation and configuration of any couplers that are needed along the pathway.

N-to-1 Interconnect

When multiple master devices arbitrate for access to a single slave device, such as a memory controller, use the AXI Interconnect core in a N-to-1 configuration.

Any of the optional conversion functions, such as data width and clock rate conversion, can also be performed in this configuration as shown in [Figure 2-2](#).

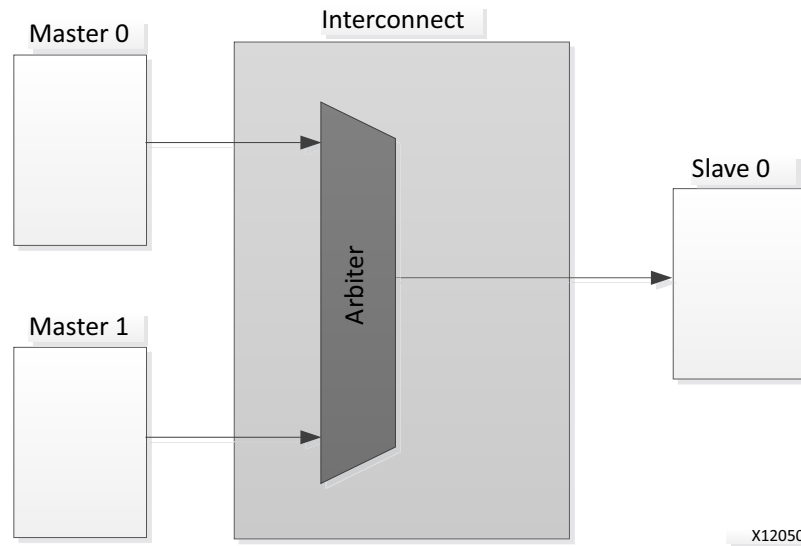


Figure 2-2: N-to-1 AXI Interconnect

1-to-N Interconnect

When a single master device, typically a processor, accesses multiple memory-mapped slave peripherals, use the AXI Interconnect core in a 1-to-N configuration. In these cases, arbitration (in the address and Write datapaths) is not performed, as shown in Figure 2-3.

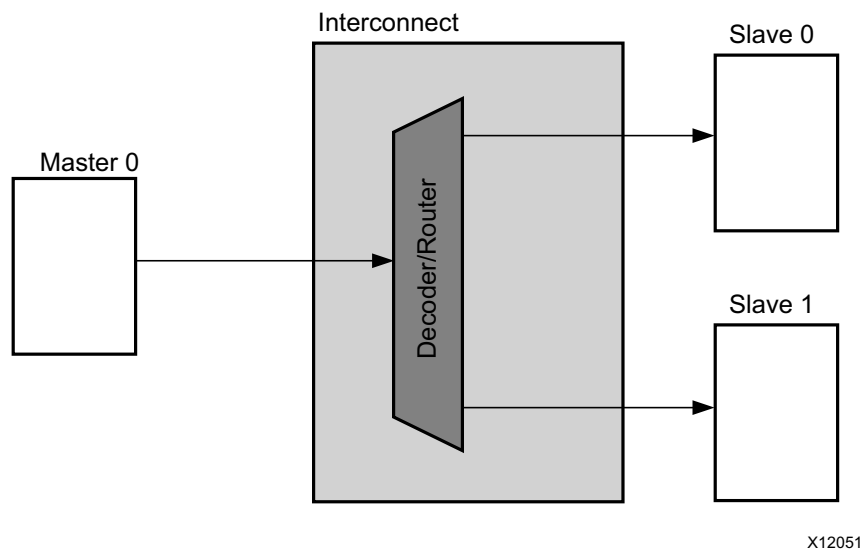
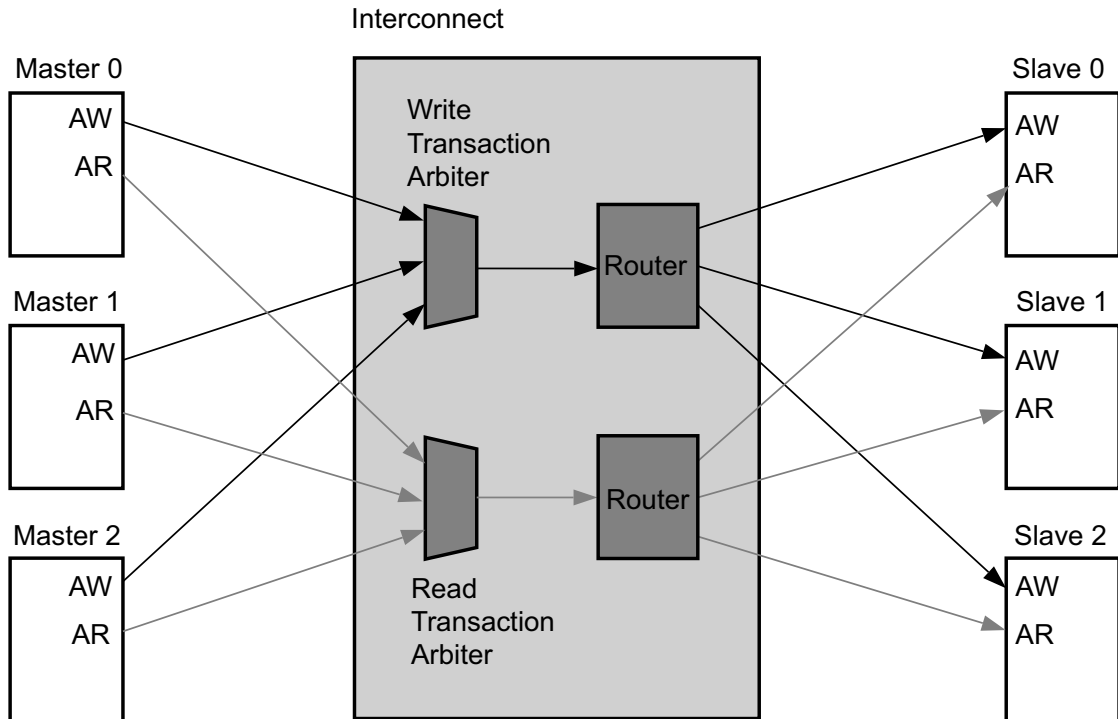


Figure 2-3: 1-to-N AXI Interconnect Use Case

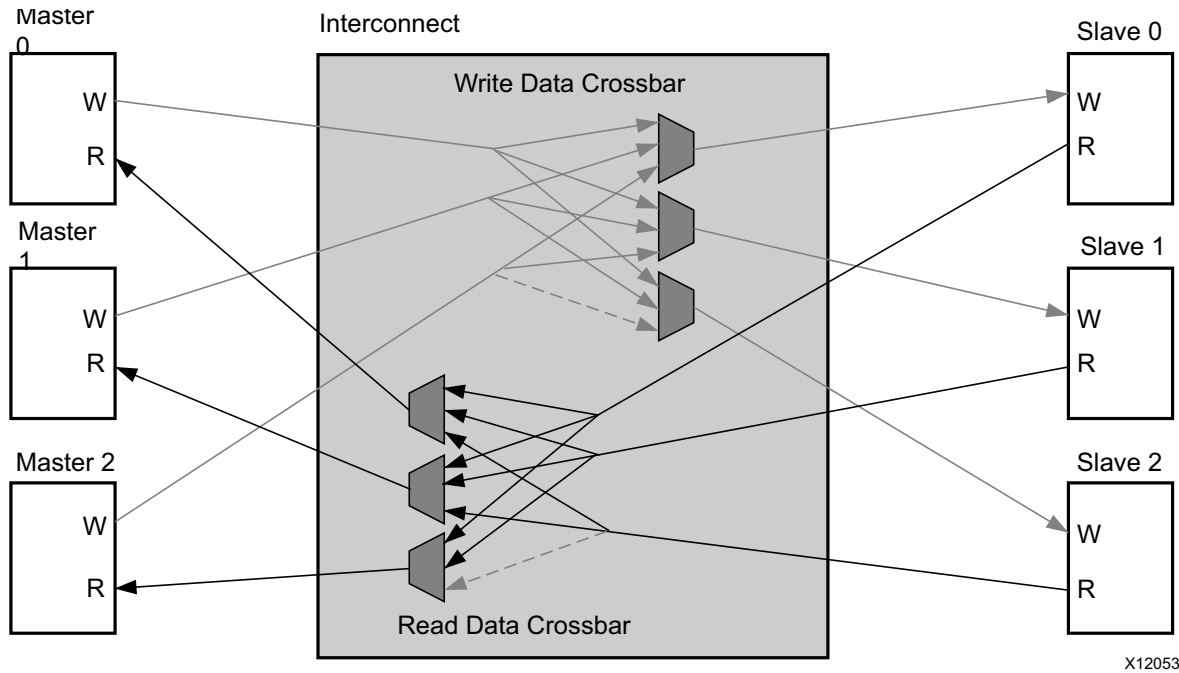
N-to-M Interconnect (Crossbar Mode)

The N-to-M use case of the AXI Interconnect core, when in Crossbar mode, features a Shared-Address Multiple-Data (SAMD) topology, consisting of sparse data crossbar connectivity, with single, shared Write and Read address arbitration, as shown in Figure 2-4 and Figure 2-5.



X12052

Figure 2-4: Shared Write and Read Address Arbitration



X12053

Figure 2-5: Sparse Crossbar Write and Read Data Pathways

Parallel Write and Read data pathways connect each SI slot to all the MI slots that it can access, according to the configured sparse connectivity map. When more than one source has data to send to different destinations, data transfers can occur independently and concurrently, provided AXI ordering rules are met. By disabling unused paths, datapath multiplexing logic and address decoding logic can be reduced, resulting in reduced FPGA resource utilization and faster timing paths.

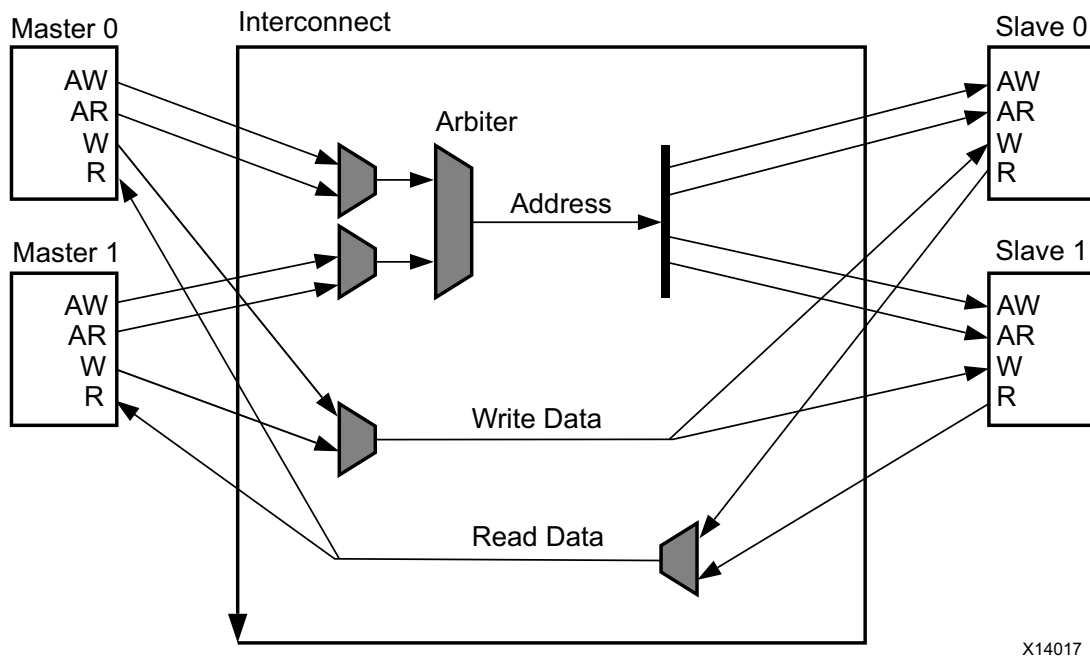
The Write address channels among all SI slots feed into a central address arbiter, which grants access to one SI slot at a time. It is also the case for the Read address channels. The winner of each arbitration cycle transfers its address information to the targeted MI slot, and pushes an entry into the appropriate command queue(s) that enable various data pathways to route data to the proper destination while enforcing AXI ordering rules. Crossbar mode is available only when AXI Crossbar is configured for AXI4 or AXI3 protocol.

N-to-M Interconnect (Shared Access Mode)

When in Shared Access mode, the N-to-M use case of the AXI Interconnect core provides for only one outstanding transaction at a time (single issuing), as shown in Figure 2-6. For each connected master, read transaction requests always take priority over writes. The arbiter then selects from among the requesting masters.

A write or read data transfer is then enabled to the targeted slave device. After the data transfer (including the write response) completes, the next request is arbitrated. Shared Access mode minimizes the resources used to implement the crossbar module of the Interconnect.

Shared Access mode is available when AXI Crossbar is configured for any AXI protocol, but is always used when configured as AXI4-Lite.



X14017

Figure 2-6: Shared Access Mode

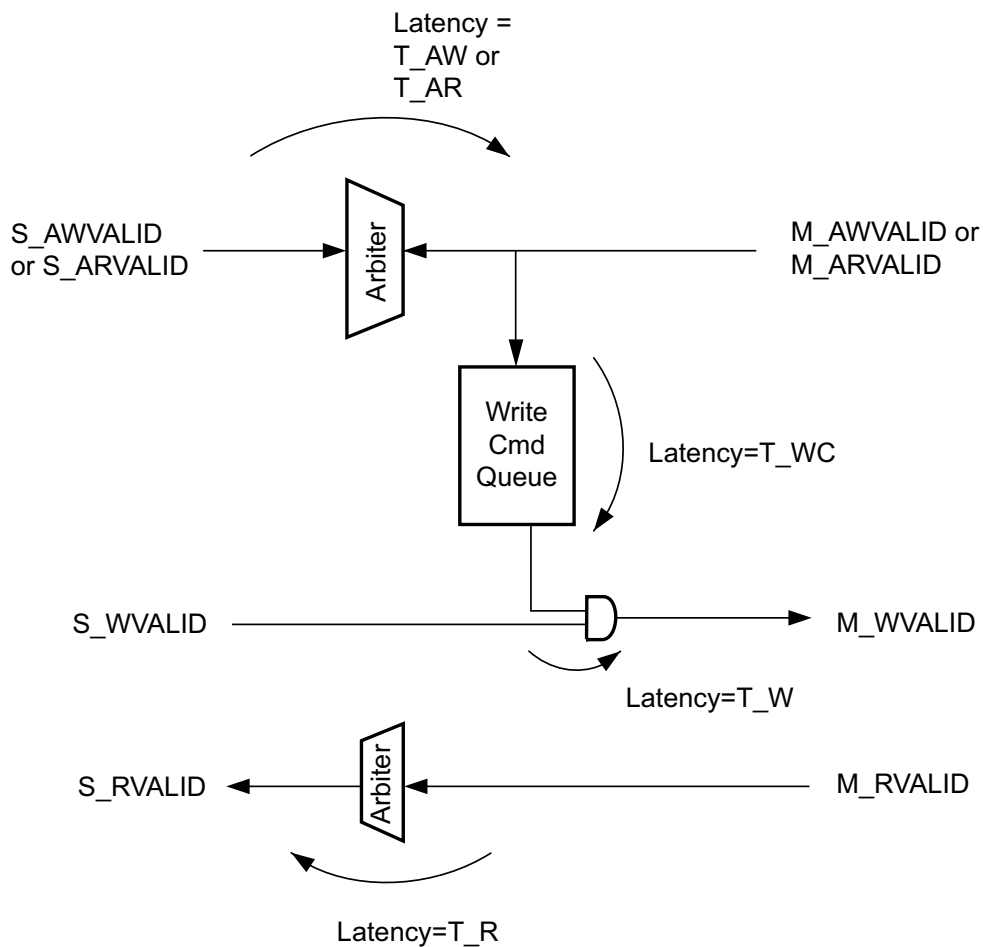
Standards

The AXI interfaces conform to the Advanced Microcontroller Bus Architecture (AMBA®) AXI version 4 specification from Advanced RISC Machine (ARM®), including the AXI4-Lite control register interface subset. See ARM AMBA AXI Protocol v2.0 [Ref 2].

Latency

AXI Crossbar

Figure 2-7 shows the baseline latency model for the crossbar module.



X12091

Figure 2-7: Crossbar Model Baseline Latency

In [Figure 2-7](#), the baseline latency is as follows:

- $T_{AW} = T_{AR} = 2$ cycles of $ac1k$, for the forward propagation of $aw/arvalid$, provided there are no pending conditions that would inhibit granting arbitration (such as a higher-priority request). Each arbitration also causes 2 bubble cycles, resulting in 3 cycles (minimum) between successive arbitrations by the same SI slot.
- $T_{WC} = 1$ cycle of $ac1k$.
- $T_W = 1$ cycle of $ac1k$, with no bubble cycles (supports continuous back-to-back data transmission).
- $T_R = 1$ or 2 cycles of $ac1k$, with no bubble cycles (supports continuous back-to-back data transmission). The second latency cycle occurs when there is a re-arbitration (when the requesting MI slot is different than the last granted MI slot) following an idle cycle. While the same MI slot propagates back-to-back data, or while multiple MI slots continuously interleave data, latency through the R-channel arbiter is 1 cycle.
- T_B (B-channel latency, not shown) = 1 or 2 cycles of $ac1k$. (Same as for T_R .)

AXI Register Slice

- **FULLY_REGISTERED** register slices (each applicable channel): 1 latency cycle with no bubble cycles (best-case 100% channel bandwidth).
- **LIGHT_WEIGHT** register slices (each applicable channel): 1 latency cycle with one bubble cycle (best-case 50% channel bandwidth), which is appropriate for AW, AR, and B channel transfers, and all transfers involving AXI4-Lite endpoints.
- **SI Reg or MI Reg**: One latency cycle, no bubble cycles.
- **SLR Crossing mode and SLR TDM Crossing mode**: 3 latency cycles (of $ac1k$), no bubble cycles.
- **Multi SLR Crossing**: Overall latency varies between 1 and 17 cycles depending on the number of SLR boundaries crossed and the number of pipeline stages configured within each SLR region.

AXI Data FIFO

- W and R channels: Three latency cycles with no bubble cycles.
- AW, AR, and B channels: One latency cycle if Packet mode is enabled; otherwise no latency.

AXI Clock Converter

For synchronous conversion, one cycle of the destination interface clock. (For each AXI channel, the destination is the interface in which the VALID signal for that channel is an output.)

For asynchronous conversion, the latency is the same as the FIFO Generator IP for Non-Built-in FIFOs, Independent Clock and first-word fall-through (FWFT). See the *FIFO Generator LogiCORE IP Product Guide* (PG057) [Ref 3].

AXI Data Width Converter

- AW and AR channels: One latency cycle.
- W channel when upsizing: One latency cycle (for each cycle in which packing completes), with no bubble cycles on the SI-side (narrow) interface.
- R channel when upsizing: One latency cycle.
- B channel: no latency.
- R channel when downsizing: no latency (for each cycle in which packing completes), with no bubble cycles on the MI-side (narrow) interface.
- W channel when downsizing: no latency.

AXI Protocol Converter

- AXI4 or AXI3 to AXI4-Lite conversion: two latency cycles from ARVALID or from (AWVALID and WVALID) when configured in "conversion" mode, no latency on any channels when configured in "unprotected" mode.
- AXI4 to AXI3 conversion:
 - AW and AR channels: One latency cycle.
 - W, R, and B channels: no latency.
- Other conversions: no latency.

AXI MMU

- AW and AR channels: One latency cycle with one bubble cycle.
- W, R, and B channels: No latency.

Maximum Performance

The tables in this section summarize the estimated maximum performance for various modules within the AXI Interconnect core. These values were generated using the Vivado® Design Suite. The values are derived from post-implementation timing reports after implementing the example design generated for each IP configuration. The overall performance of a given instance of AXI Interconnect is limited by the clock frequencies of all constituent modules.

The following devices were used to obtain the results:

- Artix®-7: xc7a200tfbg676-2

These results also apply to Zynq®-7000 devices based on the Artix-7 fabric, as described in DS187 [Ref 12].

- Kintex®-7: xc7k325tffg900-2

These results are also indicative of the expected performance of Virtex®-7 devices.

- Zynq®-7000: xc7z045ffg900-2

These results were combined with the xc7k325-2 results and represent Zynq-7000 devices based on the Kintex-7 fabric, as described in DS191 [Ref 13].

- Kintex UltraScale™: xcku085-flva1517-2

AXI Crossbar Performance: SAMD, AXI4 Protocol

Common Configuration:

- Connectivity Mode: Shared-Address/Multiple-Data (SAMD, as selected by Maximize Performance strategy)
- Protocol: AXI4 or AXI3
- Data Width: 64 or 512 (as noted)
- Read/Write Connectivity: all MI fully connected for read and write
- Thread ID Width: 0 (all SI)
- Address Width: Global = 32; per MI = 16-bit address segment (1 address range)
- Read/write Acceptance: 4
- Read/write Issuing: 8
- Arbitration Priority: 0 (round-robin)
- User Widths: 0

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-1: AXI4 SAMD Crossbar Performance (MHz): Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=350 512b=350	64b=350 512b=350	64b=310 512b=300	64b=300 512b=265	64b=300 512b=265	64b=250 512b=250	64b=250 512b=250	64b=250 512b=250	64b=230 512b=230
2	64b=340 512b=340	64b=335 512b=315	64b=310 512b=310	64b=300 512b=265			64b=250 512b=250			64b=215 512b=215
3	64b=315 512b=310	64b=290 512b=280								
4	64b=300 512b=300	64b=280 512b=265		64b=240 512b=240			64b=230 512b=225			64b=175 512b=175
5	64b=300 512b=300									
6	64b=255 512b=255									
8	64b=250 512b=230	64b=230 512b=215		64b=225 512b=215			64b=205 512b=200			
10	64b=230 512b=230									
12	64b=230 512b=225									
16	64b=205 512b=200	64b=200 512b=200		64b=190 512b=190						

Table 2-2: AXI4 SAMD Crossbar Performance (MHz): Kintex UltraScale, Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=400 512b=400	64b=400 512b=400	64b=400 512b=370	64b=400 512b=370	64b=375 512b=360	64b=345 512b=335	64b=310 512b=280	64b=300 512b=280	64b=270 512b=235
2	64b=400 512b=400	64b=385 512b=385	64b=385 512b=350	64b=360 512b=345			64b=280 512b=280			64b=260 512b=235
3	64b=360 512b=360	64b=360 512b=355								
4	64b=360 512b=360	64b=330 512b=330		64b=320 512b=320			64b=260 512b=240			64b=230 512b=195
5	64b=360 512b=360									
6	64b=345 512b=320									
8	64b=300 512b=300	64b=300 512b=280		64b=280 512b=245			64b=250 512b=190			
10	64b=300 512b=290									
12	64b=290 512b=285									
16	64b=250 512b=250	64b=250 512b=235		64b=235 512b=230						

Table 2-3: AXI4 SAMD Crossbar Performance (MHz): Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=255 512b=250	64b=230 512b=230	64b=215 512b=205	64b=215 512b=180	64b=215 512b=180	64b=165 512b=165	64b=165 512b=165	64b=165 512b=165	64b=155 512b=155
2	64b=230 512b=230	64b=230 512b=215	64b=215 512b=205	64b=200 512b=190			64b=165 512b=165			64b=145 512b=140
3	64b=225 512b=225	64b=190 512b=190								
4	64b=205 512b=200	64b=180 512b=175		64b=155 512b=155			64b=155 512b=145			64b=105 512b=105
5	64b=200 512b=200									
6	64b=175 512b=165									
8	64b=145 512b=145	64b=145 512b=145		64b=145 512b=140			64b=130 512b=120			
10	64b=145 512b=145									
12	64b=145 512b=145									
16	64b=130 512b=130	64b=130 512b=120		64b=120 512b=120						

AXI Crossbar Performance: SASD, AXI4 Protocol

Common Configuration:

- Connectivity Mode: Shared-Address/Shared-Data (SASD, as selected by Minimize Area strategy)
- Protocol: AXI4 or AXI3
- Data Width: 64 or 512 (as noted)
- Read/Write Connectivity: all SI and MI read/write
- Thread ID Width: 0 (all SI)
- Address Width: Global = 32; per MI = 16-bit address segment (1 address range)
- Arbitration Priority: 0 (round-robin)
- User Widths: 0

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-4: AXI4 SASD Crossbar Performance (MHz): Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=350 512b=350	64b=350 512b=350	64b=350 512b=350	64b=350 512b=340	64b=350 512b=315	64b=350 512b=315	64b=350 512b=315	64b=335 512b=310	64b=325 512b=290
2	64b=350 512b=350	64b=350 512b=350	64b=350 512b=350	64b=350 512b=350			64b=350 512b=315			64b=315 512b=290
3	64b=350 512b=350	64b=350 512b=350								
4	64b=350 512b=350	64b=350 512b=350		64b=325 512b=315			64b=325 512b=310			64b=315 512b=275
5	64b=290 512b=290									
6	64b=290 512b=290									
8	64b=290 512b=215	64b=225 512b=215		64b=225 512b=215			64b=215 512b=215			
10	64b=225 512b=215									
12	64b=225 512b=215									
16	64b=225 512b=215	64b=225 512b=215		64b=225 512b=215						

Table 2-5: AXI4 SASD Crossbar Performance (MHz): Kintex UltraScale, Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=400 512b=400	64b=400 512b=400	64b=400 512b=400	64b=400 512b=400	64b=400 512b=390	64b=400 512b=390	64b=400 512b=390	64b=400 512b=380	64b=400 512b=380
2	64b=400 512b=400	64b=400 512b=400	64b=400 512b=400	64b=400 512b=400			64b=400 512b=390			64b=400 512b=360
3	64b=400 512b=400	64b=400 512b=400								
4	64b=400 512b=400	64b=400 512b=400		64b=400 512b=400			64b=400 512b=390			64b=400 512b=360
5	64b=400 512b=360									
6	64b=385 512b=360									
8	64b=370 512b=300	64b=305 512b=295		64b=295 512b=295			64b=290 512b=290			
10	64b=280 512b=280									
12	64b=280 512b=280									
16	64b=280 512b=280	64b=280 512b=280		64b=280 512b=280						

Table 2-6: AXI4 SASD Crossbar Performance (MHz): Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		64b=350 512b=315	64b=350 512b=275	64b=315 512b=240	64b=280 512b=225	64b=280 512b=225	64b=240 512b=215	64b=240 512b=215	64b=225 512b=180	64b=200 512b=180
2	64b=310 512b=310	64b=310 512b=310	64b=310 512b=275	64b=310 512b=240			64b=240 512b=215			64b=200 512b=180
3	64b=300 512b=255	64b=250 512b=240								
4	64b=215 512b=215	64b=215 512b=215		64b=215 512b=215			64b=215 512b=215			64b=200 512b=180
5	64b=190 512b=190									
6	64b=190 512b=190									
8	64b=180 512b=145	64b=140 512b=140		64b=140 512b=140			64b=140 512b=140			
10	64b=140 512b=140									
12	64b=140 512b=140									
16	64b=140 512b=140	64b=140 512b=140		64b=140 512b=140						

AXI Crossbar Performance: SASD, AXI4-Lite Protocol

Common Configuration:

- Connectivity Mode: SASD (Minimize Area strategy)
- Protocol: AXI4-Lite
- Data Width: 32
- Read/Write Connectivity: all SI and MI read/write
- Address Width: Global = 32; per MI = 16-bit address segment (1 address range)
- Arbitration Priority: 0 (round-robin)

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-7: **AXI4-Lite SASD Crossbar Performance (MHz): Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2**

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		350	350	350	350	350	350	350	325	325
2	350	350	350	350			350			325
3	350	350								
4	350	340		335			335			315
5	310									
6	310									
8	240	240		240			240			
10	215									
12	205									
16	205	205		205						

Table 2-8: AXI4-Lite SASD Crossbar Performance (MHz): Kintex UltraScale, Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		400	400	400	400	400	400	400	400	400
2	400	400	400	400			400			400
3	400	400								
4	400	400		400			400			400
5	400									
6	400									
8	295	295		295			295			
10	285									
12	265									
16	265	265		265						

Table 2-9: AXI4-Lite SASD Crossbar Performance (MHz): Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		340	340	315	300	300	250	250	215	190
2	340	315	315	300			250			190
3	280	275								
4	225	225		225			225			190
5	205									
6	200									
8	145	145		145			220			
10	130									
12	130									
16	130	130		130						

AXI Clock Converter

Common Configuration:

- ID Width: 4 (AXI4 and AXI3 only)
- Address Width: 32
- User Width: 0
- Read/Write

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-10: AXI Clock Converter Performance

Protocol	Clock Conversion	Data Width	Performance (MHz)		
			Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
AXI4 or AXI3	Sync or async	32-1024	350	400	270
AXI4-Lite	Sync or async	32-1024	350	400	350

AXI Data FIFO

Common Configuration:

- ID Width: 4 (AXI4 and AXI3 only)
- Address Width: 32
- User Width: 0
- Read/Write
- Protocol: AXI3 or AXI4

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-11: AXI Data FIFO Performance

FIFO Depth	FIFO Delay (Packet Mode)	Data Width	Performance (MHz)		
			Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
32	No	32-1024	350	400	235
512	No	32-1024	350	400	265
512	Yes	32-1024	320	370	195

AXI Data Width Converter

Common Configuration:

- ID Width: 4 (AXI4 and AXI3 only)
- Address Width: 32
- User Width: 0
- Read/Write

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-12: AXI Data Width Converter Performance

Protocol	Width Conversion	FIFO Mode	Clock Conversion	SI Data Width	MI Data Width	Performance (MHz)		
						Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
AXI4 or AXI3	Downsize	n/a	n/a	64-256	< SI width	265	325	175
				512-1024	< SI width	230	295	155
	Upsize	0 (no FIFO)	n/a	< MI width	64	325	380	215
				< MI width	128-256	260	320	175
				< MI width	512-1024	220	295	145
		1 (FIFO, single clock)	n/a	< MI width	64	325	400	220
				< MI width	128-256	295	355	195
				< MI width	512-1024	230	350	155
	2 (FIFO, dual clock)	Sync or async	< MI width	64	325	400	220	
			< MI width	128-256	280	355	190	
< MI width			512-1024	230	325	155		
AXI4-Lite	Downsize	n/a	n/a	64	32	350	400	350
	Upsize	n/a	n/a	32	64	350	400	350

AXI MMU

Common Configuration:

- Protocol: AXI4, AXI3 or AXI4-Lite
- Data Width: 32-1024
- Address Width = 32

- Number of address ranges: 32

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-13: AXI MMU Performance

Performance (MHz)		
Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
350	400	305

AXI Protocol Converter

Common Configuration:

- ID Width: 4 (AXI4 and AXI3 only)
- Address Width: 32
- User Width: 0
- Read/Write

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Note: Protocol conversions other than those listed in the following table incur no logic or storage elements.

Table 2-14: AXI Protocol Converter Performance

Protocol Conversion	Translation Mode	Data Width	Performance (MHz)		
			Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
AXI4 to AXI3	0 (Unprotected)	32-1024	350	400	305
	2 (Conversion)	32-1024	350	400	265
AXI4 or AXI3 to AXI4-Lite	0 (Unprotected)	32-64	350	400	350
	2 (Conversion)	32-64	335	400	230

AXI Register Slice

Common Configuration:

- ID Width: 4 (AXI4 and AXI3 only)
- Address Width: 32
- User Width: 0
- Read/Write
- Type: Fully-registered or Light-weight
- Protocol: AXI4, AXI3 or AXI4-Lite

Note: The specifications in these tables are derived by implementing the configured IP in isolation, characterizing clock frequency based on static timing results, and then applying a 10% guardband below the highest constrained clock frequency that meets timing.

Table 2-15: AXI Register Slice Performance

Performance (MHz)		
Virtex-7 and Kintex-7 (and its Zynq-7000 derivatives), Speed Grade -2	Kintex UltraScale, Speed Grade -2	Artix-7 (and its Zynq-7000 derivatives), Speed Grade -2
350	400	350

Resource Utilization

The tables in this section indicate the estimated FPGA resource utilization for various modules within the AXI Interconnect core. These values were generated using Vivado Design Suite. They are derived from post-synthesis reports, and might change implementation.

Some typical configurations of each module are listed. The overall area of a given instance of AXI Interconnect can be estimated by accumulating the utilizations of all constituent modules.

Note: UltraScale™ architecture results are expected to be similar to 7 series device results.

AXI Crossbar Resource Utilization: SAMD, AXI4 Protocol

Common Configuration

- Connectivity Mode: SAMD (Maximize Performance strategy)
- Protocol: AXI4 or AXI3
- Data Width: 64 (Data width derating factors shown in table.)
- Read/Write Connectivity: all MI fully connected (Read-only and write-only derating factors shown in table.)
- Thread ID Width: 2 (all SI)
- Address Width: Global = 32; per MI = 16 (1 address range)
- Read/write Acceptance: 4
- Read/write Issuing: 8
- Arbitration Priority: 0 (round-robin)
- Single Thread: Disabled
- User Width: 0
- Target device: xc7vx485t

Table Cell Key

LUTs: FFs
 (base_LUTs: base_FFs +
 dw* LUTs_per_bit: FFs_per_bit)
 R = RO_LUTs: RO_FFs%
 W = WO_LUTs: WO_FFs%

- **LUTs**: Slice LUTs utilization for 64-bit Data Width
- **FFs**: Slice Flip-Flops utilization for 64-bit Data Width
- **base_LUTs**: Baseline LUTs, independent of data width
- **base_FFs**: Baseline FFs, independent of data width
- **LUTs_per_bit**: $LUTs_{for_width_DW} = base_LUTs + (DW * LUTs_per_bit)$
- **FFs_per_bit**: $FFs_{for_width_DW} = base_FFs + (DW * FFs_per_bit)$
- **RO_LUTs, RO_FFs**: All SI and MI Read-Only (64-bit data width)
- **WO_LUTs, WO_FFs**: All SI and MI Write-Only (64-bit data width)

Utilization (LUTs) for crossbar with RW_SI and RW_MI read/write slots, RO_SI and RO_MI read-only slots and WO_SI and WO_MI write-only slots (64-bit data width) is approximately (LUTs[(RW_SI + RO_SI), (RW_MI + RO_MI)] * RO_LUTs%) + (LUTs[(RW_SI + WO_SI), (RW_MI + WO_MI)] * WO_LUTs%); similarly for FFs. (SAMD Crossbar shares no resources between read and write pathways; thus RO% + WO% always = 100%.)

Table 2-16: AXI Crossbar Resource Utilization: SAMD, AXI4 Protocol

NUM SI	NUM MI										
	1	2	3	4	5	6	8	10	12	16	
1		410:530 (220:270 + dw* 3.0:4.0) R = 74:75% W = 26:25%	530:700 (270:310 + dw* 4.0:6.0) R = 75:79% W = 25:21%	660:860 (340:350 + dw* 5.0:8.0) R = 75:82% W = 25:18%	850:1030 (310:390 + dw* 8.4:10.0) R = 78:82% W = 22:18%		970:1190 (370:420 + dw* 9.4:12.0) R = 78:84% W = 22:16%	1210:1520 (400:500 + dw* 12.7:16.0) R = 78:85% W = 22:15%	1670:1850 (710:570 + dw* 15.0:20.0) R = 80:86% W = 20:14%	1770:2180 (550:640 + dw* 19.1:24.0) R = 78:87% W = 22:13%	2550:2830 (1080:790 + dw* 23.0:32.0) R = 80:88% W = 20:12%
2	770:540 (610:410 + dw* 2.6:2.0) R = 52:60% W = 48:40%	1050:740 (720:490 + dw* 5.1:4.0) R = 55:66% W = 45:34%	1260:930 (830:550 + dw* 6.8:6.0) R = 55:70% W = 45:30%	1550:1140 (1020:620 + dw* 8.2:8.0) R = 55:72% W = 45:28%			2590:1900 (1190:880 + dw* 21.9:16.0) R = 57:76% W = 43:24%				5410:3420 (2930:1370 + dw* 38.7:32.0) R = 61:80% W = 39:20%
3	1130:630 (890:510 + dw* 3.6:2.0) R = 50:58% W = 50:42%	1590:860 (1120:600 + dw* 7.3:4.0) R = 52:64% W = 48:36%									
4	1380:710 (1120:590 + dw* 4.1:2.0) R = 51:56% W = 49:44%	1920:950 (1390:690 + dw* 8.3:4.0) R = 53:62% W = 47:38%		2570:1380 (1770:870 + dw* 12.5:8.0) R = 51:67% W = 49:33%				4220:2210 (1910:1190 + dw* 36.0:16.0) R = 53:72% W = 47:28%			

Table 2-16: AXI Crossbar Resource Utilization: SAMD, AXI4 Protocol (Cont'd)

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
5	1940:810 (1580:680 + dw* 5.6:2.0) R = 48:55% W = 52:45%									
6	2110:890 (1620:760 + dw* 7.7:2.0) R = 49:54% W = 51:46%									
8	3120:1047 (2467:919 + dw* 10.2:2.0) R = 47:52% W = 53:48%	4320:1340 (3010:1080 + dw* 20.5:4.0) R = 47:56% W = 53:44%		5850:1850 (3790:1340 + dw* 32.3:8.0) R = 44:62% W = 56:38%			9610:2800 (4030:1770 + dw* 87.2:16.0) R = 42:67% W = 58:33%			
10	3960:1220 (3150:1090 + dw* 12.8:2.0) R = 46:52% W = 54:48%									

Table 2-16: AXI Crossbar Resource Utilization: SAMD, AXI4 Protocol (Cont'd)

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
12	4720:1380 (3760:1250 + dw* 14.9:2.0) R = 46:50% W = 54:50%									
16	6320:1700 (5070:1570 + dw* 19.5:2.0) R = 46:49% W = 54:51%	8370:2100 (5900:1840 + dw* 38.7:4.0) R = 46:52% W = 54:48%								

AXI Crossbar Resource Utilization: SASD, AXI4 Protocol

Common Configuration

- Connectivity Mode: SASD (Minimize Area strategy)
- Protocol: AXI4 or AXI3
- Data Width: 64 (Data width derating factors shown in table.)
- Read/Write Connectivity: all SI and MI read/write (Read-only and write-only derating factors shown in table.)
- Thread ID Width: 2 (all SI)
- Address Width: Global = 32; per MI = 16 (1 address range)
- Read/write Acceptance: 1
- Read/write Issuing: 1
- Arbitration Priority: 0 (round-robin)
- User Width: 0
- Target device: xc7vx485t

Table Cell Key

LUTs: FFs
 (base_LUTs: base_FFs +
 dw* LUTs_per_bit: FFs_per_bit)
 R = RO_LUTs: RO_FFs%
 W = WO_LUTs: WO_FFs%

- **LUTs**: Slice LUTs utilization for 64-bit Data Width
- **FFs**: Slice Flip-Flops utilization for 64-bit Data Width
- **base_LUTs**: Baseline LUTs, independent of data width
- **base_FFs**: Baseline FFs, independent of data width
- **LUTs_per_bit**: $LUTs_{for_width_DW} = base_LUTs + (DW * LUTs_per_bit)$
- **FFs_per_bit**: $FFs_{for_width_DW} = base_FFs + (DW * FFs_per_bit)$
- **RO_LUTs, RO_FFs**: All SI and MI Read-Only (64-bit data width)
- **WO_LUTs, WO_FFs**: All SI and MI Write-Only (64-bit data width)

SASD Crossbar shares resources between read and write pathways; thus RO% + WO% is generally not 100%. Resource determination for a crossbar with a mixture of read-write, read-only and write-only slots is not straightforward, but generally falls between the resources listed for the table cells for read-capable slots [(RW_SI + RO_SI), (RW_MI + RO_MI)] and write-capable slots [(RW_SI + WO_SI), (RW_MI + WO_MI)].

Table 2-17: AXI Crossbar Resource Utilization: SASD, AXI4 Protocol

	1	2	3	4	5	6	8	10	12	16
1		220:100 (150:100 + dw* 1.0:0) R = 57:90% W = 22:83%	230:100 (160:100 + dw* 1.0:0) R = 59:90% W = 24:83%	320:100 (190:100 + dw* 2.0:0) R = 66:90% W = 23:84%	360:110 (240:110 + dw* 1.8:0) R = 71:90% W = 23:84%	330:110 (120:110 + dw* 3.3:0) R = 67:91% W = 27:84%	560:110 (240:110 + dw* 5.0:0) R = 80:91% W = 20:84%	660:110 (270:110 + dw* 6.0:0) R = 81:91% W = 21:85%	690:110 (300:110 + dw* 6.0:0) R = 81:91% W = 24:85%	1220:120 (340:120 + dw* 13.8:0) R = 86:92% W = 19:86%
2	320:110 (250:110 + dw* 1.1:0) R = 58:89% W = 35:82%	370:110 (270:110 + dw* 1.6:0) R = 62:89% W = 34:83%	380:110 (280:110 + dw* 1.6:0) R = 64:89% W = 35:83%	470:110 (300:110 + dw* 2.5:0) R = 69:89% W = 32:83%			720:120 (360:120 + dw* 5.6:0) R = 78:90% W = 27:84%			1380:130 (450:130 + dw* 14.4:0) R = 84:90% W = 22:85%
3	470:110 (360:110 + dw* 1.6:0) R = 55:87% W = 40:81%	510:120 (370:120 + dw* 2.2:0) R = 59:87% W = 39:81%								
4	560:120 (450:120 + dw* 1.6:0) R = 55:86% W = 36:81%	600:120 (460:120 + dw* 2.1:0) R = 58:87% W = 36:81%		700:120 (500:120 + dw* 3.1:0) R = 63:87% W = 34:81%			950:130 (550:130 + dw* 6.2:0) R = 71:88% W = 30:82%			
5	840:120 (680:120 + dw* 2.5:0) R = 52:85% W = 49:80%									

Table 2-17: AXI Crossbar Resource Utilization: SASD, AXI4 Protocol (Cont'd)

	1	2	3	4	5	6	8	10	12	16
6	850:130 (590:130 + dw* 4.1:0) R = 54:84% W = 41:79%									
8	1380:140 (1010:140 + dw* 5.7:0) R = 55:83% W = 53:78%	1420:140 (1020:140 + dw* 6.2:0) R = 57:83% W = 53:78%		1520:140 (1060:140 + dw* 7.2:0) R = 59:83% W = 51:78%			1770:150 (1120:150 + dw* 10.2:0) R = 64:84% W = 46:79%			
10	1750:150 (1280:150 + dw* 7.3:0) R = 51:81% W = 55:77%									
12	2070:160 (1530:160 + dw* 8.4:0) R = 51:80% W = 54:76%									
16	2720:180 (2040:180 + dw* 10.7:0) R = 51:78% W = 54:74%	2770:180 (2050:180 + dw* 11.2:0) R = 51:78% W = 53:74%								

AXI Crossbar Resource Utilization: SASD, AXI4-Lite Protocol

Common Configuration

- Connectivity Mode: SASD (Minimize Area strategy)
- Protocol: AXI4-Lite
- Data Width: 32
- Read/Write Connectivity: all SI and MI read/write (Read-only and write-only derating factors shown in table.)
- Address Width: Global = 32; per MI = 16 (1 address range)
- Read/write Acceptance: 1
- Read/write Issuing: 1
- Arbitration Priority: 0 (round-robin)
- Target device: xc7vx485t

Table Cell Key

LUTs: FFs
 R = RO_LUTs: RO_FF%
 W = WO_LUTs: WO_FF%

- **LUTs**: Slice LUTs utilization
- **FFs**: Slice Flip-Flops utilization
- **RO_LUTs, RO_FF**s: All SI and MI Read-Only
- **WO_LUTs, WO_FF**s: All SI and MI Write-Only

SASD Crossbar shares resources between read and write pathways; thus RO% + WO% is generally not 100%. Resource determination for a crossbar with a mixture of read-write, read-only and write-only slots is not straightforward, but generally falls between the resources listed for the table cells for read-capable slots [(RW_SI + RO_SI), (RW_MI + RO_MI)] and write-capable slots [(RW_SI + WO_SI), (RW_MI + WO_MI)].

Table 2-18: AXI Crossbar Resource Utilization: SASD, AXI4-Lite Protocol

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
1		140:60 R = 54:89% W = 29:89%	150:70 R = 54:89% W = 34:89%	200:70 R = 63:90% W = 34:90%	220:70 R = 62:90% W = 35:90%	220:70 R = 65:90% W = 37:90%	350:70 R = 76:90% W = 31:90%	410:70 R = 78:91% W = 33:91%	370:80 R = 74:91% W = 40:91%	790:80 R = 82:91% W = 28:91%
2	200:70 R = 57:87% W = 37:87%	230:70 R = 62:88% W = 37:88%	250:70 R = 63:88% W = 40:88%	300:80 R = 65:88% W = 38:88%			450:80 R = 76:89% W = 35:89%			890:90 R = 81:90% W = 31:90%
3	290:80 R = 56:84% W = 41:84%	330:80 R = 60:85% W = 42:85%								
4	360:80 R = 56:84% W = 38:84%	390:80 R = 59:84% W = 39:84%		450:90 R = 61:85% W = 38:85%			610:90 R = 69:86% W = 36:86%			
5	490:90 R = 54:83% W = 43:83%									
6	540:90 R = 54:82% W = 41:82%									
8	780:100 R = 49:79% W = 46:79%	810:100 R = 51:80% W = 46:80%		870:110 R = 53:80% W = 45:80%			1030:110 R = 59:81% W = 43:81%			
10	1100:110 R = 53:78% W = 54:78%									

Table 2-18: AXI Crossbar Resource Utilization: SASD, AXI4-Lite Protocol (Cont'd)

NUM SI	NUM MI									
	1	2	3	4	5	6	8	10	12	16
12	1310:120 R = 54:76% W = 52:76%									
16	1750:140 R = 54:74% W = 52:74%	1780:150 R = 54:74% W = 52:74%								

AXI Clock Converter Resource Utilization

Common Configuration:

- ID Width: 2 (AXI4 and AXI3 only)
- Address Width: 22
- User Width: 0
- Target device: xc7vx485t

Table 2-19: AXI Clock Converter Resource Utilization

Protocol	Clock Conversion	Data Width	Read/Write		Read-Only		Write-Only	
			LUTs	FFs	LUTs	FFs	LUTs	FFs
AXI4, AXI3	Sync (Any Ratio)	32	110	300	50	140	60	160
		64	180	430	80	210	100	230
		128	320	700	150	330	170	370
		256	590	1250	280	590	310	660
		512	1130	2340	530	1100	600	1240
		1024	2220	4510	1040	2130	1180	2390
	Async	32	300	860	130	380	170	480
		64	350	990	150	440	190	550
		128	440	1260	200	570	240	700
		256	620	1810	280	820	340	980
		512	980	2900	450	1340	530	1560
		1024	1710	5070	790	2360	910	2710
AXI4-Lite	Sync (Any Ratio)	32	40	160	10	80	20	80
		64	40	230	10	110	20	120
	Async	32	270	730	110	310	150	420
		64	320	870	140	380	180	490

AXI Data FIFO Resource Utilization

Common Configuration:

- Protocol: AXI4 or AXI3
- ID Width: 2
- Address Width: 22
- USER Width: 0
- Target device: xc7vx485t

Table 2-20: AXI Data FIFO Resource Utilization

FIFO Configuration	Data Width	Read/Write				Read-Only (Write FIFO Depth = 0)				Write-Only (Read FIFO Depth = 0)			
		LUTs	FFs	18k BRAM	36k BRAM	LUTs	FFs	18k BRAM	36k BRAM	LUTs	FFs	18k BRAM	36k BRAM
32-deep (LUT RAM)	32	110	230	0	0	60	110	0	0	50	120	0	0
	64	150	360	0	0	80	180	0	0	70	180	0	0
	128	250	640	0	0	120	310	0	0	130	330	0	0
	256	430	1180	0	0	200	560	0	0	230	620	0	0
	512	790	2270	0	0	380	1070	0	0	410	1200	0	0
	1024	1510	4440	0	0	720	2100	0	0	790	2340	0	0
512-deep (BRAM)	32	100	190	0	2	50	90	0	1	50	100	0	1
	64	100	250	1	2	50	130	0	1	50	120	1	1
	128	100	390	1	4	50	190	0	2	50	200	1	2
	256	100	660	1	8	50	320	0	4	50	340	1	4
	512	100	1210	2	15	50	570	1	7	50	640	1	8
	1024	100	2290	2	30	50	1090	1	14	50	1200	1	16

Table 2-20: AXI Data FIFO Resource Utilization (Cont'd)

FIFO Configuration	Data Width	Read/Write				Read-Only (Write FIFO Depth = 0)				Write-Only (Read FIFO Depth = 0)			
		LUTs	FFs	18k BRAM	36k BRAM	LUTs	FFs	18k BRAM	36k BRAM	LUTs	FFs	18k BRAM	36k BRAM
512-deep, Packet-Mode	32	300	620	0	2	160	310	0	1	140	310	0	1
	64	300	690	1	2	160	340	0	1	140	350	1	1
	128	300	830	1	4	160	410	0	2	140	420	1	2
	256	300	1100	1	8	160	540	0	4	140	560	1	4
	512	300	1640	2	15	160	790	1	7	140	850	1	8
	1024	300	2730	2	30	160	1300	1	14	140	1430	1	16

AXI Data Width Converter Resource Utilization: AXI4 Upsizer

Common Configuration:

- Protocol: AXI4 or AXI3
- SI ID Width: 2
- Address Width: 22
- Target device: xc7vx485t

Table 2-21: AXI Data Width Converter Resource Utilization: AXI4 Upsizer

FIFO Mode	Clock Conversion	MI Data Width	SI Data Width	Read/Write				Read-Only			Write-Only		
				LUTs	FFs	18k BRAM	36k BRAM	LUTs	FFs	18k BRAM	LUTs	FFs	36k BRAM
No FIFO	None	64	32	580	590	0	0	270	320	0	310	270	0
		128	32–64	830	940	0	0	390	520	0	440	420	0
		256	32–128	1300	1620	0	0	600	910	0	700	710	0
		512	32–256	2070	2980	0	0	1010	1690	0	1060	1290	0
		1024	32–512	4010	5680	0	0	1820	3230	0	2190	2450	0
Packet FIFO	None	64	32	860	810	2	1	470	440	2	390	370	1
		128	32–64	1030	870	4	2	490	450	4	540	420	2
		256	32–128	1430	970	8	4	530	460	8	900	510	4
		512	32–256	2040	1140	16	8	600	470	16	1440	670	8
		1024	32–512	3280	1480	32	16	650	470	32	2630	1010	16
Packet FIFO with Clock Conversion	Sync (Any Ratio)	64	32	860	820	2	1	470	440	2	390	380	1
		128	32–64	1040	880	4	2	490	460	4	550	420	2
		256	32–128	1430	980	8	4	540	460	8	890	520	4
		512	32–256	2060	1160	16	8	600	470	16	1460	690	8
		1024	32–512	3290	1500	32	16	650	480	32	2640	1020	16
	Async	64	32	980	1280	2	1	510	640	2	470	640	1
		128	32–64	1140	1340	4	2	520	650	4	620	690	2
		256	32–128	1520	1440	8	4	560	650	8	960	790	4
		512	32–256	2160	1620	16	8	630	660	16	1530	960	8
		1024	32–512	3390	1960	32	16	680	670	32	2710	1290	16

AXI Data Width Converter Resource Utilization: AXI4 Downsizer

Common Configuration:

- Protocol: AXI4 or AXI3
- SI ID Width: 2
- Address Width: 22
- Target device: xc7vx485t

AXI Data Width Converter Resource Utilization: AXI4-Lite

Table 2-22: AXI Data Width Converter Resource Utilization: AXI4 Downsizer

SI Data Width	MI Data Width	Read/Write		Read-Only		Write-Only	
		LUTs	FFs	LUTs	FFs	LUTs	FFs
64	32	660	730	310	360	350	370
128	32–64	850	810	440	440	410	370
256	32–128	920	960	470	570	450	390
512	32–256	1160	1230	630	840	530	390
1024	64–512	1560	1750	880	1360	680	390

Common Configuration:

- Protocol: AXI4-Lite
- Address Width: 22
- Target device: xc7vx485t

Table 2-23: AXI Data Width Converter Resource Utilization: AXI4-Lite

SI Data Width	MI Data Width	Read/Write		Read-Only		Write-Only	
		LUTs	FFs	LUTs	FFs	LUTs	FFs
32	64	40	10	20	10	20	10
64	32	110	60	30	40	60	10

AXI MMU

Common Configuration:

- Protocol: AXI4, AXI3 or AXI4-Lite
- Address Width = 32
- Number of address ranges: 32
- Read/Write

Table 2-24: LUTs and FFs

LUTs	FFs
630	180

AXI Protocol Converter Resource Utilization

Common Configuration:

- ID Width: 2 (Converting from AXI4 and AXI3 only)
- Address Width: 22
- User Width: 0
- Target device: xc7vx485t

Table 2-25: AXI Protocol Converter Resource Utilization

Protocol Conversion	Translation Mode	Data Width	Read/Write		Read-Only		Write-Only	
			LUTs	FFs	LUTs	FFs	LUTs	FFs
AXI4 to AXI3	0 (Unprotected: Generate WID only)	32	50	110	0	0	50	110
		64	50	110	0	0	50	110
		128	50	110	0	0	50	110
		256	50	110	0	0	50	110
		512	50	110	0	0	50	110
		1024	50	110	0	0	50	110
	2 (Conversion: Long burst Splitting)	32	320	410	130	160	190	250
		64	320	410	130	160	190	250
		128	320	410	130	160	190	250
		256	320	410	130	160	190	250
		512	320	410	130	160	190	250
		1024	320	410	130	160	190	250

Table 2-25: AXI Protocol Converter Resource Utilization (Cont'd)

Protocol Conversion	Translation Mode	Data Width	Read/Write		Read-Only		Write-Only	
			LUTs	FFs	LUTs	FFs	LUTs	FFs
AXI4 or AXI3 to AXI4-Lite	0 (Unprotected: Store IDs)	32	40	10	10	10	10	10
		64	40	10	10	10	10	10
	2 (Conversion: Burst-to-singles)	32	500	470	290	270	220	200
		64	590	580	380	390	220	200
All Others	N/A	Any	0	0	0	0	0	0

AXI Register Slice Resource Utilization

Common Configuration:

- ID Width: 2 (AXI4 and AXI3 only)
- Read/Write Mode: Read/Write
- Address Width: 22
- User Width: 0
- Target device: xc7vx485t

Utilization is shown for each AXI channel. Total utilization is the sum of all enabled channels based on their Register Slice Option settings. (BYPASS uses no resources.)

Table 2-26: AXI Register Slice Resource Utilization

Protocol	AXI Channel	Data Width	FULL		LIGHT	
			LUTs	FFs	LUTs	FFs
AXI4 or AXI3	R	32	50	80	10	40
		64	80	140	10	70
		128	140	270	10	140
		256	270	530	10	270
		512	530	1040	10	520
		1024	1040	2060	10	1030
	W	32	50	80	10	40
		64	80	150	10	80
		128	150	300	10	150
		256	300	580	10	290
		512	590	1160	10	580
		1024	1160	2310	10	1160
	AR	N/A	60	110	10	60
AW	N/A	60	110	10	60	
B	N/A	10	10	10	10	
AXI4-Lite	R	32			10	40
		64			10	70
	W	32			10	40
		64			10	80
	AR	N/A			10	30
	AW	N/A			10	30
	B	N/A			10	10

Port Descriptions

This section lists the interface signals for the AXI Interconnect core and each of the underlying AXI infrastructure cores.

In Table 2-27 through Table 2-35, the Default column shows whether the input signal is required (REQ) or, if not, its default value if left unconnected. For the AXI Interconnect core and the AXI Crossbar core, signal connections are required only for the SI and MI that are

used. Signals that are not used in a particular protocol configuration are indicated by “d/c” (do not care).

AXI Interconnect Core I/O Signals

Table 2-27 lists the Slave Interface signals for the AXI Interconnect core. In the Signal Name column “nn” represents a two-digit sequence number (with leading zero) with range $00 \leq nn \leq N-1$, where N refers to the total number of configured Slave Interfaces, which is the number of master devices connected to the AXI Interconnect core. Each row in the table therefore defines N interface signals. When a range of values is specified in the Width column, the signal width is determined by the tools based on system connectivity.

Slave Interface I/O Signals

Table 2-27: AXI Interconnect Core Slave I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
Snn_ACLK	Input	REQ	1	Slave interface clock input
Snn_ARESETN	Input	REQ	1	Slave interface reset input (active-Low)
Snn_AXI_AWID	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–32]	Write Address Channel Transaction ID
Snn_AXI_AWADDR	Input	REQ	[12–64]	Write Address Channel Address
Snn_AXI_AWLEN	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 8 AXI3: 4	Write Address Channel Burst Length (0–255)
Snn_AXI_AWSIZE	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	3	Write Address Channel Transfer Size code (0–7)
Snn_AXI_AWBURST	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	2	Write Address Channel Burst Type code (0–2).
Snn_AXI_AWLOCK	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 1 AXI3: 2	Write Address Channel Atomic Access Type (0, 1)
Snn_AXI_AWCACHE	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	4	Write Address Channel Cache Characteristics
Snn_AXI_AWPROT	Input	0b000 ⁽³⁾	3	Write Address Channel Protection Bits
Snn_AXI_AWQOS ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	4	AXI4 Write Address Channel Quality of Service
Snn_AXI_AWUSER	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–1024]	User-defined AW Channel signals
Snn_AXI_AWVALID	Input	REQ	1	Write Address Channel Valid
Snn_AXI_AWREADY	Output		1	Write Address Channel Ready

Table 2-27: AXI Interconnect Core Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
Snn_AXI_WID	Input	AXI3: 0 AXI4, AXI4-Lite: d/c	[1–32]	Write Data Channel Transaction ID for AXI3 masters
Snn_AXI_WDATA	Input	REQ	[32, 64, 128, 256, 512, 1024]	Write Data Channel Data
Snn_AXI_WSTRB	Input	all ones	[32, 64, 128, 256, 512, 1024] / 8	Write Data Channel Byte Strobes
Snn_AXI_WLAST	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	1	Write Data Channel Last Data Beat
Snn_AXI_WUSER	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–1024]	User-defined W Channel signals
Snn_AXI_WVALID	Input	REQ	1	Write Data Channel Valid.
Snn_AXI_WREADY	Output		1	Write Data Channel Ready.
Snn_AXI_BID	Output		[1–32]	Write Response Channel Transaction ID.
Snn_AXI_BRESP	Output		2	Write Response Channel Response Code (0–3).
Snn_AXI_BUSER	Output		[1–1024]	User-defined B Channel signals.
Snn_AXI_BVALID	Output		1	Write Response Channel Valid.
Snn_AXI_BREADY	Input	REQ	1	Write Response Channel Ready.
Snn_AXI_ARID	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–32]	Read Address Channel Transaction ID.
Snn_AXI_ARADDR	Input	REQ	[12–64]	Read Address Channel Address.
Snn_AXI_ARLEN	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 8 AXI3: 4	Read Address Channel Burst Length code (0–255).
Snn_AXI_ARSIZE	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	3	Read Address Channel Transfer Size code (0–7).
Snn_AXI_ARBURST	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	2	Read Address Channel Burst Type (0–2).
Snn_AXI_ARLOCK	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 1 AXI3: 2	Read Address Channel Atomic Access Type (0, 1).
Snn_AXI_ARCACHE	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	4	Read Address Channel Cache Characteristics.

Table 2-27: AXI Interconnect Core Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
Snn_AXI_ARPROT	Input	0b000 ⁽³⁾	3	Read Address Channel Protection Bits.
Snn_AXI_ARQOS ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	4	AXI4 Read Address Channel Quality of Service.
Snn_AXI_ARUSER	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–1024]	User-defined AR Channel signals.
Snn_AXI_ARVALID	Input	REQ	1	Read Address Channel Valid.
Snn_AXI_ARREADY	Output		1	Read Address Channel Ready.
Snn_AXI_RID	Output		[1–32]	Read Data Channel Transaction ID.
Snn_AXI_RDATA	Output		[32, 64, 128, 256, 512, 1024]	Read Data Channel Data.
Snn_AXI_RRESP	Output		2	Read Data Channel Response Code (0–3).
Snn_AXI_RLAST	Output		1	Read Data Channel Last Data Beat.
Snn_AXI_RUSER	Output		[1–1024]	User-defined R Channel signals.
Snn_AXI_RVALID	Output		1	Read Data Channel Valid.
Snn_AXI_RREADY	Input	REQ	1	Read Data Channel Ready.

Notes:

1. Xilinx recommends that AXI4 master devices drive their *aw/rsize* and *aw/rburst* outputs. Typically, a master device drives an *aw/rsize* value that corresponds to its interface data width, unless application requirements dictate otherwise. Typically, a master device drives its *aw/rburst* output to 0b01, which indicates an incremental (INCR) burst.
2. Xilinx recommends that master devices drive their *aw/rcache* outputs to 0b0011 to allow the AXI Interconnect core to pack data while performing width conversion.
3. AXI protocol requires master devices to drive their *aw/rprot* output. If the *aw/rprot* signals are left undriven, it defaults to all zeros and the transaction is interpreted as secure.
4. Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any SI slot configured as AXI3.

Master Interface I/O Signals

Table 2-28 lists the Master Interface signals for the AXI Interconnect core. In the Signal Name column “mm” represents a two-digit sequence number (with leading zero) with range $00 \leq mm \leq M-1$, where M refers to the total number of configured Master Interfaces, which is the number of slave devices connected to the AXI Interconnect core. Each row in the table therefore defines M interface signals. When a range of values is specified in the Width column, the signal width is determined by the tools based on system connectivity.

Table 2-28: AXI Interconnect Core Master I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
Mmm_ACLK	Input	REQ	1	Master interface clock input
Mmm_ARESETN	Input	REQ	1	Master interface reset input (active-Low)
Mmm_AXI_AWID	Output		[1–32]	Write Address Channel Transaction ID.
Mmm_AXI_AWADDR	Output		[12–64]	Write Address Channel Address.
Mmm_AXI_AWLEN	Output		AXI4: 8 AXI3: 4	Write Address Channel Burst Length code. (0–255).
Mmm_AXI_AWSIZE	Output		3	Write Address Channel Transfer Size code (0–7).
Mmm_AXI_AWBURST	Output		2	Write Address Channel Burst Type (0–2).
Mmm_AXI_AWLOCK	Output		AXI4: 1 AXI3: 2	Write Address Channel Atomic Access Type (0, 1).
Mmm_AXI_AWCACHE	Output		4	Write Address Channel Cache Characteristics.
Mmm_AXI_AWPROT	Output		3	Write Address Channel Protection Bits
Mmm_AXI_AWREGION	Output		4	AXI4 Write Address Channel address region index.
Mmm_AXI_AWQOS ⁽¹⁾	Output		4	Write Address Channel Quality of Service.
Mmm_AXI_AWUSER	Output		[1–1024]	User-defined AW Channel signals.
Mmm_AXI_AWVALID	Output		1	Write Address Channel Valid.
Mmm_AXI_AWREADY	Input	REQ	1	Write Address Channel Ready.

Table 2-28: AXI Interconnect Core Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
Mmm_AXI_WID	Output		[1–32]	Write Data Channel Transaction ID for AXI3 slaves
Mmm_AXI_WDATA	Output		[32, 64, 128, 256, 512, 1024]	Write Data Channel Data.
Mmm_AXI_WSTRB	Output		[32, 64, 128, 256, 512, 1024] / 8	Write Data Channel Data Byte Strobes.
Mmm_AXI_WLAST	Output		1	Write Data Channel Last Data Beat.
Mmm_AXI_WUSER	Output		[1–1024]	User-defined W Channel signals.
Mmm_AXI_WVALID	Output		1	Write Data Channel Valid.
Mmm_AXI_WREADY	Input	REQ	1	Write Data Channel Ready.
Mmm_AXI_BID	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	[1–32]	Write Response Channel Transaction ID.
Mmm_AXI_BRESP	Input	0b00	2	Write Response Channel Response Code (0–3).
Mmm_AXI_BUSER	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–1024]	User-defined B Channel signals.
Mmm_AXI_BVALID	Input	REQ	1	Write Response Channel Valid.
Mmm_AXI_BREADY	Output		1	Write Response Channel Ready.
Mmm_AXI_ARID	Output		[1–32]	Read Address Channel Transaction ID.
Mmm_AXI_ARADDR	Output		[12–64]	Read Address Channel Address.
Mmm_AXI_ARLEN	Output		AXI4: 8 AXI3: 4	Read Address Channel Burst Length code (0–255).
Mmm_AXI_ARSIZE	Output		3	Read Address Channel Transfer Size code (0–7).
Mmm_AXI_ARBURST	Output		2	Read Address Channel Burst Type (0–2).
Mmm_AXI_ARLOCK	Output		AXI4: 1 AXI3: 2	Read Address Channel Atomic Access Type (0,1).

Table 2-28: AXI Interconnect Core Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
Mmm_AXI_ARCACHE	Output		4	Read Address Channel Cache Characteristics.
Mmm_AXI_ARPROT	Output		3	Read Address Channel Protection Bits.
Mmm_AXI_ARREGION	Output		4	AXI4 Read Address Channel address region index.
Mmm_AXI_ARQOS ⁽¹⁾	Output		4	AXI4 Read Address Channel Quality of Service.
Mmm_AXI_ARUSER	Output		[1–1024]	User-defined AR Channel signals.
Mmm_AXI_ARVALID	Output		1	Read Address Channel Valid.
Mmm_AXI_ARREADY	Input	REQ	1	Read Address Channel Ready.
Mmm_AXI_RID	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	[1–32]	Read Data Channel Transaction ID.
Mmm_AXI_RDATA	Input	REQ	[32, 64, 128, 256, 512, 1024]	Read Data Channel Data.
Mmm_AXI_RRESP	Input	0b00	2	Read Data Channel Response Code (0–3).
Mmm_AXI_RLAST	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	1	Read Data Channel Last Data Beat.
Mmm_AXI_RUSER	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	[1–1024]	User-defined R Channel signals.
Mmm_AXI_RVALID	Input	REQ	1	Read Data Channel Valid.
Mmm_AXI_RREADY	Output		1	Read Data Channel Ready.

Notes:

1. Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any MI slot configured as AXI3.

Table 2-29: AXI Interconnect Core Global Port Signals

Port Signal Name	Direction	Default	Width	Description (Range)
ACLK	Input	REQ	1	Crossbar clock input.
ARESETN	Input	REQ	1	Crossbar Reset (active-Low).

AXI Crossbar Core I/O Signals

Table 2-30 lists the Slave Interface signals for the AXI Crossbar core. In the Width column “N” refers to the total number of configured SI slots, which is the number of master devices connected to the AXI Crossbar core.

Slave I/O Signals

Table 2-30: AXI Crossbar Slave I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
s_axi_awid	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*ID_WIDTH	Write Address Channel Transaction ID
s_axi_awaddr	Input	REQ	N*ADDR_WIDTH	Write Address Channel Address
s_axi_awlen	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: N*8 AXI3: N*4	Write Address Channel Burst Length (0–255)
s_axi_awsz	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	N*3	Write Address Channel Transfer Size code (0–7)
s_axi_awburst	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	N*2	Write Address Channel Burst Type code (0–2).
s_axi_awlock	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: N*1 AXI3: N*2	Write Address Channel Atomic Access Type (0, 1)
s_axi_awcache	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	N*4	Write Address Channel Cache Characteristics
s_axi_awprot	Input	0b000 ⁽³⁾	N*3	Write Address Channel Protection Bits
s_axi_awqos ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	N*4	AXI4 Write Address Channel Quality of Service
s_axi_awuser	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*AWUSER_WIDTH	User-defined AW Channel signals
s_axi_awvalid	Input	REQ	N*1	Write Address Channel Valid
s_axi_awready	Output		N*1	Write Address Channel Ready
s_axi_wid	Input	AXI3: 0 AXI4, AXI4-Lite: d/c	N*ID_WIDTH	Write Data Channel Transaction ID for AXI3 masters
s_axi_wdata	Input	REQ	N*DATA_WIDTH	Write Data Channel Data
s_axi_wstrb	Input	all ones	N*DATA_WIDTH/8	Write Data Channel Byte Strobes
s_axi_wlast	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*1	Write Data Channel Last Data Beat

Table 2-30: AXI Crossbar Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
s_axi_wuser	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*WUSER_WIDTH	User-defined W Channel signals
s_axi_wvalid	Input	REQ	N*1	Write Data Channel Valid.
s_axi_wready	Output		N*1	Write Data Channel Ready.
s_axi_bid	Output		N*ID_WIDTH	Write Response Channel Transaction ID.
s_axi_bresp	Output		N*2	Write Response Channel Response Code (0–3).
s_axi_buser	Output		N*BUSER_WIDTH	User-defined B Channel signals.
s_axi_bvalid	Output		N*1	Write Response Channel Valid.
s_axi_bready	Input	REQ	N*1	Write Response Channel Ready.
s_axi_arid	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*ID_WIDTH	Read Address Channel Transaction ID.
s_axi_araddr	Input	REQ	N*ADDR_WIDTH	Read Address Channel Address.
s_axi_arlen	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: N*8 AXI3: N*4	Read Address Channel Burst Length code (0–255).
s_axi_arsize	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	N*3	Read Address Channel Transfer Size code (0–7).
s_axi_arburst	Input	AXI3, AXI4: REQ ⁽¹⁾ Lite: d/c	N*2	Read Address Channel Burst Type (0–2).
s_axi_arlock	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: N*1 AXI3: N*2	Read Address Channel Atomic Access Type (0, 1).
s_axi_arcache	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	N*4	Read Address Channel Cache Characteristics.
s_axi_arprot	Input	0b000 ⁽³⁾	N*3	Read Address Channel Protection Bits.
s_axi_arqos ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	N*4	AXI4 Read Address Channel Quality of Service.
s_axi_aruser	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	N*ARUSER_WIDTH	User-defined AR Channel signals.
s_axi_arvalid	Input	REQ	N*1	Read Address Channel Valid.
s_axi_arready	Output		N*1	Read Address Channel Ready.
s_axi_rid	Output		N*ID_WIDTH	Read Data Channel Transaction ID.

Table 2-30: AXI Crossbar Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
s_axi_rdata	Output		N*DATA_WIDTH	Read Data Channel Data.
s_axi_rresp	Output		N*2	Read Data Channel Response Code (0–3).
s_axi_rlast	Output		N*1	Read Data Channel Last Data Beat.
s_axi_ruser	Output		N*RUSER_WIDTH	User-defined R Channel signals.
s_axi_rvalid	Output		N*1	Read Data Channel Valid.
s_axi_rready	Input	REQ	N*1	Read Data Channel Ready.

Notes:

1. Xilinx recommends that AXI4 master devices drive their `aw/rsize` and `aw/rburst` outputs. Typically, a master device drives an `aw/rsize` value that corresponds to its interface data width, unless application requirements dictate otherwise. Typically, a master device drives its `aw/rburst` output to 0b01, which indicates an incremental (INCR) burst.
2. Xilinx recommends that master devices drive their `aw/rcache` outputs to 0b0011 to allow the AXI Interconnect core to pack data while performing width conversion and to allow store-and-forward in datapath FIFOs.
3. AXI protocol requires master devices to drive their `aw/rprot` output. If the `aw/rprot` signals are left undriven, it would default to all zeros and the transaction would be interpreted as secure.
4. Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any SI slot configured as AXI3.

Master I/O Signals

Table 2-31 lists the Master Interface signals for the AXI Crossbar core. In the Width column of Table 2-31, “M” refers to the total number of configured Master Interface (MI) slots, which is the number of slave devices connected to the AXI Crossbar core.

Table 2-31: AXI Crossbar Master I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
m_axi_awid	Output		M*ID_WIDTH	Write Address Channel Transaction ID.
m_axi_awaddr	Output		M*ADDR_WIDTH	Write Address Channel Address.
m_axi_awlen	Output		AXI4: M*8 AXI3: M*4	Write Address Channel Burst Length code. (0–255).
m_axi_awsz	Output		M*3	Write Address Channel Transfer Size code (0–7).
m_axi_awburst	Output		M*2	Write Address Channel Burst Type (0–2).
m_axi_awlock	Output		AXI4: M*1 AXI3: M*2	Write Address Channel Atomic Access Type (0, 1).

Table 2-31: AXI Crossbar Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_awcache	Output		M*4	Write Address Channel Cache Characteristics.
m_axi_awprot	Output		M*3	Write Address Channel Protection Bits
m_axi_awregion	Output		M*4	AXI4 Write Address Channel address region index.
m_axi_awqos ⁽¹⁾	Output		M*4	Write Address Channel Quality of Service.
m_axi_awuser	Output		M*AWUSER_WIDTH	User-defined AW Channel signals.
m_axi_awvalid	Output		M*1	Write Address Channel Valid.
m_axi_awready	Input	REQ	M*1	Write Address Channel Ready.
m_axi_wid	Output		M*ID_WIDTH	Write Data Channel Transaction ID for AXI3 slaves.
m_axi_wdata	Output		M*DATA_WIDTH	Write Data Channel Data.
m_axi_wstrb	Output		M*DATA_WIDTH/8	Write Data Channel Data Byte Strobes.
m_axi_wlast	Output		1	Write Data Channel Last Data Beat.
m_axi_wuser	Output		M*WUSER_WIDTH	User-defined W Channel signals.
m_axi_wvalid	Output		M*1	Write Data Channel Valid.
m_axi_wready	Input	REQ	M*1	Write Data Channel Ready.
m_axi_bid	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	M*ID_WIDTH	Write Response Channel Transaction ID.
m_axi_bresp	Input	0b00	M*2	Write Response Channel Response Code (0–3).
m_axi_buser	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	M*BUSER_WIDTH	User-defined B Channel signals.
m_axi_bvalid	Input	REQ	M*1	Write Response Channel Valid.
m_axi_bready	Output		M*1	Write Response Channel Ready.
m_axi_arid	Output		M*ID_WIDTH	Read Address Channel Transaction ID.

Table 2-31: AXI Crossbar Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_araddr	Output		M*ADDR_WIDTH	Read Address Channel Address.
m_axi_arlen	Output		AXI4: M*8 AXI3: M*4	Read Address Channel Burst Length code (0–255).
m_axi_arsize	Output		M*3	Read Address Channel Transfer Size code (0–7).
m_axi_arburst	Output		M*2	Read Address Channel Burst Type (0–2).
m_axi_arlock	Output		AXI4: M*1 AXI3: M*2	Read Address Channel Atomic Access Type (0,1).
m_axi_arcache	Output		M*4	Read Address Channel Cache Characteristics.
m_axi_arprot	Output		M*3	Read Address Channel Protection Bits.
m_axi_arregion	Output		M*4	AXI4 Read Address Channel address region index.
m_axi_arqos ⁽¹⁾	Output		M*4	AXI4 Read Address Channel Quality of Service.
m_axi_aruser	Output		M*ARUSER_WIDTH	User-defined AR Channel signals.
m_axi_arvalid	Output		M*1	Read Address Channel Valid.
m_axi_arready	Input	REQ	M*1	Read Address Channel Ready.
m_axi_rid	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	M*ID_WIDTH	Read Data Channel Transaction ID.
m_axi_rdata	Input	REQ	M*DATA_WIDTH	Read Data Channel Data.
m_axi_rresp	Input	0b00	M*2	Read Data Channel Response Code (0–3).
m_axi_rlast	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	M*1	Read Data Channel Last Data Beat.
m_axi_ruser	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	M*RUSER_WIDTH	User-defined R Channel signals.
m_axi_rvalid	Input	REQ	M*1	Read Data Channel Valid.

Table 2-31: AXI Crossbar Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_rready	Output		M*1	Read Data Channel Ready.

Notes:

- Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any MI slot configured as AXI3.

Table 2-32: AXI Crossbar Global Port Signals

Port Signal Name	Direction	Default	Width	Description (Range)
aclk	Input	REQ	1	clock input.
aresetn	Input	REQ	1	Global Reset (active-Low).

Other AXI Infrastructure Core I/O Signals

This section defines the I/O Interface signals for the following AXI Infrastructure cores:

- [AXI Data Width Converter](#)
- [AXI Clock Converter](#)
- [AXI Protocol Converter](#)
- [AXI Data FIFO](#)
- [AXI Register Slice](#)
- [AXI MMU](#)

Slave I/O Signals

Table 2-33 lists the Slave Interface signals for the cores.

Table 2-33: AXI Infrastructure Core Slave I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
s_axi_aclk	Input	REQ	1	Slave interface clock input. AXI Clock Converter and Data Width Converter core only.
s_axi_aresetn	Input	REQ	1	Slave interface reset input (active-Low). AXI Clock Converter and Data Width Converter core only.
s_axi_awid	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	ID_WIDTH	Write Address Channel Transaction ID
s_axi_awaddr	Input	REQ	ADDR_WIDTH	Write Address Channel Address

Table 2-33: AXI Infrastructure Core Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
s_axi_awlen	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 8 AXI3: 4	Write Address Channel Burst Length (0–255)
s_axi_awsz	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	3	Write Address Channel Transfer Size code (0–7)
s_axi_awburst	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	2	Write Address Channel Burst Type code (0–2).
s_axi_awlock	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 1 AXI3: 2	Write Address Channel Atomic Access Type (0, 1)
s_axi_awcache	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	4	Write Address Channel Cache Characteristics
s_axi_awprot	Input	0b000 ⁽³⁾	3	Write Address Channel Protection Bits
s_axi_awqos ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	4	AXI4 Write Address Channel Quality of Service
s_axi_awregion	Input	AXI4: 0; AXI3, AXI4-Lite: d/c	4	AXI4 Write Address Channel address region index
s_axi_awuser ⁽⁵⁾	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AWUSER_WIDTH	User-defined AW Channel signals
s_axi_awvalid	Input	REQ	1	Write Address Channel Valid
s_axi_awready	Output		1	Write Address Channel Ready
s_axi_wid	Input	AXI3: 0 AXI4, AXI4-Lite: d/c	ID_WIDTH	Write Data Channel Transaction ID for AXI3 masters
s_axi_wdata	Input	REQ	Data Width Converter: S_AXI_DATA_WIDTH; Others: DATA_WIDTH	Write Data Channel Data
s_axi_wstrb	Input	all ones	Data Width Converter: S_AXI_DATA_WIDTH/8; Others: DATA_WIDTH/8	Write Data Channel Byte Strokes
s_axi_wlast	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	1	Write Data Channel Last Data Beat
s_axi_wuser ⁽⁵⁾	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	WUSER_WIDTH	User-defined W Channel signals
s_axi_wvalid	Input	REQ	1	Write Data Channel Valid.

Table 2-33: AXI Infrastructure Core Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
s_axi_wready	Output		1	Write Data Channel Ready.
s_axi_bid	Output		ID_WIDTH	Write Response Channel Transaction ID.
s_axi_bresp	Output		2	Write Response Channel Response Code (0–3).
s_axi_buser ⁽⁵⁾	Output		BUSER_WIDTH	User-defined B Channel signals.
s_axi_bvalid	Output		1	Write Response Channel Valid.
s_axi_bready	Input	REQ	1	Write Response Channel Ready.
s_axi_arid	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	ID_WIDTH	Read Address Channel Transaction ID.
s_axi_araddr	Input	REQ	ADDR_WIDTH	Read Address Channel Address.
s_axi_arlen	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 8 AXI3: 4	Read Address Channel Burst Length code (0–255).
s_axi_arsize	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	3	Read Address Channel Transfer Size code (0–7).
s_axi_arburst	Input	AXI3, AXI4: REQ ⁽¹⁾ AXI4-Lite: d/c	2	Read Address Channel Burst Type (0–2).
s_axi_arlock	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	AXI4: 1 AXI3: 2	Read Address Channel Atomic Access Type (0, 1).
s_axi_arcache	Input	AXI3, AXI4: 0 ⁽²⁾ AXI4-Lite: d/c	4	Read Address Channel Cache Characteristics.
s_axi_arprot	Input	0b000 ⁽³⁾	3	Read Address Channel Protection Bits.
s_axi_arregion	Input	AXI4: 0; AXI3, AXI4-Lite: d/c	4	AXI4 Read Address Channel address region index
s_axi_arqos ⁽⁴⁾	Input	AXI4: 0 AXI4-Lite: d/c	4	AXI4 Read Address Channel Quality of Service.
s_axi_aruser ⁽⁵⁾	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	ARUSER_WIDTH	User-defined AR Channel signals.
s_axi_arvalid	Input	REQ	1	Read Address Channel Valid.
s_axi_arready	Output		1	Read Address Channel Ready.

Table 2-33: AXI Infrastructure Core Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
s_axi_rid	Output		ID_WIDTH	Read Data Channel Transaction ID.
s_axi_rdata	Output		Data Width Converter: S_AXI_DATA_WIDTH; Others: DATA_WIDTH	Read Data Channel Data.
s_axi_rresp	Output		2	Read Data Channel Response Code (0–3).
s_axi_rlast	Output		1	Read Data Channel Last Data Beat.
s_axi_ruser ⁽⁵⁾	Output		RUSER_WIDTH	User-defined R Channel signals.
s_axi_rvalid	Output		1	Read Data Channel Valid.
s_axi_rready	Input	REQ	1	Read Data Channel Ready.

Notes:

1. Xilinx recommends that AXI4 master devices drive their `aw/rsize` and `aw/rburst` outputs. Typically, a master device drives an `aw/rsize` value that corresponds to its interface data width, unless application requirements dictate otherwise. Typically, a master device drives its `aw/rburst` output to 0b01, which indicates an incremental (INCR) burst.
2. Xilinx recommends that master devices drive their `aw/rcache` outputs to 0b0011 to allow the AXI Interconnect core to pack data while performing width conversion and to allow store-and-forward in datapath FIFOs.
3. AXI protocol requires master devices to drive their `aw/rprot` output. If the `aw/rprot` signals are left undriven, it would default to all zeros and the transaction would be interpreted as secure.
4. Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any SI slot configured as AXI3.
5. Signal not present on Data Width Converter core.

Master I/O Signals

Table 2-34 lists the Master Interface signals for the cores.

Table 2-34: AXI Infrastructure Core Master I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
m_axi_aclk	Input	REQ	1	Master interface clock input. AXI Clock Converter and Data Width Converter core only.
m_axi_aresetn	Input	REQ	1	Master interface reset input (active-Low). AXI Clock Converter and Data Width Converter core only.
m_axi_awid ⁽¹⁾	Output		ID_WIDTH	Write Address Channel Transaction ID.
m_axi_awaddr	Output		ADDR_WIDTH	Write Address Channel Address.

Table 2-34: AXI Infrastructure Core Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_awlen	Output		AXI4: 8 AXI3: 4	Write Address Channel Burst Length code. (0–255).
m_axi_awsz	Output		3	Write Address Channel Transfer Size code (0–7).
m_axi_awburst	Output		2	Write Address Channel Burst Type (0–2).
m_axi_awlock	Output		AXI4: 1 AXI3: 2	Write Address Channel Atomic Access Type (0, 1).
m_axi_awcache	Output		4	Write Address Channel Cache Characteristics.
m_axi_awprot	Output		3	Write Address Channel Protection Bits
m_axi_awregion	Output		4	AXI4 Write Address Channel address region index.
m_axi_awqos ⁽²⁾	Output		4	Write Address Channel Quality of Service.
m_axi_awuser ⁽¹⁾	Output		AWUSER_WIDTH	User-defined AW Channel signals.
m_axi_awvalid	Output		1	Write Address Channel Valid.
m_axi_awready	Input	REQ	1	Write Address Channel Ready.
m_axi_wid ⁽¹⁾	Output		ID_WIDTH	Write Data Channel Transaction ID for AXI3 slaves
m_axi_wdata	Output		Data Width Converter: M_AXI_DATA_WIDTH; Others: DATA_WIDTH	Write Data Channel Data.
m_axi_wstrb	Output		Data Width Converter: M_AXI_DATA_WIDTH/8; Others: DATA_WIDTH/8	Write Data Channel Data Byte Strokes.
m_axi_wlast	Output		1	Write Data Channel Last Data Beat.
m_axi_wuser ⁽¹⁾	Output		WUSER_WIDTH	User-defined W Channel signals.
m_axi_wvalid	Output		1	Write Data Channel Valid.
m_axi_wready	Input	REQ	1	Write Data Channel Ready.

Table 2-34: AXI Infrastructure Core Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_bid ⁽¹⁾	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	ID_WIDTH	Write Response Channel Transaction ID.
m_axi_bresp	Input	0b00	2	Write Response Channel Response Code (0–3).
m_axi_buser ⁽¹⁾	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	BUSER_WIDTH	User-defined B Channel signals.
m_axi_bvalid	Input	REQ	1	Write Response Channel Valid.
m_axi_bready	Output		1	Write Response Channel Ready.
m_axi_arid ⁽¹⁾	Output		ID_WIDTH	Read Address Channel Transaction ID.
m_axi_araddr	Output		ADDR_WIDTH	Read Address Channel Address.
m_axi_arlen	Output		AXI4: 8 AXI3: 4	Read Address Channel Burst Length code (0–255).
m_axi_arsize	Output		3	Read Address Channel Transfer Size code (0–7).
m_axi_arburst	Output		2	Read Address Channel Burst Type (0–2).
m_axi_arlock	Output		AXI4: 1 AXI3: 2	Read Address Channel Atomic Access Type (0,1).
m_axi_arcache	Output		4	Read Address Channel Cache Characteristics.
m_axi_arprot	Output		3	Read Address Channel Protection Bits.
m_axi_arregion	Output		4	AXI4 Read Address Channel address region index.
m_axi_arqos ⁽²⁾	Output		4	AXI4 Read Address Channel Quality of Service.
m_axi_aruser ⁽¹⁾	Output		ARUSER_WIDTH	User-defined AR Channel signals.
m_axi_arvalid	Output		1	Read Address Channel Valid.
m_axi_arready	Input	REQ	1	Read Address Channel Ready.
m_axi_rid ⁽¹⁾	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	ID_WIDTH	Read Data Channel Transaction ID.

Table 2-34: AXI Infrastructure Core Master I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
m_axi_rdata	Input	REQ	Data Width Converter: M_AXI_DATA_WIDTH; Others: DATA_WIDTH	Read Data Channel Data.
m_axi_rresp	Input	0b00	2	Read Data Channel Response Code (0–3).
m_axi_rlast	Input	AXI3, AXI4: REQ AXI4-Lite: d/c	1	Read Data Channel Last Data Beat.
m_axi_ruser ⁽¹⁾	Input	AXI3, AXI4: 0 AXI4-Lite: d/c	RUSER_WIDTH	User-defined R Channel signals.
m_axi_rvalid	Input	REQ	1	Read Data Channel Valid.
m_axi_rready	Output		1	Read Data Channel Ready.

Notes:

- Signal not present on Data Width Converter core.
- Although the QOS signals are defined only by the AXI4 protocol specification, this interconnect IP core also propagates QOS signals for any MI slot configured as AXI

Table 2-35: AXI Infrastructure Core Global Port Signals

Port Signal Name	Direction	Default	Width	Description (Range)
aclk	Input	REQ	1	Clock input. Except AXI Clock Converter and Data Width Converter core.
aresetn	Input	REQ	1	Global Reset (active-Low). Except AXI Clock Converter and Data Width Converter core.
aclk2x	Input	REQ	1	This auxiliary clock input is only enabled when one or more AXI channels are configured in SLR TDM Crossing mode. The input must be exactly twice the frequency of aclk and should be generated from the same clock source with zero phase shift.

Register Space

None of the cores described in this document contain any memory-mapped control or status registers.

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

AXI Interconnect Core Functionality

These subsections describe the functionality within the AXI Interconnect core.

- [AXI Crossbar Functionality](#)
- [Width Conversion](#)
- [Clock Conversion](#)
- [Protocol Conversion](#)
- [AXI Register Slices](#)
- [AXI Data FIFO](#)

AXI Crossbar Functionality

Each instance of AXI Interconnect core contains one AXI Crossbar instance, whenever there is more than one connected master or more than one connected slave. It is recommended that the AXI Crossbar be deployed automatically by instantiating the AXI Interconnect core in an IP integrator design, rather than being instantiated directly (stand-alone) in a design.

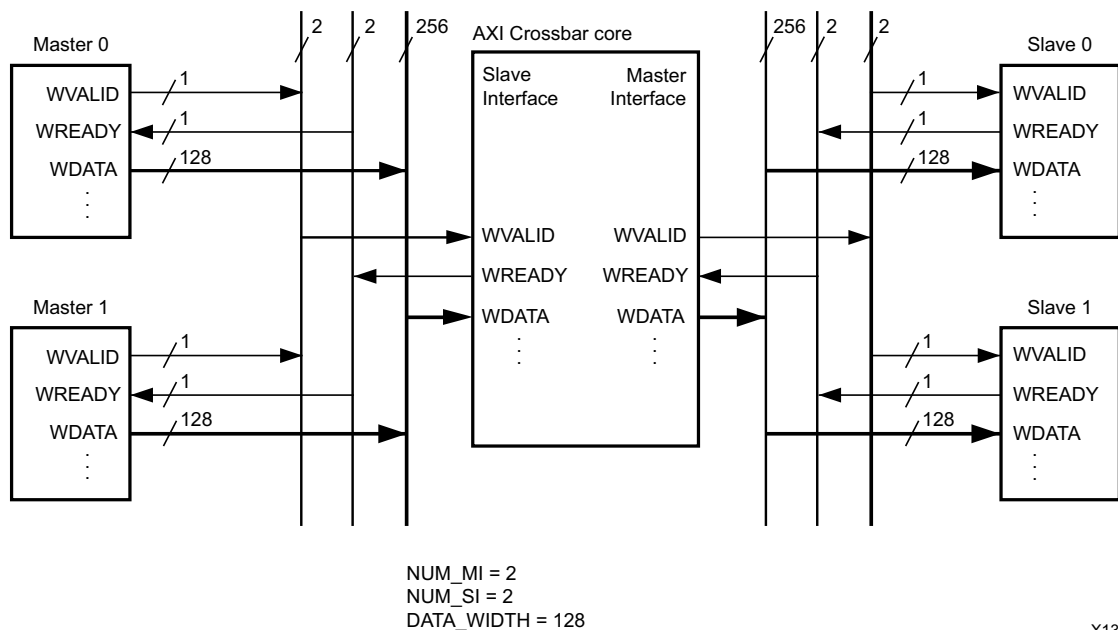
Crossbar Signal Interface

The interface of the AXI Crossbar core consists of a single, vectored AXI SI, plus a single, vectored AXI MI. Each vectored interface can be configured to connect to between 1 and 16 master/slave devices. The pathways connecting to all the Slave Interfaces of the AXI Interconnect core are merged together to connect to the vectored SI interface of the AXI Crossbar. The pathways connecting to all the Master Interfaces of the AXI Interconnect core are merged together to connect to the vectored MI interface of the AXI Crossbar.

For each signal comprising a vectored AXI interface on the AXI Crossbar core, its natural width is multiplied by the number of devices to which it is connected. All of the bit slices that connect to a single device are referred to as a *slot* of the interface. For example, the `awsize` signal carries a 3-bit value indicating the number of bytes transferred during each data beats in a Write transaction. If the AXI Crossbar core is configured with two SI slots, the `s_axi_awsizes` signal is a total of 6-bits wide.

In cases where the number of used bits varies from slot to slot (such as the number of `S_AXI_AWID` bits received from each connected master), the stride of the vectored signal is typically the maximum configured signal width among all the slots. (See the I/O signal tables in [Port Descriptions in Chapter 2](#) for details.) The unused bit positions are then tied off (inputs) or trimmed (outputs).

For example, if the AXI Crossbar core is configured with two SI slots and a data width of 128 bits, then each of the `wdata` and `rdata` signals on the SI of the core is a total of 256 bits wide, as shown in [Figure 3-1](#).



X13094

Figure 3-1: Vectored Slave/Master Interface

Specifically:

- "slot 0" uses `wdata [127:0]`
- "slot 1" uses `wdata [255:128]`, and connects to `wdata [127:0]` of the "Master 1" device).

If you instantiate the AXI Crossbar core directly into a design, you should tie off unused bit positions to zero to avoid synthesis and/or simulation warnings. When connecting to a read-only or write-only master or slave device, you should tie off all input bit positions corresponding to unused write or read channels, respectively.

Use of ID Signals

The transaction ID signals that propagate from SI to MI (`awid` and `arid`) and back again (`bid` and `rid`) identify the original source of each transaction, and therefore, how responses received on the MI are to be routed back to the originating SI slot, and ultimately to the originating endpoint master across the interconnect topology of the system.

Endpoint master devices can optionally output `awid` and `arid` signals that the master device can use to select among multiple "threads" of transactions, as though the master IP core was comprised of multiple master devices internally. The "reordering depth" is the total number of ID values that can be generated by a master, and is assumed to be $2^{idwidth}$, where `idwidth` is specified by the `THREAD_ID_WIDTH` parameter of each SI slot. Master devices with a reordering depth of one need not have any ID signals on their interface.

Transaction ordering is as follows:

- Transactions belonging to the same thread must be returned in order.
- Transactions among different threads can be returned out-of-order.

ID values among all the SI slots are made unique before propagating to any MI slot. The AXI Crossbar core prefixes a constant unique "master ID" value to the `awid` and `arid` signals sampled at each SI slot (if any).

A `BASE_ID` parameter associated with each SI slot allows the AXI Crossbar core to assign master IDs at compile time. Because endpoint master devices are not required to drive their assigned master ID on their own ID outputs, master devices do not need to be aware of their own assigned master ID values.

Master devices connected to the AXI Crossbar can use a different number of ID bits from one another. The AXI Crossbar only samples the ID bits defined by the `THREAD_ID_WIDTH` parameter for each SI slot. When assigning `BASE_ID` values for each SI slot, all lower order bits in the range `[THREAD_ID_WIDTH-1 : 0]` must be zero. There must be no overlap among the ID ranges defined by `BASE_ID` and `THREAD_ID_WIDTH` across all SI slots. (These rules are enforced by the tools.)

The `ID_WIDTH` parameter on the AXI Crossbar core specifies the width of the complete transaction ID signal used by its internal transaction ordering logic and propagated by all MI slots. The value of `ID_WIDTH` must be high enough to include enough high-order bits (Master ID) to uniquely distinguish between all the SI slots. The tools automatically set the values of the `ID_WIDTH` and all `BASE_ID` parameters to satisfy these requirements.

After reserving enough low-order ID bits to accommodate the maximum `THREAD_ID_WIDTH` value, the crossbar sets the high-order bits of `BASE_ID` to be the binary SI-slot sequence number (0 up to 0xF), which is the "Master ID" value. IP integrator automatically sets the parameter `ID_WIDTH` to accommodate the Master ID plus the maximum `THREAD_ID_WIDTH` value `[ceil_log2 NUM_SI + max(THREAD_ID_WIDTH)]`.

When configuring the AXI Crossbar as a stand-alone core in the Vivado® Integrated Design Environment (IDE), the value of the `ID_WIDTH` must exceed the maximum `THREAD_ID_WIDTH` by at least `ceil_log2 NUM_SI`. The `ID_WIDTH` is also used to determine the stride and total width of the ID signals on the SI of the crossbar, even though only a subset of those signals are sampled. Unsampled bit positions of the SI-side ID inputs should be tied-off to zero to avoid warnings during design compilation.

When two Interconnect instances are cascaded so that an MI slot of one instance connects to an SI slot of another instance, all ID signals produced by the upstream AXI Interconnect core are treated as though they are the thread ID bits of a connected master device. As with other master devices, the Crossbar in the downstream AXI Interconnect core prefixes the ID signals sampled from a cascaded SI slot with a unique master ID. This causes the ID width to grow as it propagates forward across a cascaded AXI Interconnect core topology.

All responses matching that master ID are routed back to the upstream AXI Interconnect core instance.

Address Decode

The AXI Crossbar core must determine which MI slot is the target of each transaction by decoding the address of each AW and AR channel transaction from the SI slots. This address decode involves only those upper-order address bits needed to distinguish between MI slots and ignores lower-order bits that might be used to distinguish locations within the connected slave device. The entire address value received from the SI is presented to the MI and made available to the slave device. It is visible to any connected monitors, even though the high-order address bits are typically not reused by the slave device.

In some cases, there might be multiple, possibly disjoint, address ranges that define when a single slave device (MI slot) is accessed. The address decode logic in the AXI Crossbar core includes the multiple ranges that determine selection of each MI slot. Differentiation between the multiple address ranges is also typically required by the functionality of the connected slave device.

Whenever a transaction address received on the SI does not match any of the ranges being decoded by the AXI Crossbar core, the transaction is trapped and handled by a decode error module within the AXI Crossbar core. The Crossbar generates a protocol-compliant response back to the originating master with the decoder error (DECERR) response code. The offending transaction is not seen by any connected slave.



IMPORTANT: *All range sizes must be a power of 2. The base address of all ranges must be aligned to (integer multiple of) its size. There must be no overlap among all address ranges across all MI slots configured in the AXI Crossbar. (These rules are enforced by the tools.) AXI Crossbar does not support address remapping.*

The Crossbar IP contains one address map table used to decode the addresses received on all SI slots. If you need to restrict any connected master device so that it does not have access to the entire address range of any connected slave device, then insert the AXI memory management unit (MMU) IP to configure address ranges specific to each master.

Transaction Acceptance and Issuing Limits

The WRITE_ACCEPTANCE and READ_ACCEPTANCE parameters limit the number of current outstanding transactions of each type that the crossbar will accept, per SI slot. The crossbar maintains transaction counters to accommodate the maximum number of concurrent threads, depending on the SI-slot ACCEPTANCE limit or the number of different AWID/ARID values ($2^{**} \text{THREAD_ID_WIDTH}$), whichever is smaller. The ACCEPTANCE limit parameters do not take into account the number of address transfers that might be accepted and stored in buffer modules, such as register slices and clock converters, that could be implemented along the address channel in the SI hemisphere, prior to arriving at the crossbar.

The WRITE_ISSUING and READ_ISSUING parameters limit the total number of current outstanding transactions of each type that the crossbar issues (with any ID value). The ISSUING limit parameters do not take into account the number of address transfers that might be accepted and stored in buffer modules, such as register slices and clock converters, that could be implemented along the address channel in the MI hemisphere, after being issued by the crossbar.

Regarding acceptance and issuing counters:

- A Write transaction is considered to be complete (counter decremented) when a `bvalid/bready` handshake completes at the crossbar.
- A Read transaction is considered to be complete when an `rvalid/rready` handshake completes at the crossbar in which `rlast` is asserted.

Write or Read transactions received at SI slots that have reached their acceptance limit, or that target an MI slot that has reached its issuing limit, are disqualified from arbitration so that the Write or Read arbiter, respectively, can continue to grant arbitration to other qualified SI slots, instead of stalling.

Increasing the value of an ACCEPTANCE or ISSUING parameter can increase data throughput by allowing Write and Read commands to be pipelined in connected slave devices, thus avoiding idle cycles on the Write and Read data channels. However, increasing the parameter values too much could lead to head-of-line blocking when multiple master devices access a shared slave.

When the tools copy the parameter values from the connected master and slave devices to the AXI Interconnect core:

- ISSUING parameters on connected master devices map to ACCEPTANCE parameters on the SI of the AXI Interconnect core
- ACCEPTANCE parameters on connected slave devices map to ISSUING parameters on the MI of the AXI Interconnect core

Note: For AXI4-Lite SI slots and MI slots, the ACCEPTANCE and ISSUING parameters, respectively, are ignored and only one outstanding transaction at a time is allowed.

Transaction Arbitration

Primarily, arbitration is granted based on the relative priority of the associated SI slot. The `ARB_PRIORITY` parameter for each SI slot can be set to a static priority value between 0 and 15; higher values have higher priority. In case of a tie:

- SI slot requests are disqualified if the SI slot acceptance limit or the targeted MI issuing limit has been reached. These disqualified requests are ignored by the arbiter.
- When the priority level of all qualified requesting SI slots have level 0, arbitration among them is decided by round-robin.
- When the highest priority value among the requesting SI slots is greater than 0, priority among slots sharing that priority value is based on their slot number; lower slot numbers have higher priority.

Write or Read transactions received at SI slots that have reached their acceptance limit are temporarily disqualified from arbitration so that the Write or Read arbiter, respectively, can continue to grant arbitration to other qualified SI slots, rather than stalling.

Furthermore, transactions that target an MI slot that has reached its issuing limit are also temporarily disqualified from arbitration. Such transaction requests remain disqualified until one cycle after the completion of a prior transaction on the targeted MI slot. (Completion of a write transaction occurs when `BVALID` and `BREADY` are both sampled High; completion of a read transaction occurs when `RVALID`, `RREADY` and `RLAST` are sampled High.) During the cycle in which the transaction completes, before the previously-pending requests become qualified, if a new assertion of `AWVALID` or `ARVALID` occurs on any other SI slot, it is allowed to be granted arbitration, even ahead of the previously-pending requests, regardless of its relative position in the round-robin queue. The arbiter is vulnerable to this priority inversion only during the one cycle in which a transaction completes on an MI slot that has previously reached its issuing limit. However, if a master device repeatedly asserts its `AWVALID`/`ARVALID` during the same cycle as it asserts its `BREADY`/`RREADY` in response to a prior transaction completion, it might lead to arbitration starvation on one or more of the previously-requesting SI slots. Any pipeline delay between the master and the crossbar, such as a width converter, will prevent the SI slot from re-requesting arbitration during the same cycle as a prior transaction completion. You can introduce such a pipeline delay, if needed, by simply enabling the register slice for the SI slot in the configuration of the AXI Interconnect.

Cyclic Dependency Avoidance

Any time there is more than one transaction ID (issued by one or more master devices) on which multiple outstanding transactions can be issued, and there is more than one connected slave device that can queue multiple transactions, and any of the slave devices can respond out-of-order on either the R or B channel, there is a potential cyclic dependency (deadlock) risk. Because the AXI Crossbar core is fully AXI-compliant, the AXI Crossbar core is equipped to handle slave devices that support out-of-order response.

How Deadlock Occurs

The following example shows how a sequence of Read transactions can result in deadlock. A similar situation also applies to a sequence of Write transactions when a slave device can reorder its Write response. This example shows a case where there are two master devices (M0 and M1) and two slave devices (S0 and S1) connected using the AXI Crossbar core.

1. Master device M0 reads from Slave device S0.
2. Master device M0 then reads from Slave device S1 (using the same ID thread).
3. Master device M1 then reads from Slave device S1.
4. Master device M1 then reads from Slave device S0 (using the same ID thread).
5. Slave device S0 responds to Master device M1 first. It re-orders the Read response, which is allowable because the received transaction IDs are different. However, the AXI Crossbar core cannot pass the response to Master device M1 because Master device M1 must first receive its response from Slave device S1.
6. Slave device S1 responds to Master device M0 (it does not re-order). But the AXI Crossbar core cannot pass the response to Master device M0 because Master device M0 must first receive its response from Slave device S0.

This results in deadlock.

Avoiding Deadlock Using Single Slave Per ID

The method used in the AXI Crossbar core to avoid deadlock is “Single Slave per ID.” This method does not impact the performance of the transactions of most critical concern. These are the pipelining of multiple Reads and Writes, and by multiple master devices to a performance-critical slave device, such as a memory controller.

The “Single Slave per ID” method imposes the restriction that each ID thread received at each SI slot (from each master device) can have outstanding transactions (in each of the write and read directions) to only one MI slot at a time. However, MI slots are still permitted to issue multiple outstanding transactions originating from multiple SI slots.

By imposing this rule in the example shown in the previous section, the Read transaction from M0 to S1 in [step 2](#) is stalled until S0 completes its response to M0. Similarly, the transaction from M1 to S0 in [step 4](#) is stalled until S1 completes its response to M1. Whatever ways the transactions proceed forward under these conditions would avoid the interdependencies that could cause deadlock.

Besides preventing deadlock, the Single Slave per ID rule also guarantees in-order completion of all Write transactions at the SIs, even if different MI slots are targeted by a transaction thread in successive transactions. For example, a master device writes to a direct memory access (DMA) descriptor in memory, then writes (using the same AWID) to a control register in a DMA engine that subsequently reads that descriptor.

Because the AXI Crossbar core does not allow the second Write to propagate to the DMA slave device until the first Write completes (Write response received from the memory controller), there is no risk that the DMA reads stale descriptor data from memory. Each master device is therefore guaranteed in-order completion of transactions to various slave devices, in the same direction, and on the same ID thread. Therefore, under those conditions, master devices do not need to condition subsequent Write transactions on receiving Write responses for prior transactions.



IMPORTANT: *AXI protocol provides no means to ensure in-order completion between Write and Read transactions other than waiting for the B-Channel responses of all earlier writes to complete.*

Error Signaling

The error conditions detected in the AXI Crossbar core follow.

- **Address decode error:** No eligible MI slot mapped to the address of the transaction, according to the connectivity map and applicable Write-only/Read-only parameters.
- An MI slot with the SECURE parameter enabled is targeted by a transaction in which `awprot [1]` or `arprot [1]` is set (unsecure).

If either of the preceding error conditions are detected, the AXI Crossbar core generates a protocol-compliant DECERR response to the connected master, and does not propagate the transaction to any MI slot.

The AXI Crossbar core does not detect the following error conditions.

- The response ID received on an MI slot does not map to any SI slot. This is indicative of a slave malfunction or system connectivity error that violates AXI protocol. The AXI Crossbar core does not assert the `ready` signal on the MI slot. The entire response (Write response or Read data burst) is permanently blocked by the AXI Crossbar core. This can cause the problematic slave device and any master device expecting to receive the response to hang.
- The AXI Crossbar core does not trap AXI4 protocol violations, which are the responsibility of the endpoint IP.
- The AXI Crossbar core neither supports nor traps Write data interleaving (all Write data is routed according to write transaction order; any `wid` sampled at the SI is not used for data transfer routing).
- The AXI Crossbar core does not trap narrow burst violations. This occurs when a slave device is configured with its narrow burst support logic disabled and it receives a transaction where there is a length > 1 data beat and the data transfer size is less than its physical data width. Narrow burst support should only be disabled when you are sure that all connected masters do not produce narrow bursts.

- The tools generally enforce design rules that prevent erroneous configurations at compile time. Therefore, no error detection is provided by the AXI Crossbar core for these configuration errors:
 - Parameter value range violations
 - Address or ID range overlap, non-binary size or base value misalignment.

Width Conversion

Each of the SI and MI on the AXI Interconnect core can be individually configured to have a data width of 32, 64, 128, 256, 512, or 1024 bits. When the data width of an interface is configured differently than the data width of the internal Crossbar, a width conversion module is automatically instantiated along the pathway.

The width conversion functions differ depending on whether the datapath width gets wider (upsizing) or narrower (downsizing) when moving in the direction from the SI toward the MI. The width conversion functions are the same in either the SI hemisphere (translating from an SI to the Crossbar) or the MI hemisphere (translating from the Crossbar to a MI).

Selecting a sufficiently high-data width for the AXI Crossbar can avoid loss of data bandwidth. For example, you could elect to set the AXI Crossbar to match the width of a speed-critical slave, such as a memory controller, even though all the masters that access that slave have narrower data widths.

With that setting, the AXI Interconnect core can perform data packing (Write transactions) or serialization (Read transactions) concurrently along multiple SI slot pathways in the SI hemisphere while the wide slave device and crossbar periodically maintain a data throughput rate higher than any one master device can sustain.

In contrast, selecting a lower data width for the AXI Crossbar can reduce logic resource utilization for less speed-critical designs. When seeking to minimize resource utilization, set the data width of the AXI Crossbar so that it minimizes the total number of width converters.

[Table 3-1](#) lists the transformations performed by the Data Width Converter for various parametric configurations and incident SI transactions. [Table 3-2](#) provides details of the resulting output (MI-side) transactions for each of the transformations. Use the Transformation Formula name in [Table 3-1](#) as an index into [Table 3-2](#).

[Table 3-1](#) and [Table 3-2](#) use these following designators when describing properties, signals, and derived equations.

1. $si.DW = SI_DATA_WIDTH$
2. $mi.DW = MI_DATA_WIDTH$
3. $si.LEN = S_AXI_AWLEN$ or $C_S_AXI_ARLEN$, as applicable
4. $si.ADDR = S_AXI_AWADDR$ or $C_S_AXI_ARADDR$, as applicable

5. $si.Bytes = si.DW / 8$
6. $mi.Bytes = mi.DW / 8$
7. $mi.ByteMask = mi.Bytes - 1$
8. $si.SIZE = S_AXI_AWSIZE$ or S_AXI_ARSIZE , as applicable
9. $si.SizeMask = (2^{**}si.SIZE) - 1$
10. $mi.SizeMask = (2^{**}mi.SIZE) - 1$
11. $mi.AlignedStart = si.ADDR \& \sim mi.ByteMask$
12. $mi.AlignedEnd = ((si.ADDR \& \sim si.SizeMask) + (si.LEN * 2^{**}si.SIZE)) \& \sim mi.ByteMask$
13. $mi.upsize_LEN = (mi.AlignedEnd - mi.AlignedStart) / mi.Bytes$
14. $si.conv_ratio = \text{ceil}((2^{**}si.SIZE) / mi.Bytes)$
15. $si.downsize_LEN = (si.LEN + 1) * si.conv_ratio - 1$
16. $mi.AlignedAdjustment = (si.ADDR \& si.SizeMask \& \sim mi.ByteMask) / mi.Bytes$
17. $si.burst_bytes = 2^{**}si.SIZE * (si.LEN + 1)$
18. $si.burst_mask = si.burst_bytes - 1$
19. $si.wrap_address = si.ADDR \& \sim si.burst_mask$
20. $si.wrap1_LEN = (si.burst_bytes - (si.ADDR \& si.burst_mask)) / mi.Bytes - 1$
21. $si.wrap2_LEN = (si.ADDR \& si.burst_mask) / mi.Bytes - 1$
22. $Downsize_ratio = \text{ceil}((2^{**}si.SIZE) / (MI_DATA_WIDTH / 8))$
23. MI beats = SI beats * Downsize_ratio (less MI beats skipped due to unaligned ADDR)
24. $max_beats = 256$ if (PROTOCOL == AXI4), 16 if (PROTOCOL == AXI3)

Table 3-1: AXI Data Width Converter Functional Truth Table

DATA_WIDTH	s_axi_a*burst	PROTOCOL	Input Conditions	Resulting Output	Transformation Formula
SI > MI (downsizer)	b01 (INCR)	0 (AXI4)	downsize_ratio = 1	Transaction unchanged	Pass-through Downsize
		1 (AXI3)	downsize_ratio = 1	Transaction unchanged	Pass-through Downsize
		0 (AXI4)	MI beats <= 256, downsize_ratio > 1	1 burst	INCR Downsize
		0 (AXI4)	MI beats > 256, downsize_ratio > 1	Split into max 256-beat bursts	Split INCR Downsize
		1 (AXI3)	MI beats <= 16, downsize_ratio > 1	1 burst	INCR Downsize
		1 (AXI3)	MI beats > 16, downsize_ratio > 1	Split into max 16-beat bursts	Split INCR Downsize
	b10 (WRAP)	0 (AXI4) or 1 (AXI3)	downsize_ratio = 1	Transaction unchanged (remains WRAP)	Pass-through Downsize
		0 (AXI4) or 1 (AXI3)	MI beats <= 16, downsize_ratio > 1	1 burst (remains WRAP)	WRAP Downsize
		0 (AXI4)	16 < MI beats <= 256, ADDR is burst-aligned	1 burst; change to INCR	WRAP-to-INCR Downsize
		0 (AXI4)	16 < MI beats <= 256, ADDR is not burst-aligned (wrapping required)	Split into 2 bursts; change to INCR	WRAP-to-INCR Downsize
		0 (AXI4)	MI beats > 256	Split into max 256-beat bursts; change to INCR	Split WRAP-to-INCR Downsize
		1 (AXI3)	MI beats > 16	Split into max 16-beat bursts; change to INCR	Split WRAP-to-INCR Downsize
	b00 (FIXED)	0 (AXI4) or 1 (AXI3)	downsize_ratio = 1	Transaction not modified (remains FIXED)	Pass-through Downsize
		0 (AXI4)	downsize_ratio > 1	Split into (s_axi_alen + 1) bursts; change to INCR	FIXED-to-INCR Downsize

DATA_WIDTH	s_axi_a*burst	PROTOCOL	Input Conditions	Resulting Output	Transformation Formula	
SI > MI (downsizer) (continued)	b0b00 (FIXED) (continued)	1 (AXI3)	$1 < \text{downsize_ratio} \leq 16$	Split into (s_axi_alen + 1) bursts; change to INCR	FIXED-to-INCR Downsize	
		1 (AXI3)	$\text{downsize_ratio} > 16$	Split into max 16-beat bursts; change to INCR	Split FIXED-to-INCR Downsize	
	x	2 (AXI4-Lite)	Write && ~s_axi_awaddr[2] && (s_axi_wstrb[7:4] != 0) && (s_axi_wstrb[3:0] != 0)	Split into 2 transactions	Lite Split Downsize	
			Write && ~s_axi_awaddr[2] && (s_axi_wstrb[7:4] == 0)	Transaction not modified	Lite Low-order Write Downsize	
			Write && (s_axi_awaddr[2] ((s_axi_wstrb[7:4] != 0) && (s_axi_wstrb[3:0] == 0)))	One transaction with m_axi_awaddr = s_axi_awaddr 'b100; m_axi_wdata = s_axi_wdata[63:32]	Lite Unaligned Downsize	
			Read && ~s_axi_araddr[2]	Split into 2 transactions	Lite Split Downsize	
			Read && s_axi_araddr[2]	Transaction not modified; s_axi_rdata[63:32] = m_axi_rdata; s_axi_rdata[31:0] undetermined	Lite Unaligned Downsize	
	SI < MI (upsizer)		0 (AXI4) or 1 (AXI3)	$\sim S_AXI_A*CACHE[1]$	Transaction not modified	Pass-through Upsize
		b01 (INCR)	x	$S_AXI_A*CACHE[1]$	1 burst	INCR Upsize
		b10 (WRAP)	0 (AXI4) or 1 (AXI3)	$S_AXI_A*CACHE[1]$	1 burst	WRAP Upsize
b00 (FIXED)		0 (AXI4) or 1 (AXI3)	x	Transaction not modified	Pass-through Upsize	
x		2 (AXI4-Lite)	x	Transaction not modified	Pass-through Upsize	

Table 3-2: AXI Data Width Converter Transaction Transformation Formulae

Transformation Formula	Conditions	Output Transactions	Output (MI) LEN	Output (MI) ADDR	Output (MI) BURST	Output (MI) LOCK
Pass-through Downsize ⁽¹⁾	x	1	No Change	No Change	s_axi_a*burst	s_axi_a*lock
INCR Downsize ⁽²⁾	x	1	si.downsize_LEN - mi.AlignedAdjustment	No Change	INCR	s_axi_a*lock
Split INCR Downsize ⁽²⁾	x	ceil ((si.downsize_LEN+1) / max_beats)	first = max_beats - mi.AlignedAdjustment - 1 ; last = si.downsize_LEN % max_beats; others = max_beats - 1	first = si.ADDR; transaction i = (si.ADDR & ~si.SizeMask) + ((i-1) * max_beats*mi.Bytes)	INCR	0
WRAP Downsize ⁽²⁾	x	1	si.downsize_LEN	No change	WRAP	s_axi_a*lock
WRAP-to-INCR Downsize ⁽²⁾	if ((si.ADDR & si.burst_mask) == 0) else	1	si.wrap1_LEN	si.ADDR	INCR	s_axi_a*lock
		2	first = si.wrap1_LEN ; second = si.wrap2_LEN	first = si.ADDR; second = si.wrap_address	INCR	0

Transformation Formula	Conditions	Output Transactions	Output (MI) LEN	Output (MI) ADDR	Output (MI) BURST	Output (MI) LOCK
Split WRAP-to-INCR Downsize ⁽²⁾	if ((si.ADDR & si.burst_mask) == 0)	ceil ((si.wrap1_LEN+1) / max_beats)	all = max_beats	first = si.ADDR; transaction i = (si.ADDR & ~si.SizeMask) + ((i-1) * max_beats*mi.Bytes)	INCR	0
	else	ceil ((si.wrap1_LEN+1) / max_beats) + ceil ((si.wrap2_LEN+1) / max_beats)	all = max_beats	first = si.ADDR; (others TBD, wrap as required)	INCR	0
FIXED-to-INCR Downsize ⁽²⁾	x	si.LEN+1	all = max(si.conv_ratio - mi.AlignedAdjustment - 1, 0)	all = si.ADDR	INCR	0
Split FIXED-to-INCR Downsize ⁽²⁾	x	(si.LEN+1) * int((si.conv_ratio - mi.AlignedAdjustment) / max_beats)	first = (si.conv_ratio - mi.AlignedAdjustment - 1) % max_beats; others = max_beats - 1	first = si.ADDR; (others TBD, repeat si.ADDR or increment as needed)	INCR	0
Lite Split Downsize	x	2	N/A (0)	first = si.ADDR; second = si.ADDR 'b100	N/A (singles)	N/A
Lite Low-order Write Downsize	x	1	N/A (0)	si.ADDR	N/A (singles)	N/A
Lite Unaligned Downsize	x	1	N/A (0)	si.ADDR 'b100	N/A (singles)	N/A
Pass-through Upsize ⁽¹⁾	x	1	No change	No change	s_axi_a*burst	s_axi_a*lock
INCR Upsize ⁽²⁾	x	1	mi.upsize_LEN	No change	INCR	s_axi_a*lock

Transformation Formula	Conditions	Output Transactions	Output (MI) LEN	Output (MI) ADDR	Output (MI) BURST	Output (MI) LOCK
WRAP Upsize ⁽²⁾	Write	1	$\text{ceil}((\text{si.LEN}+1) * (2^{**}\text{si.SIZE}) / \text{mi.Bytes}) - 1$	$\text{si.wrap_address} + (\text{ceil}((\text{si.ADDR} \& \text{si.burst_mask}) / \text{mi.Bytes}) * \text{mi.Bytes}) \% \text{si.burst_bytes}$	If (mi.LEN>0) then WRAP, else INCR	s_axi_a*lock
	Read	1	$\text{ceil}((\text{si.LEN}+1) * (2^{**}\text{si.SIZE}) / \text{mi.Bytes}) - 1$	$\text{si.wrap_address} + (\text{int}((\text{si.ADDR} \& \text{si.burst_mask}) / \text{mi.Bytes}) * \text{mi.Bytes})$	(If mi.LEN>0) then WRAP, else INCR	s_axi_a*lock

Notes:

1. Output (MI) SIZE = si.SIZE
2. Output (MI) SIZE = log2(mi.Bytes)

AXI Downsizer

The Width Conversion core performs a downsizer function whenever the data width on the SI side is wider than that on the MI side. During transactions in which the size of the data transfers (according to `awsize` or `arsize`) is wider than the data width on the MI side, downsizing is performed and, in the transaction issued to the MI side, the number of data beats is multiplied up accordingly.

- For writes, data serialization occurs on the W-channel between the SI and MI.
- For reads, data merging occurs on the R-channel between the MI and SI.

During merging, the AXI Data Width Converter sets the `rresp` for each output data beat (on the SI) to the worst-case error condition encountered among the input data beats being merged, according to the following descending precedence order: DECERR, SLVERR, OKAY, EXOKAY.

See [Table 3-1](#) and [Table 3-23](#) for details on the downsizing transformations for the various configurations and transaction types. When the transfer size of the transaction is equal to or less than the MI side data width, the transaction (address channel values) remains unchanged. Data transfers pass through unchanged except for byte-lane steering. This applies to both writes and reads.

The AXI Data Width Converter core factors up the length of each burst and detects when the resulting burst length would exceed the maximum burst limit (256 data beats for AXI4, 16 for AXI3). In such cases, the AXI Data Width Converter splits the transaction automatically into multiple conforming burst transactions.

- Exclusive Access is not supported through downsizers when burst lengths require splitting. If the `awlock` or `arlock` signal indicates an Exclusive Access write or read transaction, and downsizing results in splitting, then the AXI Data Width Converter core changes the `lock` signal in all output transactions to indicate Normal Access (0).
- When a downsized Write transaction results in splitting, the AXI Data Width Converter core coalesces the multiple Write responses at the MI and issues one Write response on the SI. The core sets the error response code (BRESP) to the worst-case error condition encountered among the multiple input responses, according to the following descending precedence order: DECERR, SLVERR, OKAY (EXOKAY cannot occur in a split transaction).

Downsizing, including transaction splitting, is not restricted by values of the `AW/ARCACHE` signal (specifically the “Modifiable” bit). Transaction splitting due to downsizing cannot be restricted by `CACHE` because there is no other alternative for completing the transaction.

The Data Width Converter allows multiple outstanding transactions to be propagated. Transaction characteristics from the AW/AR channel transfers are queued while awaiting corresponding response transfers.

However, due to the possibility of write response and read data re-ordering, transactions issued on the MI side of the Data Width Converter are restricted to a reordering depth of 1 (single ID thread).

As a result, the currently active transaction ID is stored internally and no ID signals are present on the MI. This eliminates the need for downstream IP cores to store and process ID information, saving logic.

Because the Data Width Converter changes the number of transfers on address and data channels between the SI and MI, no `user` signals are propagated across the core.

The Data Width Converter does not support downsizing directly from 1024 bits to 32. When configuring the AXI Interconnect core, if any SI is 1024 bits, the AXI Crossbar data width of the core must be set to a value greater than 32. If any MI is 32 bits, the AXI Crossbar core data width must be set to a value less than 1024.

When configured for AXI4-Lite protocol, the Data Width Converter provides for downsizing from 64-bit to 32-bit AXI4-Lite transfers. For reads, each SI transaction results in 2 MI transactions when the address is 64-bit-aligned; otherwise, 1 MI transaction for unaligned address. For writes, 64-bit-aligned SI transactions result in 2 MI transactions only when there is at least 1 byte lane (`WSTRB`) active for each 32-bit word.

You can also instantiate the AXI Data Width Converter core directly in your design (without AXI Interconnect core) along any pathway between a wide AXI master device and a narrower AXI slave.

AXI Upsizer

The Width Conversion core performs an upsizer function whenever the data width on the MI side is wider than on the SI side. Data packing is performed (for `incr` and `wrap` bursts), provided the `AW/ARCACHE[1]` bit (“Modifiable”) is asserted.

In the resulting transaction issued to the MI side, the number of data beats is reduced accordingly.

- For Writes, data merging occurs on the W-channel between the SI and MI.
- For Reads, data serialization occurs on the R-channel between the MI and SI.

The AXI Interconnect core replicates the `rresp` from each MI-side input data beat onto the `rresp` of each SI-side output data beat.

See [Table 3-1](#) and [Table 3-2](#) for details on the upsizing transformations for various configurations and transaction types.

When the `AW/ARCACHE[1]` bit is deasserted, the transaction (address channel values) remains unchanged and data transfers pass through unchanged except for byte-lane steering. This latter functionality is commonly referred to as an “expander.” Upsizing never results in transaction splitting.

The AXI Data Width Converter core allows multiple outstanding transactions to be propagated (AXI4 and AXI3 protocols only). Transaction characteristics from the AW/AR channel transfers are queued while awaiting corresponding response transfers. However, due to the possibility of read data re-ordering, transactions issued on the M1 side of the Data Width Converter are restricted to a reordering depth of 1 (single ID thread). As a result, the currently active transaction ID is stored internally and no ID signals are present on the M1. This eliminates the need for downstream IP cores to store and process ID information, saving logic.

When upsizing transfers in AXI3 and AXI4 protocol, the Data Width Converter IP core can also perform FIFO buffering. When this option is enabled, data merging on the W-channel and serialization on the R-channel is integrated into the FIFO buffer by using the asymmetrical block RAM feature of the FPGA. When FIFO buffering is enabled, the Data Width Converter IP core can also perform synchronous or asynchronous clock frequency conversion. (See [Clock Conversion](#) for available features.) Asynchronous clock conversion is integrated into the FIFO buffer by using the dual-clock feature of the FIFO. Integrating width conversion and optional clock conversion into the FIFO results in reduced latency and significantly reduced register utilization compared with implementing the same functions separately. When enabled, buffering is performed using the 512-deep “packet FIFO mode” (store and forward) method on both write and read channels. (See [AXI Data FIFO](#) for details.) When using the AXI Interconnect core in IP integrator, the IP core automatically recognizes opportunities to combine requested FIFO buffering and required clock conversion into the Data Width Converter IP core for datapaths that require upsizing.

When the Width Converter is configured in FIFO mode, it functions as a packet-mode FIFO, as described in the [AXI Data FIFO](#) section. That means propagation of AW channel transfers get delayed until complete write data bursts are stored in the FIFO, and AR channel transfers will get delayed as the FIFO fills until there is sufficient vacancy to accept the whole read data burst. Unlike the AXI Data FIFO, a fixed number of transactions can be accommodated in the FIFO, regardless of the actual length of the bursts, due to the width conversion function being performed. The FIFO is always implemented with a depth of 512 (single BRAM block), as viewed on the wider M1 interface. When configured to upsize the data width by a factor of 2, there are only 4 burst buffers implemented in the FIFO. When configured to upsize by any larger ratio, there will be 8 buffers implemented. This will, in turn, limit the maximum number of transactions the Data Width Converter can issue to the number of buffers minus one. Furthermore, burst buffers only become free, allowing more commands to be issued, only after each data burst is completely read from the output side of the FIFO.

When configured for AXI4-Lite protocol, the Data Width Converter provides for upsizing from 32-bit to 64-bit AXI4-Lite transfers. Each S1 transaction always results in one M1 transaction; no data packing is performed.

You can also instantiate the AXI Data Width Converter core directly in your design (without AXI Interconnect core) along any pathway between a narrow AXI master device and a wider AXI slave.

Clock Conversion

Each of the SI and MI on the AXI Interconnect core has its own `ac1k` pin, so that transfers through the Interconnect can cross clock domains. The AXI Crossbar core also has its own `ac1k` input. When the clock source driving the `ac1k` pin of a SI or MI is different than the clock source driving the internal Crossbar, an AXI Clock Converter core is automatically instantiated along the pathway.

Clock conversion can be performed in any of the following ways.

- Synchronous clock-rate acceleration (1:N), where the MI-side clock rate is an edge-aligned integer multiple of the SI-side clock rate.
- Synchronous clock-rate reduction (N:1), where the SI-side clock rate is an edge-aligned integer multiple of the MI-side clock rate.
- Asynchronous clock rate conversion either accelerates or reduces clock rate by passing the signals of each channel through an asynchronous FIFO.

When the tools determine that the relationship between an interface and the crossbar is an integer ratio (faster or slower) within the range 1:16 to 16:1, the tools automatically configure the Clock Converter to perform synchronous conversion; otherwise, the Clock Converter is configured in asynchronous mode.

When synchronous conversion is used, the clocks on either side must be derived from the same clock source with sufficiently small skew so that hold times are not violated by signals crossing in either direction. All clock domain crossings are resynchronized by flops within the AXI Clock Converter core. All flops that participate in resynchronization remain visible to static timing analysis tools and are covered by the period timing constraints defined for the clock signals themselves. The AXI Clock Converter core does not need to generate any multicyle timing constraints to override these resynchronization paths.

All paths that originate or end outside of the AXI Clock Converter core are neither over-constrained nor under-constrained with respect to their own clock domains.

When the AXI Clock Converter core is configured in asynchronous mode, all clock domain crossings are performed in an underlying instance of the FIFO Generator core, which is designed to internally resynchronize its write and read clock domains, regardless of the phase or frequency relationship. In asynchronous mode, the appropriate data-path-only timing constraints are generated by the core to cover all resynchronization paths.

Designs containing clock conversions should define their clocks in a system-level XDC file using the `create_clock` command. Each IP instance that implements an asynchronous clock-domain-crossing (CDC) attempts to generate IP-level timing constraints based on the clocks defined for the design. The purpose of the IP-generated constraints is to prevent timing violations during static timing analysis for CDCs that get resynchronized by the IP core. If you implement a design containing asynchronous clock conversions without defining your clocks at the system level, you can get warnings telling you that clocks cannot be found and that the generated `set_max_delay` constraints cannot be applied.

These warnings have no impact on the correct functional implementation of your design, and they will resolve when you eventually provide the required system-level clock definitions.

Clock converters always introduce latency. Asynchronous conversion incurs more latency and uses more logic resources than synchronous conversion. Passing through a clock converter in both the SI and MI hemispheres when traversing any pathway across the AXI Interconnect core is wasteful. Select clocks to avoid passing through clock converters in both hemispheres whenever possible.

To reduce the number of clock converters in the system, it can be advantageous to cascade AXI Interconnect core instances, grouping together similarly clocked devices. For example, connecting a group of low-frequency AXI4-Lite slaves to a separate AXI Interconnect core clocked at low frequency could consolidate the clock domain crossing onto a single converter in the pathway between the cascaded AXI Interconnect core instances.

When speed-critical devices, such as a memory controller, are connected to an AXI Interconnect core, best data throughput is usually achieved by clocking the AXI Crossbar with the same clock source as the speed-critical slave.

You can also instantiate the AXI Clock Converter core directly in your design (without AXI Interconnect core) along any pathway between an AXI master and slave device operating in different clock domains. The AXI Clock Converter core operates in any AXI protocol.

Protocol Conversion

Each of the SI and MI on the AXI Interconnect core can be individually configured to operate in AXI4, AXI3 or AXI4-Lite protocol. When the protocol of an interface is configured differently than the protocol of the internal Crossbar, an AXI Protocol Converter core is automatically instantiated along the pathway.

Conversion to AXI4-Lite

AXI4 or AXI3 master devices can be connected through an AXI Protocol Converter core to an AXI4-Lite slave to convert either single-beat transfers or multi-beat bursts issued by the master. In this mode, the Protocol Converter acts as a single-acceptance slave, in which there is only one outstanding SI transaction active at a time.

The transaction ID (*awid* or *arid*) received at the SI is stripped and stored in the conversion block, and retrieved during response transfers as *BID* or *RID*.

By default, the AXI4-Lite protocol conversion converts AXI4/AXI3 bursts into a sequence of single-beat transactions for AXI4-Lite slaves. When directly instantiating the Protocol Converter IP core, the *TRANSLATION_MODE* parameter can be set to 0 (Unprotected) to save resources when AXI4-Lite slaves are accessed only by well-behaved masters that issue only single-beat transactions.

In Unprotected mode, Endpoint masters having wider data width than the target AXI4-Lite slave should issue only transactions in which the data transfer size (according to AWSIZE or ARSIZE) is no larger than the data width of the targeted AXI4-Lite slave. This is to ensure that an intervening AXI Data Width Converter does not generate a multi-beat burst transfer while downsizing. Figure 3-2 illustrates the "unprotected" AXI4-Lite conversion logic.

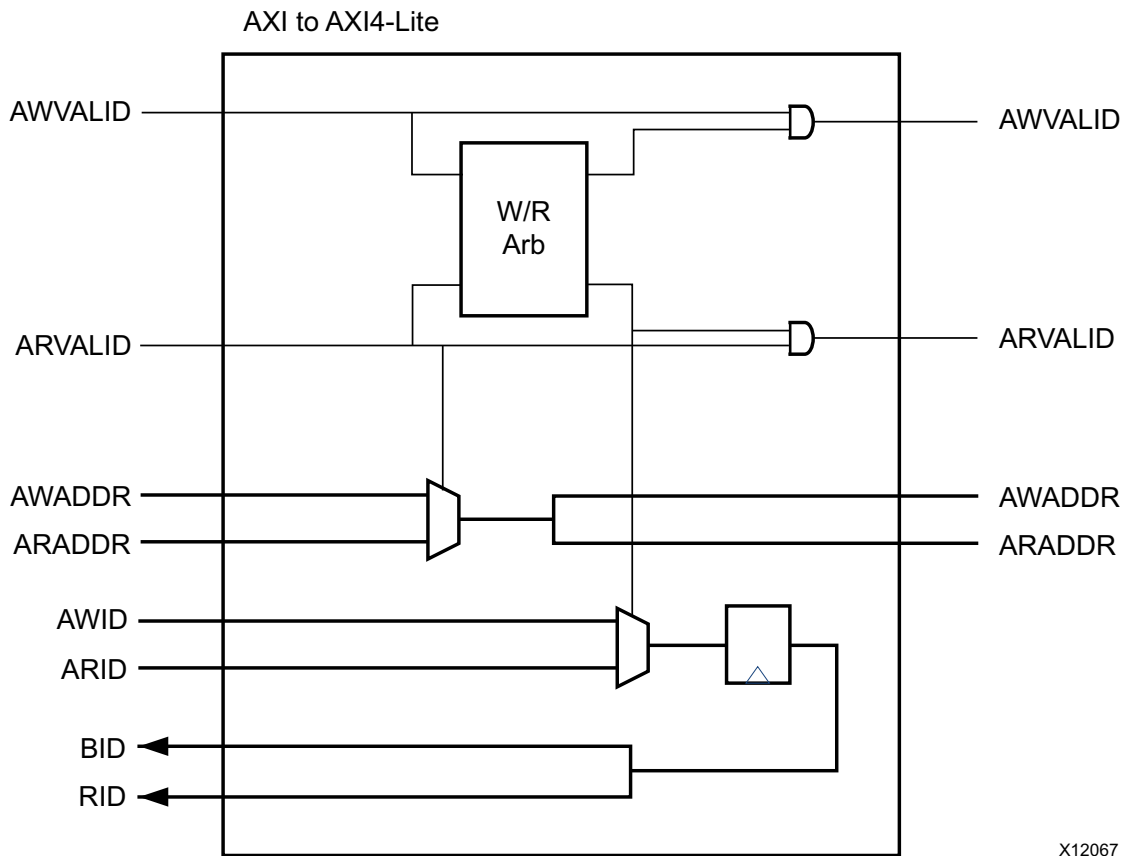


Figure 3-2: Unprotected AXI4-Lite Conversion Logic

AXI4-to-AXI3 Converter

An AXI4 master device can be connected through an AXI Protocol Converter core to an AXI3 slave. The Protocol Converter also produces the required WID output on the MI based on the AWID received at the SI.

By default, any time a burst longer than 16 data beats is received, the command is split into several shorter burst transactions. Other than that, the address channel characteristics are not modified by the Protocol Converter (the data transfer size is never modified).

The AXI3 converter module normally allows multiple outstanding transactions to be propagated. Transaction characteristics from the AW/AR channel transfers are queued while awaiting corresponding response transfers.

However, due to the possibility of write response and read data re-ordering, transaction acceptance by the Protocol Converter (while in AXI3 conversion mode) is dynamically reduced to a single outstanding transaction at a time (for each of the write and read directions) whenever a transaction requires splitting. (Unlike Data Width Converter, ID signals are always propagated between SI and MI, regardless of splitting.)

When directly instantiating the Protocol Converter IP core, the TRANSLATION_MODE parameter can be set to 0 (Unprotected) to save resources when AXI3 slaves are accessed only by well-behaved masters that issue transactions that never exceed 16 data beats (after any intervening width conversion).

Other Conversions

Conversions from AXI4-Lite to AXI4 or AXI3, and conversions from AXI3 to AXI4, require no functional logic. Output signals present in the MI-side protocol which are not present in the SI-side protocol are tied off to their default values.

You can also instantiate the AXI Protocol Converter core directly in your design (without AXI Interconnect core) along any pathway between an AXI master and slave device having different AXI protocol interfaces.

AXI Register Slices

You can optionally insert AXI Register Slice cores on selected pathways between the SI, crossbar and MI within the AXI Interconnect core, as needed, to break critical timing paths and achieve higher clock frequency. For each Register Slice instance, you can selectively enable pipelining on any of the five AXI channels. You can also instantiate the AXI Register Slice core directly in your design (without AXI Interconnect core) along any pathway between an AXI master and slave device.

The following modes of register slice are available on each AXI channel.

- **Fully_Registered:** Implemented as a two-deep FIFO buffer, this mode supports throttling by the channel source and/or channel destination as well as back-to-back transfers without incurring bubble cycles (up to 100% duty cycle). This mode is appropriate on W and R channels carrying bandwidth-critical AXI4 or AXI3 burst transfers.
- **Light_Weight:** Implemented as a simple one-stage pipeline register, this mode minimizes resources while isolating timing paths, but always incurs one bubble cycle following each transfer, resulting in a maximum duty cycle of 50%. This mode is appropriate on AW, AR and B channels, which normally do not require back-to-back transfers, and for all channels operating in AXI4-Lite protocol.

- **Registered Input:** This mode is selected as SI Reg on forward-propagating channels and as MI Reg on response channels. Implemented as a 4-deep FIFO buffer, this mode passes the VALID handshake input and all payload inputs on the source side through simple flip-flops before applying any throttling logic. This mode is particularly useful to pipeline AXI pathways crossing into an SLR region in a SSI device. Similar to the Fully-Registered mode, this mode supports back-to-back transfers (100% duty cycle) with one cycle of forward latency. See [AXI Register Slice Parameters](#) and [Register Slice Options](#).
- **SLR Crossing:** Adds extra pipeline stages to optimally cross an SLR boundary in SSI devices. The SI interface of the Register Slice and its connected AXI master device would then be located in one SLR, while the MI interface and its connected AXI slave device would be located in an adjacent SLR. All SLR crossings within the core are flop-to-flop with fanout=1. See [Constraining the Core](#) for floorplanning guidance.
- **SLR TDM Crossing:** Similar to SLR Crossing, except it consumes half the number of payload wires across the SLR boundary and propagates the cross-SLR signals at twice the frequency of the AXI interfaces. Configuring any AXI channel in this mode requires driving the `ac1k2x` clock input with a double-frequency edge-aligned clock signal.
- **Multi SLR Crossing:** Supports spanning zero or more SLR boundaries using a single Register Slice instance. Also, provides a selectable number of intermediate pipeline stages within each SLR to help close timing. All SLR crossings within the core are flop-to-flop with fanout=1. See [Constraining the Core](#) for floorplanning guidance.
- **Bypass:** Directly connects the SI to the MI.

AXI Data FIFO

You can optionally insert AXI Data FIFO cores on selected pathways between the SI, crossbar and MI within the AXI Interconnect core, as needed, to provide data buffering and achieve higher throughput. For each Data FIFO instance, you can selectively enable buffering on the write data channel, read data channel or both.

You can also instantiate the AXI Data FIFO core directly in your design (without AXI Interconnect core) along any pathway between an AXI master and slave device.

Under some circumstances, throughput is improved by buffering data bursts. This is commonly the case when the data rate at an SI or MI differs from the data rate of the AXI Crossbar, typically due to data width or clock rate conversion. Data buffering also allows real-time devices to tolerate transaction arbitration latency. Depending on your interconnect topology, it can be beneficial to place the Data FIFOs before or after the AXI Crossbar, or both.

There are three modes of Data FIFO that can be configured for each of the write and read paths.

- 32-deep LUT-RAM based FIFO (data channel only).

- 512-deep block RAM based FIFO (data channel only).
- 512-deep block RAM based Packet FIFO.

The Packet FIFO mode is used to avoid full/empty stalls in the middle of bursts. In addition to the 512-deep FIFO on the data channel, packet mode also implements a 32-deep FIFO on the corresponding address channel. Enabling packet mode for write causes a delay in the issuing of the write transaction on the AW channel until the entire write burst is stored in the FIFO, thus avoiding stalling due to a slow write data source. Enabling packet mode for read causes a delay in the issuing of the read transaction on the AR channel until the FIFO has enough vacancy to store the entire burst, according to `ARLEN`. (The “vacancy” is the amount of free space in the R channel FIFO that has not already been committed by previously issued AR commands.) This avoids stalling due to a slow read destination.

AXI Data FIFO does not support buffering of AXI4-Lite transactions.

AXI MMU

The AXI MMU IP is generally inserted automatically by the AXI Interconnect between an endpoint master device and the Crossbar to perform special address decoding services. There is generally no reason to instantiate the MMU directly in a design.

The Interconnect inserts the MMU whenever the address map view of any endpoint slave device differs among multiple connected master devices. This generally occurs when a master is to have access to a subset (aperture) of the slave's entire address range. The MMU enforces proper address range checking to prevent such masters from accessing beyond the intended address region. Any attempt to access outside of the ranges configured into the MMU will result in the MMU responding with a protocol-compliant response with the `BRESP/RRESP` field set to the `DECERR` value (2'b11); such transactions are not propagated through the MMU. The MMU can be configured with up to 256 address ranges of endpoint slave addresses that can be accessed by the master. The MMU can further define any of the address ranges as read-only or write-only.

Design Parameters

This section lists the configuration parameters for the AXI Interconnect core and each of the underlying AXI infrastructure cores.

AXI Interconnect Core Parameters

Table 3-3, Table 3-4, and Table 3-6 list the parameters used to configure the AXI Interconnect core only. When deploying the AXI Interconnect core in your design, additional parameters belonging to each of the underlying AXI Infrastructure cores are also available for customization when not automatically set according to system connectivity.

Table 3-3: AXI Interconnect Core Global Parameters

Parameter Name	Default Value	Format/Range	Description
NUM_SI	2	Integer (1-16)	Number of Slave Interfaces
NUM_MI	1	Integer (1-64) <ul style="list-style-type: none"> Range (1-16) when Number of Slave Interfaces > 1, Range (1-64) when Number of Slave Interfaces == 1 	Number of Master Interfaces
STRATEGY	0	Integer (0,1,2)	Control the implementation option strategy of the interconnect. <ul style="list-style-type: none"> If the parameter value is 0, the current settings are used. If the parameter value is 1, features that minimize the area of the infrastructure IP cores used within the interconnect instance are enabled. If the parameter value is 2, features that maximize the performance of the infrastructure IP cores used within the interconnect instance are enabled. (See AXI Interconnect Core — Top Level Settings for details.)

Table 3-4: AXI Interconnect Core SI-Related Parameters

Parameter Name	Default Value	Format/Range	Description
Snn_HAS_REGSLICE	0	Integer (0, 1)	<p>Controls AXI register slice insertion on SI.</p> <ul style="list-style-type: none"> If None (0) is selected, no register slice is inserted. If Outer (1) is selected, a register slice is inserted at the SI side of the SI coupler cells hierarchy. If Auto (2) is selected, a register slice is automatically inserted in the SI coupler cells hierarchy if SI coupler cells with common timing paths are detected. If Outer and Auto (3) is selected, a register slice is inserted at the SI side of the SI coupler cells hierarchy and an additional register slice can be inserted if SI coupler cells with common timing paths are detected. <p>For the AXI4-Lite protocol, use LIGHT_WEIGHT for all channels. For other protocols, use FULLY_REGISTERED on W and R channels and use LIGHT_WEIGHT on AW, AR and B channels.</p>
Snn_HAS_DATA_FIFO	0	Integer (0, 1, 2)	<p>Insert AXI data FIFO on SI.</p> <ul style="list-style-type: none"> If parameter value is 0, no data FIFO is inserted. If parameter value is 1, 32-deep data FIFOs are inserted on the W and R channels. If parameter value is 2, 512-deep data FIFOs are inserted on the write and read channels, and their packet mode feature is enabled.

Table 3-5: Table AXI interconnect Core Advanced Options

Parameter Name	Default Value	Format/Range	Description
ENABLE_ADVANCED_OPTIONS	0	Integer (0,1)	Setting a value of 1 enables a series of advanced configuration parameters (below). Setting a value of 0 disables all advanced configuration parameters.
ENABLE_PROTOCOL_CHECKERS	0	Integer (0,1)	Setting a value of 1 adds an AXI Protocol Checker IP core to each enabled master and slave interface and marks the interfaces for debug.
PCHK_WAITS	0	Integer (0..1024)	Specifies the maximum number of idle cycles for READY monitoring in all of the enabled protocol checkers.
PCHK_MAX_RD_BURSTS	2	Integer (2,4,8,16,32,64)	Specifies the maximum number of outstanding READ transactions per ID in all of the enabled protocol checkers.
PCHK_MAX_WR_BURSTS	2	Integer (2,4,8,16,32,64)	Specifies the maximum number of outstanding WRITE transactions per ID in all of the enabled protocol checkers.
XBAR_DATA_WIDTH	32	Integer (32,64,128,256,512,1024)	If specified, this value overrides any IP integrator automated value for the DATA_WIDTH parameter of the AXI Crossbar within the AXI Interconnect core Instance.
Snn_ARB_PRIORITY	0	Integer (0-15)	Specifies the value for the corresponding Snn_ARB_PRIORITY parameter of the AXI Crossbar instance connecting to the interconnect SI. Available when advanced configuration options have been enabled and NUM_SI>1.

Table 3-6: AXI Interconnect Core MI-Related Parameters

Parameter Name	Default Value	Format/Range	Description
Mnn_HAS_REGSLICE	0	Integer (0, 1)	<p>Controls AXI register slice insertion on MI.</p> <ul style="list-style-type: none"> If None (0) is selected, no register slice is inserted. If Outer (1) is selected, a register slice is inserted at the MI side of the MI coupler cells hierarchy. If Auto (2) is selected, a register slice is automatically inserted in the MI coupler cells hierarchy if MI coupler cells with common timing paths are detected. If Outer and Auto (3) is selected, a register slice is inserted at the MI side of the MI coupler cells hierarchy and an additional register slice might be inserted if MI coupler cells with common timing paths are detected. <p>For AXI4-Lite protocol, use LIGHT_WEIGHT for all channels. For other protocols, use FULLY_REGISTERED on W and R channels and use LIGHT_WEIGHT on AW, AR and B channels</p>
Mnn_HAS_DATA_FIFO	0	Integer (0, 1, 2)	<p>Insert AXI Data FIFO on MI</p> <ul style="list-style-type: none"> If parameter value is 0, no data FIFO is inserted. If parameter value is 1, 32-deep data FIFOs are inserted on the W and R channels. If parameter value is 2, a 512-deep data FIFOs are inserted on the write and read channels, and their packet mode feature is enabled.

AXI Crossbar Core Parameters

Table 3-7 lists the global parameters for the AXI Crossbar core.

Table 3-7: AXI Crossbar Global Parameters

Parameter Name	Default Value	Format/Range	Description
NUM_SI ^a	1	Integer (1-16)	Number of SI slots.
NUM_MI ^(a)	2	Integer (1-16)	Number of MI slots.
ID_WIDTH ^(a)	0	Integer (0-32)	Width of all ID signals propagated by the AXI Crossbar core. This is the actual width of ID signals on each MI slot. Each SI slot uses a subset of this width for its thread ID signals, if any.
ADDR_WIDTH ^(a)	32	For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)	Width of all ADDR signals for all SI slots and MI slots.
DATA_WIDTH ^(a)	32	Integer (32, 64, 128, 256, 512, 1024)	Data width of the internal interconnect Write and Read datapaths.
AWUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of awuser signals (if any) for all AXI4 SI slots and MI slots.
ARUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of aruser signals (if any) for all AXI4 SI slots and MI slots.
WUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of wuser signals (if any) for all AXI4 SI slots and MI slots.
RUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of ruser signals (if any) for all AXI4 SI slots and MI slots.
BUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of buser signals (if any) for all AXI4 SI slots and MI slots.
CONNECTIVITY_MODE	SAMD	String (SASD,SAMD)	Shared-Access (SASD); Sparse Crossbar (SAMD)
ADDR_RANGES	1	Integer (1-16)	Number of Address Ranges per MI slot
PROTOCOL ^(a)	AXI4	String (AXI4, AXI3, AXI4LITE)	Protocol of all interfaces
R_REGISTER	0	Integer (0, 1)	Enable Read Channel Internal Register Slice (SASD mode only)
STRATEGY	0	Integer (0, 1, 2)	Crossbar Optimization Strategy: 0 = Custom Settings 1 = Minimize Area 2 = Maximize Performance See AXI Crossbar Core — Global Tab for details.

a. Automatically set by tools based on system connectivity

Table 3-8 lists the SI-related parameters for the AXI Crossbar core. In the Parameter Name column “nn” represents a two-digit sequence number (with leading zero) with range $00 \leq nn \leq N-1$, where N refers to the total number of configured Slave Interfaces, which is the number of master devices connected to the AXI Crossbar core. Each row in the table therefore defines a set of N parameters.

Table 3-8: AXI Crossbar Slave Interface-Related Parameters

Parameter Name	Default Value	Format/Range	Description
Snn_BASE_ID ^{a b}	0	Bit32 (0-0xFFFFFFFF)	Base ID of each SI slot
Snn_THREAD_ID_WIDTH ^{a b}	0	Integer (0-32)	Number of variable low-order ID bits of each SI slot. Each value must be \leq ID_WIDTH.
Snn_SINGLE_THREAD	0	Integer (0, 1)	ID-thread support by SI slot: 0 = Accept multiple outstanding thread ID values (performance optimized). 1 = Accept only one outstanding thread ID value at a time (area optimized).
Snn_ARB_PRIORITY	0	Integer (0-16)	Arbitration priority among each SI slot. Higher values indicate higher priority. All slots with value 0 participate in round-robin arbitration.
Snn_WRITE_ACCEPTANCE	2	Integer (1-32)	Number of data-active Write transactions that each AXI SI slot can generate
Snn_READ_ACCEPTANCE	2	Integer (1-32)	Number of active Read transactions that each AXI SI slot can generate

- a. Automatically set by tools based on system connectivity.
- b. Parameter Snn_BASE_ID is not user modifiable; it is always set by the tools. THREAD_ID_WIDTH for each SI slot is determined by the number of ID bits propagated from each connected master device. After reserving enough low-order ID bits to accommodate the maximum THREAD_ID_WIDTH value, the crossbar sets the high-order bits of BASE_ID to be the SI-slot sequence number (0 up to 0xF), which is the “Master ID” value. IP integrator automatically sets parameter ID_WIDTH to accommodate the Master ID plus the maximum THREAD_ID_WIDTH value [$\text{ceil_log2 NUM_SI} + \text{max(THREAD_ID_WIDTH)}$]. When configuring AXI Crossbar as a stand-alone core in Vivado IDE, the value of ID_WIDTH must exceed the maximum THREAD_ID_WIDTH by at least ceil_log2 NUM_SI .

Table 3-9 lists the MI-related parameters for the AXI Crossbar core. In the Parameter Name column “mm” represents a two-digit sequence number (with leading zero) with range $00 \leq mm \leq M-1$, where M refers to the total number of configured Master Interfaces, which is the number of slave devices connected to the AXI Crossbar core. Each row in the table therefore defines a set of M parameters.

Table 3-9: AXI Crossbar Master Interface-Related Parameters

Parameter Name	Default Value	Format/Range	Description
Mmm_Aaa_BASE_ADDR ^a	mm * 0x100000 for Mmm_A00_BASE_ADDR, otherwise unused (0xFFFFFFFFFFFFFFF)	Bit64	Base address of each address range aa (where $0 \leq aa \leq \text{ADDR_RANGES}-1$) of each MI slot, mm (where $0 \leq mm \leq M-1$). All low-order bits of base address in the range [Mmm_Aaa_ADDR_WIDTH-1: 0] must be zero.
Mmm_Aaa_ADDR_WIDTH ^a	12 for range A00, otherwise 0 (unused)	For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)	Number of address bits representing the address space (in bytes) covered by each address range aa (where $0 \leq aa \leq \text{ADDR_RANGES}-1$) of each MI slot, mm (where $0 \leq mm \leq M-1$).
Mmm_Snn_WRITE_CONNECTIVITY	1	Integer (0, 1)	Enables the pathway between Snn and Mmm for write.
Mmm_Snn_READ_CONNECTIVITY	1	Integer (0, 1)	Enables the pathway between Snn and Mmm for read.
Mmm_WRITE_ISSUING	4	Integer (1-32)	Number of data-active Write transactions that each AXI4 MI slot can generate
Mmm_READ_ISSUING	4	Integer (1-32)	Number of active Read transactions that each AXI4 MI slot can generate
Mmm_SECURE	0	Integer (0, 1)	Indicates whether each MI slot connects to a secure slave device (allows TrustZone secure access). 0 = non-secure slave device 1 = secure slave device

- a. The size of all address ranges must be a power of 2, as determined by $2^{**}Mmm_Aaa_ADDR_WIDTH$. The Mmm_Aaa_BASE_ADDR of all ranges must be aligned to (integer multiple of) its size. There must be no overlap among all address ranges across all MI slots configured in the AXI Crossbar. Unused address ranges are designated by setting Mmm_Aaa_ADDR_WIDTH = 0 and setting BASE_ADDR to all ones (0xFFFFFFFFFFFFFFF). AXI Crossbar does not support address remapping.

Other AXI Infrastructure Core Parameters

This section defines the configuration parameters for the following AXI Infrastructure cores:

- [AXI Data Width Converter Parameters](#)
- [AXI Clock Converter Parameters](#)
- [AXI Protocol Converter Parameters](#)
- [AXI Data FIFO Parameters](#)
- [AXI Register Slice Parameters](#)
- [AXI MMU](#)

Table 3-10 lists the parameters common to all AXI Infrastructure cores, unless otherwise noted.

Table 3-10: AXI Infrastructure Common Parameters

Parameter Name	Default Value	Format/Range	Description
ID_WIDTH ^a	0	Integer (0-32)	Width of all ID signals propagated by the core. Except Data Width Converter core.
ADDR_WIDTH ^(a)	32	For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)	Width of all addr signals. Except AXI MMU core.
DATA_WIDTH ^(a)	32	For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)	Data width of the Write and Read datapaths. Except AXI Data Width Converter core.
AWUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of awuser signals (if any). Except Data Width Converter core.
ARUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of aruser signals (if any). Except Data Width Converter core.
WUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of wuser signals (if any). Except Data Width Converter core.
RUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of ruser signals (if any). Except Data Width Converter core.
BUSER_WIDTH ^(a)	0	Integer (0-1024)	Width of buser signals (if any). Except Data Width Converter core.
READ_WRITE_MODE ^(a)	READ_WRITE	String (READ_WRITE, READ_ONLY, WRITE_ONLY)	Enables read channels and/or write channels
PROTOCOL ^(a)	AXI4	String (AXI4, AXI3, AXI4LITE)	Protocol of all interfaces. Except AXI Protocol Converter core.

a. Automatically set by tools based on system connectivity

AXI Data Width Converter Parameters

Table 3-11 lists the parameters specific to the AXI Data Width Converter core.

Table 3-11: AXI Data Width Converter Parameters

Parameter Name	Default Value	Format/Range	Description
SI_DATA_WIDTH ^a	32	For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)	Data width of the SI-side Write and Read datapaths.
MI_DATA_WIDTH ^(a)	64	For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)	Data width of the MI-side Write and Read datapaths. (Must be different than SI_DATA_WIDTH)
SI_ID_WIDTH ^(a)	0	Integer (0-32)	Width of all ID signals (if any) on SI
FIFO_MODE ^(a)	0	Integer (0, 1, 2)	<ul style="list-style-type: none"> • 0 = No FIFO • 1 = Packet FIFO • 2 = Packet FIFO with Clock Conversion Modes 1 and 2 are supported only when PROTOCOL = AXI3 or AXI4 and SI_DATA_WIDTH < MI_DATA_WIDTH.
ACLK_RATIO ^(a)	1:2	String ("16:1"... "2:1", "1:2"... "1:16")	Ratio of SI-side clock frequency to MI. ^b
ACLK_ASYNC ^(a)	0	Integer (0, 1)	Enable asynchronous conversion. You can override automatic value from 0 to 1 to force asynchronous conversion. ^(b)
SYNCHRONIZATION_STAGES	2	Integer (2-8)	Defines the number of synchronizer stages across the cross clock domain logic.

a. Automatically set by tools based on system connectivity

b. Parameter setting is used only when FIFO_MODE = 2.

AXI Clock Converter Parameters

Table 3-12 lists the parameters specific to the AXI Clock Converter core.

Table 3-12: AXI Clock Converter Parameters

Parameter Name	Default Value	Format/Range	Description
ACLK_RATIO ^a	1:2	String ("16:1"... "2:1", "1:2"... "1:16")	Ratio of SI-side clock frequency to MI.
ACLK_ASYNC ^(a)	0	Integer (0, 1)	Enable asynchronous conversion. You can override automatic value from 0 to 1 to force asynchronous conversion.
SYNCHRONIZATION_STAGES	3	Integer (2-8)	Defines the number of synchronizer stages across the cross clock domain logic.

a. Automatically set by tools based on system connectivity

AXI Protocol Converter Parameters

Table 3-13 lists the parameters specific to the AXI Protocol Converter core.

Table 3-13: AXI Protocol Converter Parameters

Parameter Name	Default Value	Format/Range	Description
SI_PROTOCOL ^a	AXI4	String (AXI4, AXI3, AXI4LITE)	Protocol of SI.
MI_PROTOCOL ^(a)	AXI4LITE	String (AXI4, AXI3, AXI4LITE)	Protocol of MI.
TRANSLATION_MODE	2	Integer (0, 2)	0 = Unprotected: Master must be well-behaved 2 = Conversion: Incompatible bursts split into multiple transactions See AXI Protocol Converter, page 136 .

a. Automatically set by tools based on system connectivity

AXI Data FIFO Parameters

Table 3-14 lists the parameters specific to the AXI Data FIFO core.

Table 3-14: AXI Data FIFO Parameters

Parameter Name	Default Value	Format/Range	Description
WRITE_FIFO_DEPTH	0	Integer (0, 32, 512)	0 = No FIFO, 32 = LUT-RAM FIFO ⁽¹⁾ 512 = Block RAM FIFO ⁽¹⁾
READ_FIFO_DEPTH	0	Integer (0, 32, 512)	0 = No FIFO, 32 = LUT-RAM FIFO ⁽¹⁾ 512 = Block RAM FIFO ⁽¹⁾
WRITE_FIFO_DELAY	0	Integer (0, 1)	Enable Packet-mode FIFO (available when WRITE_FIFO_DEPTH = 512)
READ_FIFO_DELAY	0	Integer (0, 1)	Enable Packet-mode FIFO (available when READ_FIFO_DEPTH = 512)

Notes:

1. Available for AXI4 or AXI3 only.

AXI Register Slice Parameters

Table 3-15 lists the parameters specific to the AXI Register Slice core.

Table 3-15: AXI Register Slice Parameters

Parameter Name	Default Value	Format/Range	Description
REG_AW	Light	String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)	Mode of channel register slice ⁽¹⁾
REG_W	For AXI4 or AXI3: Full For AXI4-Lite: Light	String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)	Mode of channel register slice ⁽¹⁾
REG_B	Light	String (Bypass, Full, Light, MI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)	Mode of channel register slice ⁽¹⁾
REG_AR	Light	String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)	Mode of channel register slice ⁽¹⁾

Table 3-15: AXI Register Slice Parameters (Cont'd)

Parameter Name	Default Value	Format/Range	Description
REG_R	For AXI4 or AXI3: Full For AXI4-Lite: Light	String (Bypass, Full, Light, MI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)	Mode of channel register slice ⁽¹⁾

Notes:

1. Bypass: SI pins of channel are directly wired to MI.

Full: Implemented as a 2-deep FIFO buffer, supporting throttling by the channel source and/or destination as well as back-to-back transfers without incurring bubble cycles.

Light: Implemented as a simple 1-stage pipeline register, minimizing resources while isolating timing paths, but always incurring 1 bubble cycle following each transfer.

SI_Reg/MI_Reg: Passes the VALID handshake input and all payload inputs on the source side through simple flip-flops; supports back-to-back transfers.

SLR Crossing: Adds extra pipeline stages to optimally cross one SLR boundary in SSI devices.

SLR TDM Crossing: Similar to SLR Crossing, except it consumes half the number of payload wires across the SLR boundary and propagates the cross-SLR signals at twice the frequency of the AXI interfaces.

Multi SLR Crossing: Supports spanning zero or more SLR boundaries using a single Register Slice instance.

AXI MMU Parameters

Table 3-16: AXI MMU Parameters

Parameter Name	Default Value	Format/Range	Description
NUM_RANGES	1	Integer [0...256]	Number of Address Ranges
SI_ADDR_WIDTH	32	Integer [1...64]	Width of SI-side addr signals
MI_ADDR_WIDTH	32	Integer [SI_ADDR_WIDTH...64]	Width of MI-side addr signals
Dddd_BASE_ADDR ⁽¹⁾	ddd * 0x10000	Bit64	Base address of each address range ddd (where 000 <= ddd <= NUM_RANGES-1)
Dddd_ADDR_WIDTH ⁽¹⁾	16	Integer [0...SI_ADDR_WIDTH]	Base address of each address range ddd (where 000 <= ddd <= NUM_RANGES-1)
Dddd_READ_WRITE_MODE	READ_WRITE	String [READ_WRITE, READ_ONLY, WRITE_ONLY]	Enables read and/or write access to the target of each range

Notes:

1. The size of all address ranges must be a power of 2, as determined by $2^{Dddd_ADDR_WIDTH}$. The Dddd_BASE_ADDR of all ranges must be aligned to (integer multiple of) its size. There must be no overlap among all address ranges. Unused (null) address range can be designated by setting Dddd_ADDR_WIDTH = 0 (Dddd_BASE_ADDR is then ignored). If NUM_RANGES = 0, all address ranges are ignored.

Clocking

Each SI and each MI of the AXI Interconnect core has its own corresponding ACLK input, as does the underlying Crossbar core. See [Clock Conversion](#) for information about capabilities of the Interconnect for performing clock domain crossing.

When the clock source connected to the ACLK input of an SI or MI is the same as the clock connected to the Crossbar (no clock conversion), the SI or MI ACLK input is used to synchronize all AXI Infrastructure cores (such as Data Width Converter or Register Slice), if any, which are instantiated in the SI or MI hemisphere, respectively. When the `ac1k` of a SI is connected to a different source than the Crossbar, a Clock Converter core is instantiated. All Infrastructure cores on the SI (left) side of the Clock Converter, if any, are synchronized by the SI ACLK, while all cores between the Clock Converter and the Crossbar, if any, are synchronized by the Crossbar `ac1k` input. Similarly, when clock conversion is performed in the MI hemisphere, Infrastructure cores on the MI (right) side of the Clock Converter, if any, are synchronized by the MI `ac1k`, while all cores between the Crossbar and the Clock Converter, if any, are synchronized by the Crossbar `ac1k` input. The clock conversion IP core, including Data Width Converter configured in FIFO Mode, can perform either synchronous or asynchronous clock conversion. When configured for synchronous conversion, it is required that the SI and MI clocks remain edge-aligned at all times.

When using a Clock Wizard IP core to generate the AXI clocks, make sure the actual output clock frequencies maintain an integer-ratio relationship (1:2, 1:3... 1:16), as the actual frequency might differ from the requested frequency for each clock.

Resets

Each of the SI, MI and Crossbar `ac1k` is accompanied by an `aresetn` input, which must be synchronized to the corresponding `ac1k`. (This version of the AXI Interconnect core does not internally resynchronize any `aresetn` inputs.)

All AXI Interconnect Infrastructure cores deassert all `valid` and `ready` outputs shortly after `aresetn` is sampled active, and for the duration of the `aresetn` pulse.



IMPORTANT: *Each of the SI and MI must be put into the reset state at some time during the reset cycle of the Crossbar, and vice-versa, for every occurrence of reset. None of the AXI Interconnect cores support partial resetting. That is, whenever one interface is reset, all interfaces must be reset, and the resets must overlap. It is not necessary for multiple `aresetn` inputs to be deasserted during the same clock cycle.*



IMPORTANT: *All cores connected to the AXI Interconnect core, or to any AXI Infrastructure core described in this document, must have their connected AXI interface put into the reset state (`aresetn` outputs deasserted) at some time during the reset cycle of the AXI Interconnect core (must overlap), for every occurrence of reset. None of the AXI Interconnect cores support resetting one end of an AXI interface connection without the other, in either direction. It is not necessary for the `aresetn` input of the connected core to be deasserted during the same clock cycle as the AXI Interconnect core.*



RECOMMENDED: *As a general design guideline, Xilinx recommends asserting system `aresetn` signals for a minimum of 16 clock cycles (of the slowest `aresetn`), as that is known to satisfy the preceding reset requirements.*

Design Flow Steps

This chapter 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 Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 4\]](#)
- *Vivado Design Suite User Guide: Getting Started* (UG910) [\[Ref 5\]](#)
- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [\[Ref 6\]](#)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [\[Ref 7\]](#)

Customizing and Generating the Core

This section describes the Vivado Integrated Design Environment (IDE) used to specify IP options for the cores.

You can customize the IP core 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.

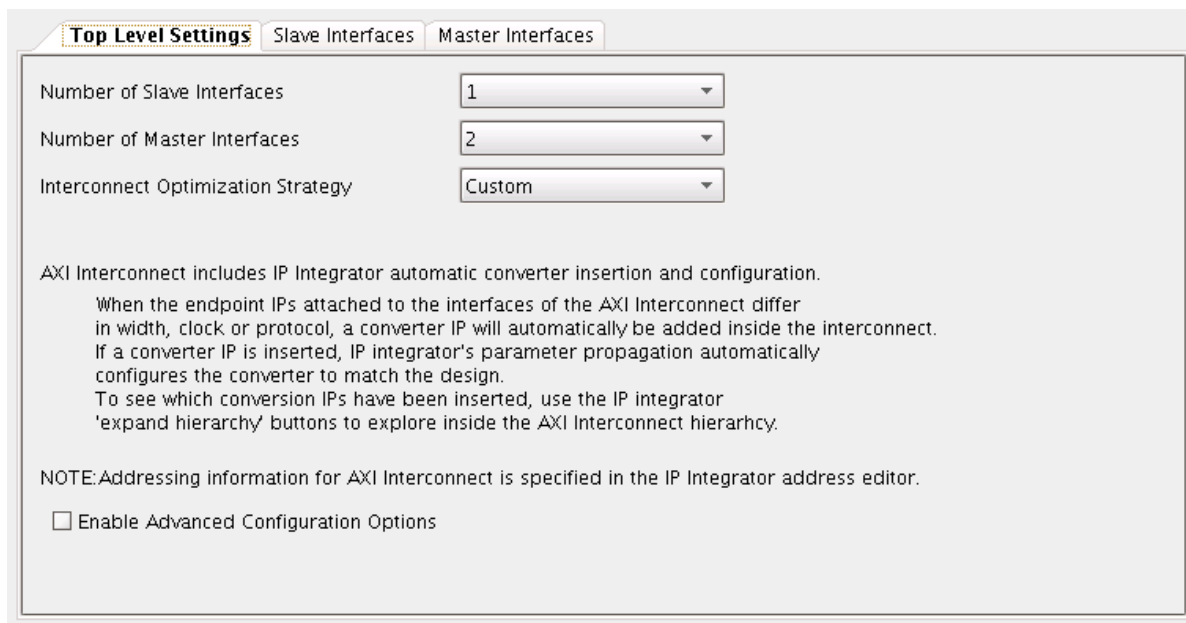
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 4\]](#) and the *Vivado Design Suite User Guide: Getting Started* (UG910) [\[Ref 5\]](#).

Note: When using the AXI Interconnect core in IP integrator, you can dive into its hierarchy and open the configuration dialogs of the underlying AXI Infrastructure cores to view their automatic parameter settings, but you cannot edit the parameters. To explicitly set the parameter values, instantiate the infrastructure core outside of the AXI Interconnect core in the IP integrator block design.

If you are customizing and generating the core in the Vivado IP integrator, see *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [\[Ref 6\]](#) for detailed information.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

AXI Interconnect Core — Top Level Settings



Number of Slave Interfaces: 1
 Number of Master Interfaces: 2
 Interconnect Optimization Strategy: Custom

AXI Interconnect includes IP Integrator automatic converter insertion and configuration.
 When the endpoint IPs attached to the interfaces of the AXI Interconnect differ in width, clock or protocol, a converter IP will automatically be added inside the interconnect. If a converter IP is inserted, IP integrator's parameter propagation automatically configures the converter to match the design.
 To see which conversion IPs have been inserted, use the IP integrator 'expand hierarchy' buttons to explore inside the AXI Interconnect hierarchy.

NOTE: Addressing information for AXI Interconnect is specified in the IP Integrator address editor.

Enable Advanced Configuration Options

Figure 4-1: AXI Interconnect Core – Top Level Settings

Interconnect Optimization Strategy

- Description: Controls the implementation option strategy of the interconnect.
 - If the parameter value is Custom (0), the Interconnect is configured according to the parameter settings in the other configuration pages.
 - If the parameter value is Minimize Area (1), features that minimize the area of the infrastructure IP cores used within the interconnect instance are enabled. Specifically, the crossbar within the Interconnect is configured in SASD (Shared-Access) mode.
 - If the parameter value is Maximize Performance (2), features that maximize the performance of the infrastructure IP cores used within the interconnect instance are enabled. Specifically, the crossbar within the Interconnect is configured in SAMD (Crossbar) mode, packet-mode FIFOs are enabled on all SI interfaces, and all crossbar acceptance and issuing thresholds are increased to allow multiple outstanding transactions for each connected master and slave.
- Format/Range: Integer (0, 1 2)
- Default Value: 0

Number of Slave Interfaces

- Format/Range: Integer (1-16)
- Default Value: 2

Number of Master Interfaces

- Format/Range: Integer (1-16) when Number of Slave Interfaces == 1, (1-64) when Number of Slave Interfaces > 1
- Default Value: 1

AXI Interconnect Core — Slave Interfaces Tab

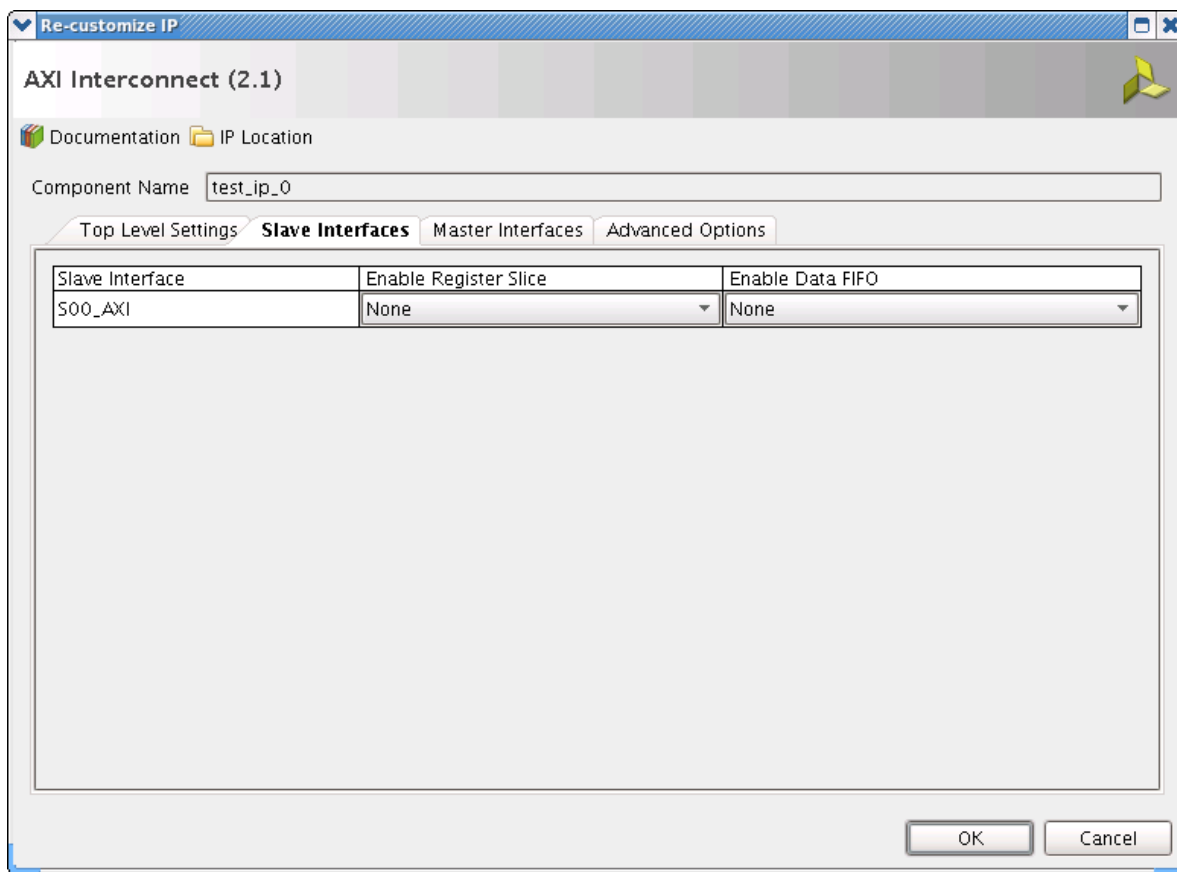


Figure 4-2: AXI Interconnect Core – Slave Interfaces Tab

Enable Register Slice

- Description: Controls AXI register slice insertion on SI.
 - If **None (0)** is selected, no register slice is inserted.
 - If **Outer (1)** is selected, a register slice is inserted at the SI side of the SI coupler cells hierarchy.
 - If **Auto (2)** is selected, a register slice is automatically inserted in the SI coupler cells hierarchy if SI coupler cells with common timing paths are detected.
 - If **Outer and Auto (3)** is selected, a register slice is inserted at the SI side of the SI coupler cells hierarchy and an additional register slice can be inserted if SI coupler cells with common timing paths are detected.

For the AXI4-Lite protocol, use LIGHT_WEIGHT for all channels. For other protocols, use FULLY_REGISTERED on W and R channels and use LIGHT_WEIGHT on AW, AR and B channels.

Enable Data FIFO

- Description: Insert AXI Data FIFO on SI
 - If parameter value is 0, no data FIFO is inserted.
 - If parameter value is 1, a 32-deep data FIFO is inserted.
 - If parameter value is 2, a 512-deep data FIFO is inserted and its packet mode feature is enabled.
- Format/Range: Integer (0, 1, 2)
- Default Value: 0

AXI Interconnect Core — Master Interfaces Tab

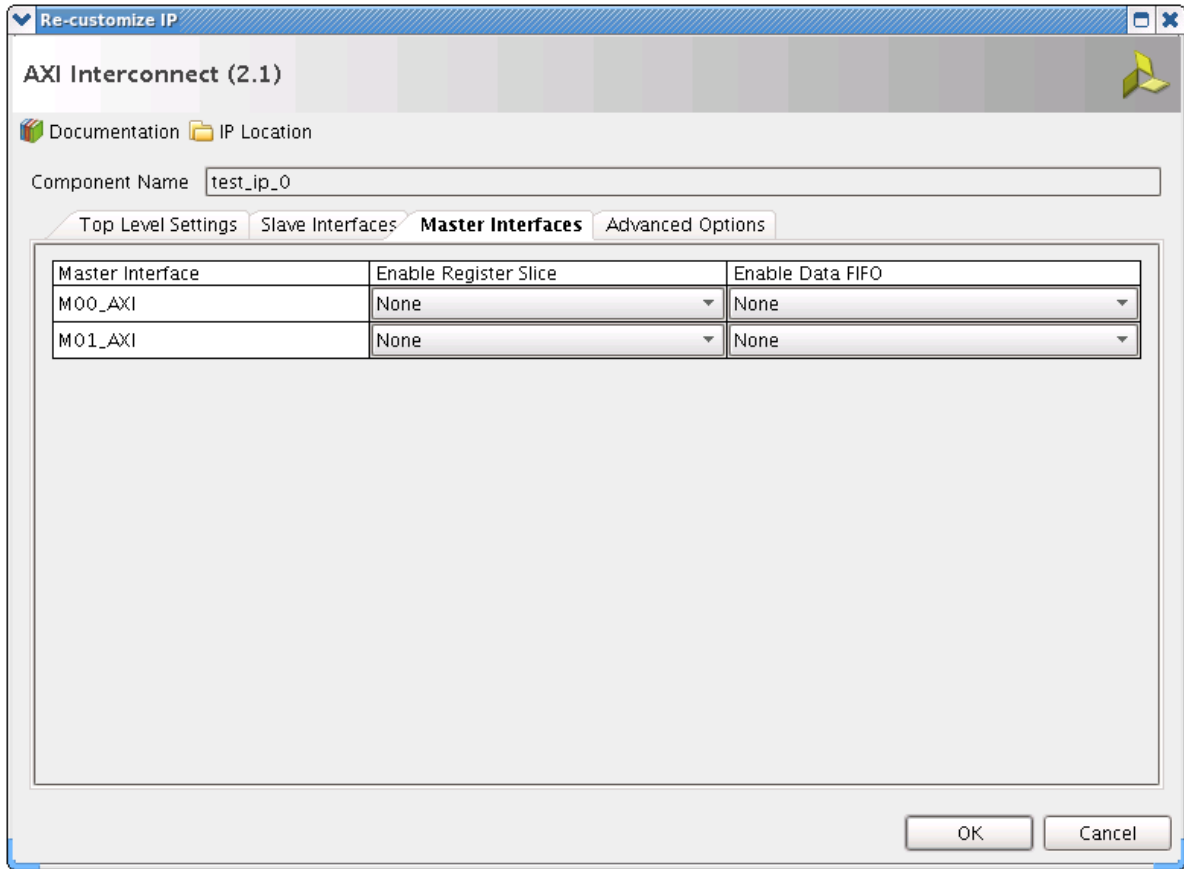


Figure 4-3: AXI Interconnect Core – Master Interfaces Tab

Enable Register Slice

- Description: Controls AXI register slice insertion on MI.
 - If **None (0)** is selected, no register slice is inserted.
 - If **Outer (1)** is selected, a register slice is inserted at the MI side of the MI coupler cells hierarchy.
 - If **Auto (2)** is selected, a register slice is automatically inserted in the MI coupler cells hierarchy if MI coupler cells with common timing paths are detected.
 - If **Outer and Auto (3)** is selected, a register slice is inserted at the MI side of the MI coupler cells hierarchy and an additional register slice might be inserted if MI coupler cells with common timing paths are detected.

For AXI4-Lite protocol, use LIGHT_WEIGHT for all channels. For other protocols, use FULLY_REGISTERED on W and R channels and use LIGHT_WEIGHT on AW, AR and B channels.

Enable Data FIFO

- Description: Insert AXI Data FIFO on MI
 - If parameter value is 0, no data FIFO is inserted.
 - If parameter value is 1, a 32-deep data FIFO is inserted.
 - If parameter value is 2, a 512-deep data FIFO is inserted and its packet mode feature is enabled.
- Format/Range: Integer (0, 1, 2)
- Default Value: 0

AXI Interconnect Core — Advanced Options Tab

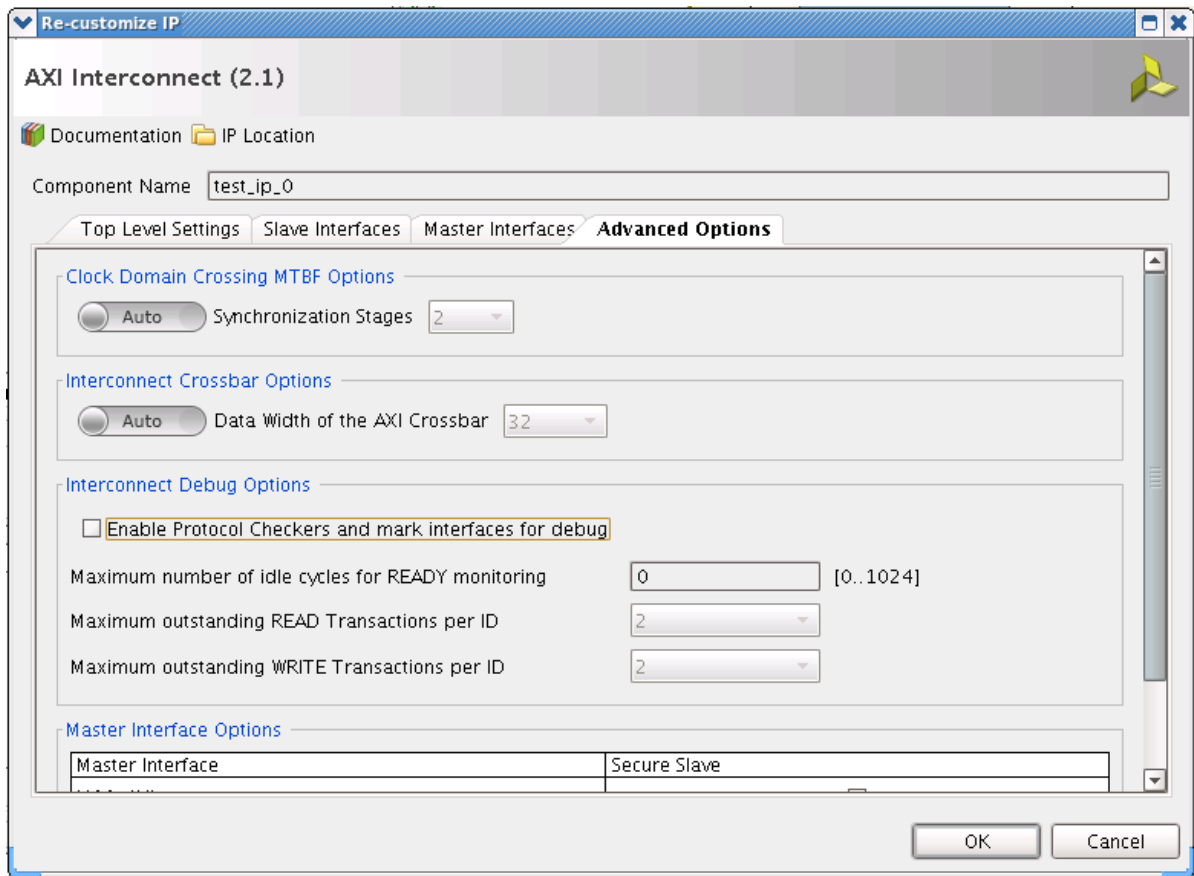


Figure 4-4: AXI Interconnect Core – Advanced Options Tab

Clock Domain Crossing MTBF Options

Synchronization Stages:

Description: Specifies the number of synchronization stages used in any asynchronous clock domain conversion couplers instantiated within the AXI Interconnect core.

Format/Range: Integer (2-8)

Default: 2

Interconnect Crossbar Options

Data Width of the AXI Crossbar:

When specified, this value overrides any IP integrator automated value for the DATA_WIDTH parameter of the AXI Crossbar within the AXI Interconnect Instance.

Interconnect Debug Options

Enable Protocol Checkers and mark interfaces for debug:

When checked, AXI Protocol Checker IP cores are instantiated inside the AXI Interconnect core and connected to each enabled AXI Master and AXI Slave interface. Each AXI Interconnect core AXI interface is also marked for debug.

- **Maximum number of idle cycles for READY monitoring:** Specifies the maximum number of idle cycles for READY monitoring in all of the enabled protocol checkers.
- **Maximum READ Transactions per ID:** Specifies the maximum number of outstanding READ transactions per ID in all of the enabled protocol checkers.
- **Maximum Write Transactions per ID:** Specifies the maximum number of outstanding WRITE transactions per ID in all of the enabled protocol checkers.

Master Interface Options

Secure Slave: Indicates whether each MI slot connects to a secure slave devices (requires TrustZone secure access)

Slave Interface Options

Arbitration Priority: Assigns an arbitration priority to the AXI Crossbar for each enabled SI (requires NUM_SI > 1).

AXI Crossbar Core — Global Tab

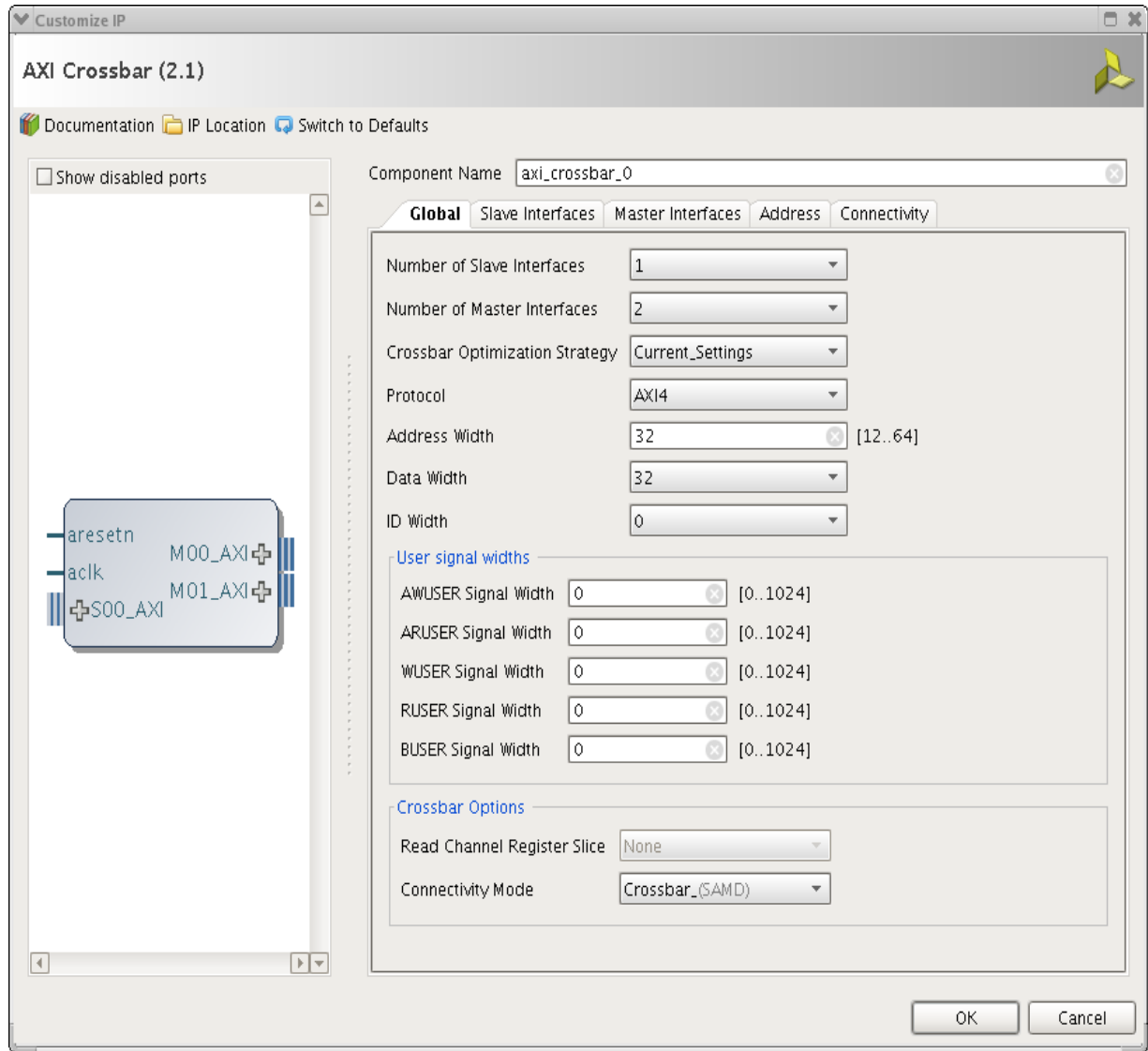


Figure 4-5: AXI Crossbar Core – Global Tab

Number of Slave Interfaces

- Description: Number of SI Slots
- Format/Range: Integer (1,16)
- Default Value: 2

Number of Master Interfaces

- Description: Number of MI Slots
- Format/Range: Integer (1,16)
- Default Value: 1

Crossbar Optimization Strategy

If set to Current Settings (0), the crossbar is configured according to the parameter settings in the other configuration pages.

If set to Minimize Area (1), the crossbar is configured in SASD (Shared-Access) mode.

If set to Maximize Performance (2), the crossbar is configured in SAMD (Crossbar) mode and all acceptance and issuing thresholds are increased to allow multiple outstanding transactions for each connected master and slave.

PROTOCOL

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

Address Width

- Description: Width of all ADDR signals for all SI slots and MI slots.
- Format/Range: For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)
- Default Value: 32

Data Width

- Description: Data width of the internal interconnect Write and Read datapaths.
- Format/Range: Integer (For AXI4 or AXI3: 32, 64, 128, 256, 512, 1024; for AXI4Lite: 32, 64)
- Default Value: 32

ID Width

- Description: Width of all ID signals propagated by the AXI Interconnect core. This is the actual width of ID signals on each MI slot. Each SI slot uses a subset of this width for its thread ID signals, if any.
- Format/Range: Integer (4-32)
- Default Value: 4

User Signal Widths

- Width of awuser Signal
 - Description: Width of `awuser` signals (if any) for all AXI4 SI slots and MI slots.
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- Width of aruser Signal
 - Description: Width of `aruser` signals (if any) for all AXI4 SI slots and MI slots.
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- Width of WUSER Signal
 - Description: Width of `wuser` signals (if any) for all AXI4 SI slots and MI slots.
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- Width of RUSER Signal
 - Description: Width of `ruser` signals (if any) for all AXI4 SI slots and MI slots.
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- Width of BUSER Signal
 - Description: Width of `buser` signals (if any) for all AXI4 SI slots and MI slots.
 - Format/Range: Integer (0-1024)
 - Default Value: 0

Crossbar Options

- Read Channel Register Slice
 - Description: Enable Read Channel Internal Register Slice (SASD mode only)
 - Format/Range: Integer (0, 1)
 - Default Value: 0
- Connectivity Mode
 - Description: Shared-Access (SASD); Sparse Crossbar (SAMD)
 - Format/Range: String (SASD,SAMD)
 - Default Value: SAMD

AXI Crossbar Core — Slave Interface Tab

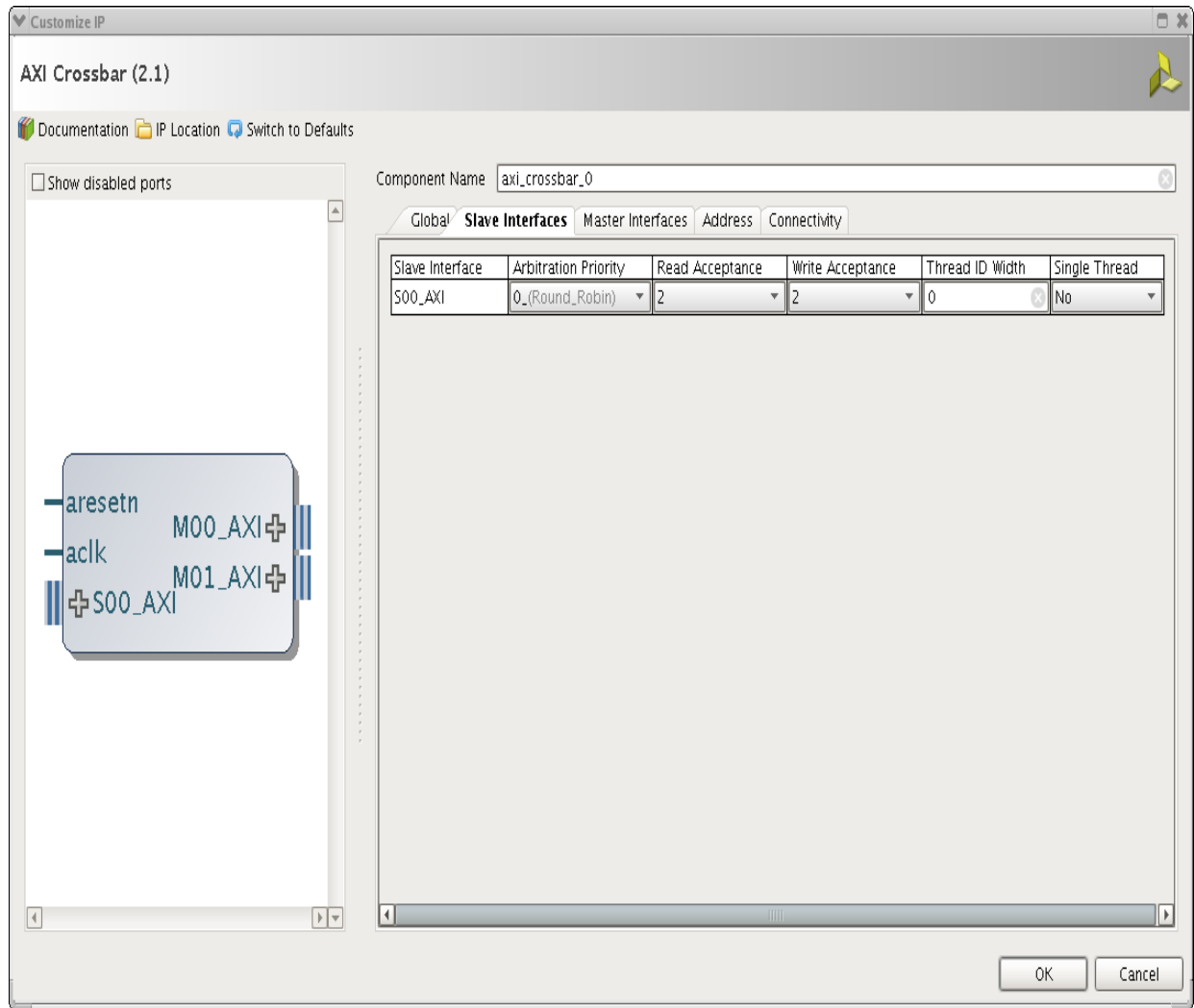


Figure 4-6: AXI Crossbar Core – Slave Interface Tab

Arbitration Priority

- Description: Arbitration priority among each SI slot. Higher values indicate higher priority. All slots with value 0 participate in round-robin arbitration.
- Format/Range: Integer (0-16)
- Default Value: 0

Read Acceptance

- Description: Number of active Read transactions that each AXI SI slot can generate
- Format/Range: Integer (1-32)
- Default Value: 2

Write Acceptance

- Description: Number of data-active Write transactions that each AXI SI slot can generate
- Format/Range: Integer (1-32)
- Default Value: 2

Thread ID Width

- Description: Number of variable low-order ID bits sampled by each SI slot.
- Format/Range: Integer (0-32). Each value must be \leq ID_WIDTH.
- Default Value: 0

Single Thread

- Description: ID-thread support by SI slot:
0 = Accept multiple outstanding thread ID values (performance optimized).
1 = Accept only one outstanding thread ID value at a time (area optimized).
- Format/Range: Integer (0-1)
- Default Value: 0

AXI Crossbar Core — Master Interface Tab

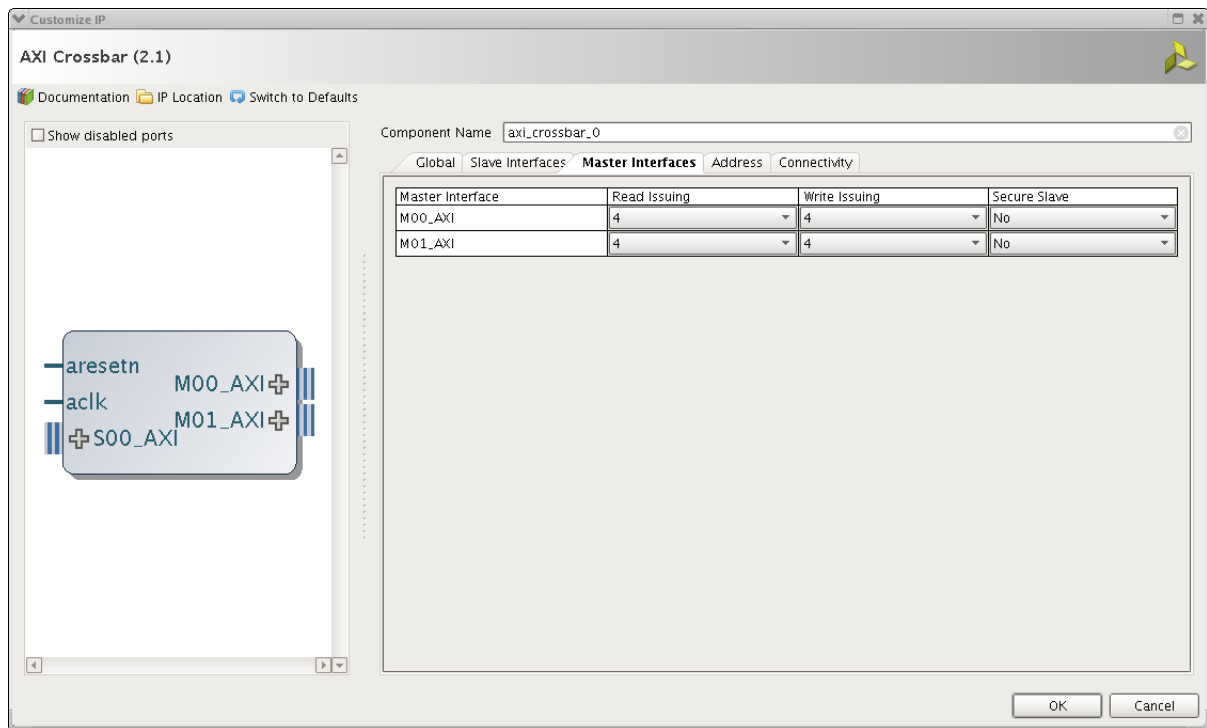


Figure 4-7: AXI Crossbar Core -- Master Interface Tab

Read Issuing

- Description: Number of active Read transactions that each AXI4 MI slot can generate
- Format/Range: Integer (1-32)
- Default Value: 4

Write Issuing

- Description: Number of data-active Write transactions that each AXI4 MI slot can generate
- Format/Range: Integer (1-32)
- Default Value: 4

Secure Slave

- Description: Indicates whether each MI slot connects to a secure slave device (allows TrustZone secure access).

0 = non-secure slave device

1 = secure slave device
- Format/Range: Integer (0-1)
- Default Value: 0

AXI Crossbar Core — Address Tab

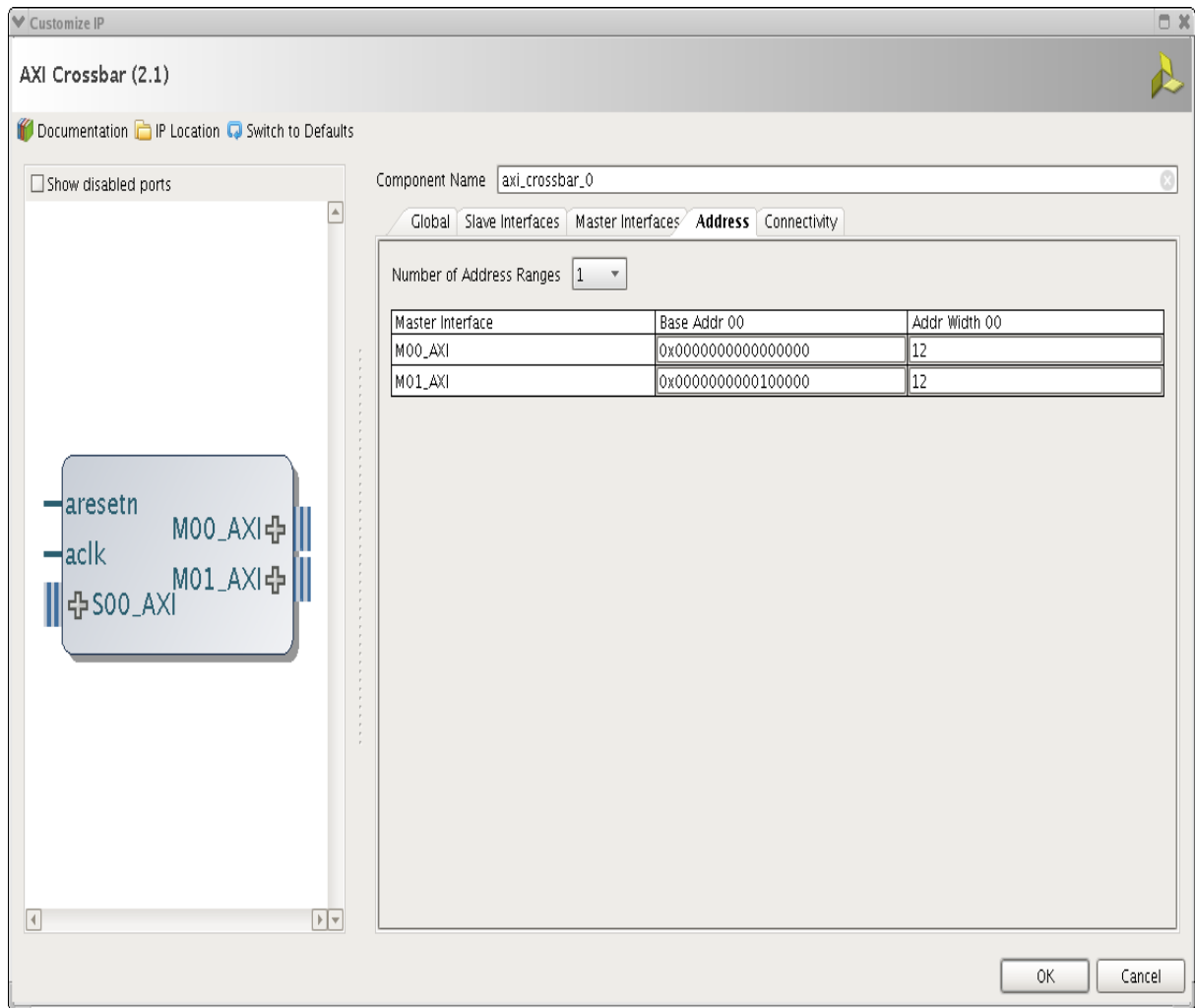


Figure 4-8: AXI Crossbar Core – Address Tab

Number of Address Ranges

- Description: Number of Address Ranges per MI slot.
- Format/Range: Integer (1-16)
- Default Value: 1

Base Addr

- Description: Base address of each address range of each MI slot. All low-order bits of base address in the range [ADDR_WIDTH-1: 0] must be zero.
- Format/Range: Bit-string, 64-bits
- Default Value: mm * 0x100000 for Mmm_A00_BASE_ADDR, otherwise unused (0xFFFFFFFFFFFFFFFF)

Address Width

- Description: Number of address bits representing the address space (in bytes) covered by each address range of each MI slot.
- Format/Range: For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)
- Default Value: 12 for range A00, otherwise 0 (unused)

AXI Crossbar Core — Connectivity Tab

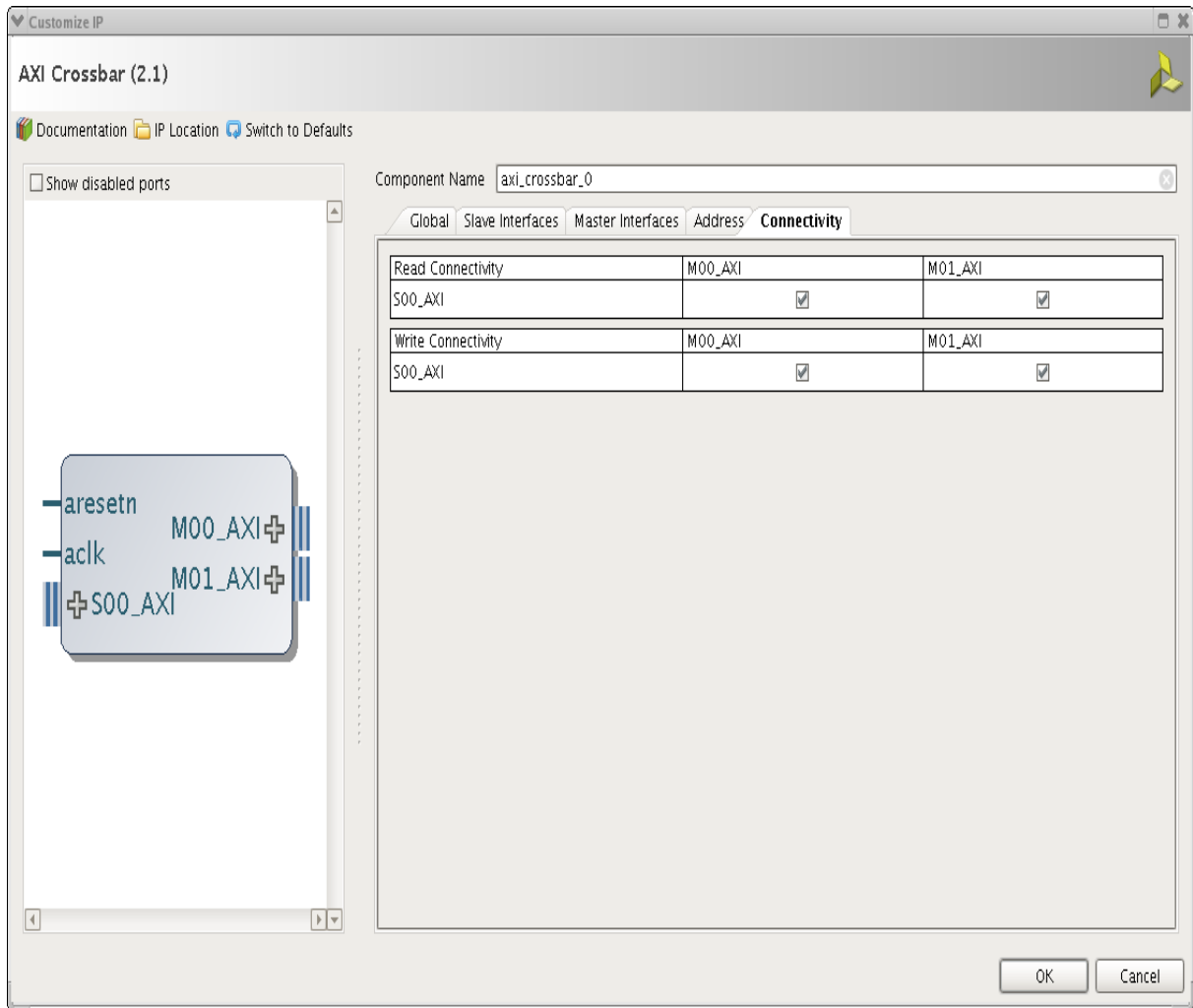


Figure 4-9: AXI Crossbar Core – Connectivity Tab

Connectivity

Enables the pathway between Snn and Mmm for read and/or write.

AXI Data Width Converter

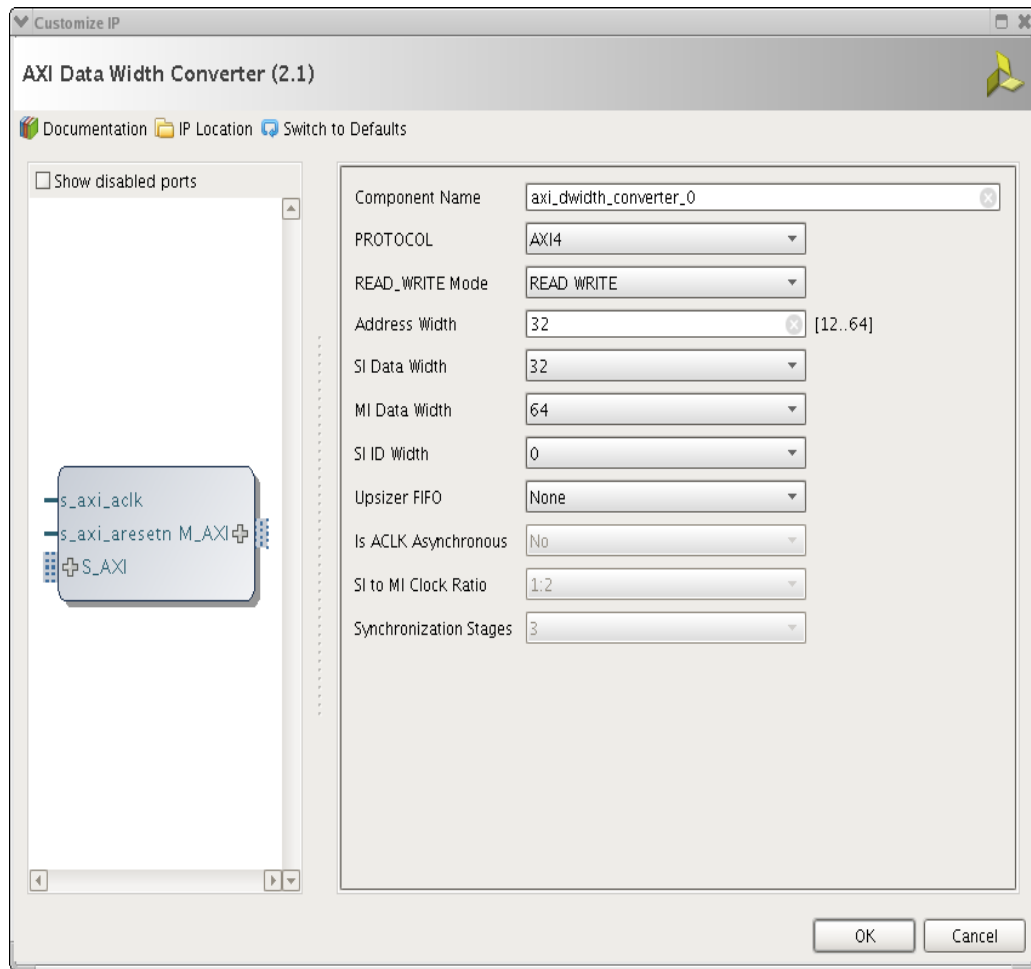


Figure 4-10: AXI Data Width Converter

Protocol

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

READ_WRITE Mode

- Description: Enables read channels and/or write channels. Automatically set by tools based on system connectivity
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Address Width

- Description: Width of all `addr` signals. Automatically set by tools based on system connectivity.
- Format/Range: Integer For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)
- Default Value: 32

SI Data Width

- Description: Data width of the SI-side Write and Read datapaths.
- Format/Range: For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)
- Default Value: 32

MI Data Width

- Description: Data width of the MI-side Write and Read datapaths. (Must be different than `SI_DATA_WIDTH`)
- Format/Range: For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)
- Default Value: 64

SI ID Width

- Description: Width of all ID signals (if any) on SI
- Format/Range: Integer (0-32)
- Default Value: 0

FIFO_MODE

- Description:
 - 0 = No FIFO
 - 1 = Packet FIFO
 - 2 = Packet FIFO with Clock Conversion

Modes 1 and 2 are supported only when `PROTOCOL = AXI3` or `AXI4` and `SI_DATA_WIDTH < MI_DATA_WIDTH`.

- Format/Range: Integer (0, 1, 2)
- Default Value: 0

ACLK_RATIO

- Description: Ratio of SI-side clock frequency to MI. Available only when FIFO_MODE = 2 and ACLK_ASYNC = 0.
- Format/Range: String ("16:1"... "2:1", "1:2"... "1:16")
- Default Value: 1:2

ACLK_ASYNC

- Description: Enable asynchronous conversion. Available only when FIFO_MODE = 2.
- Format/Range: Integer (0, 1)
- Default Value: 0

Synchronization Stages

- Description: Specifies the number of synchronization stages used in any asynchronous clock domain conversion within the core. Available only when FIFO_MODE=2 and ACLK_ASYNC=1.
- Format/Range: Integer (2-8)
- Default: 3

AXI Clock Converter

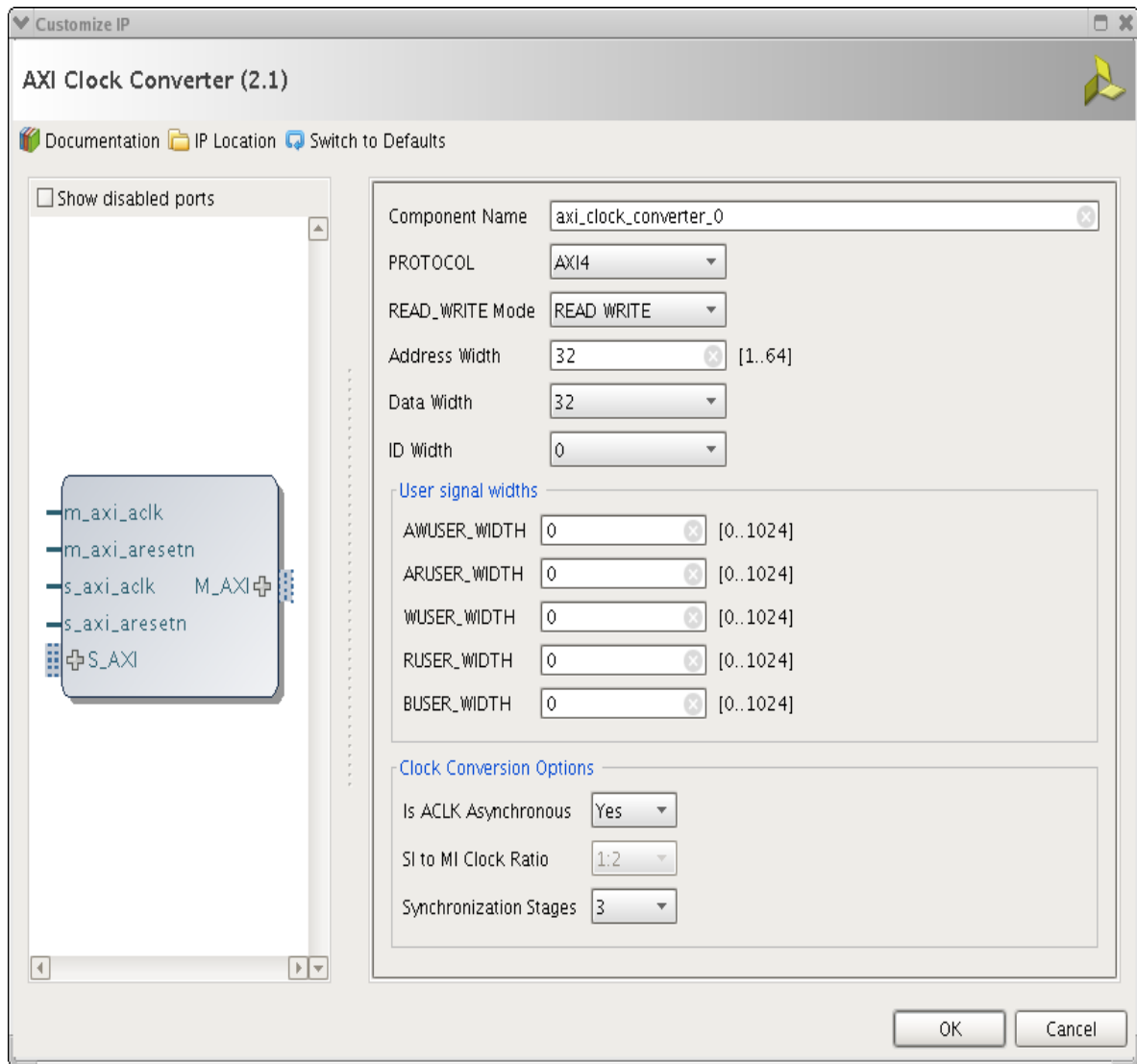


Figure 4-11: AXI Clock Converter

Protocol

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

READ_WRITE Mode

- Description: Enables read channels and/or write channels. Automatically set by tools based on system connectivity
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Address Width

- Description: Width of all `addr` signals. Automatically set by tools based on system connectivity.
- Format/Range: Integer For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)
- Default Value: 32

Data Width

- Description: Data width of the Write and Read datapaths. Automatically set by tools based on system connectivity
- Format/Range: For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)
- Default Value: 32

ID Width

- Description: Width of all ID signals propagated by the core. Automatically set by tools based on system connectivity.
- Format/Range: Integer (0-32)
- Default Value: 0

User signal widths

- AWUSER_WIDTH
 - Description: Width of `awuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

- ARUSER_WIDTH
 - Description: Width of `aruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- WUSER_WIDTH
 - Description: Width of `wuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- RUSER_WIDTH
 - Description: Width of `ruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- BUSER_WIDTH
 - Description: Width of `buser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

Clock Conversion Options

- SI to MI Clock Ratio
Available only when "Is ACLK Asynchronous" is 0.
 - Description: Ratio of SI-side clock frequency to MI.
 - Format/Range: String ("16:1"... "2:1", "1:2"... "1:16")
 - Default Value: 1:2
- Is ACLK Asynchronous
 - Description: Enable asynchronous conversion. You can override automatic value from 0 to 1 to force asynchronous conversion.
 - Format/Range: Integer (0, 1)
 - Default Value: 1

Synchronization Stages

- Description: Specifies the number of synchronization stages used in any asynchronous clock domain conversion within the core. Available only when "Is ACLK Asynchronous"=1.
- Format/Range: Integer (2-8)
- Default: 3

AXI Protocol Converter

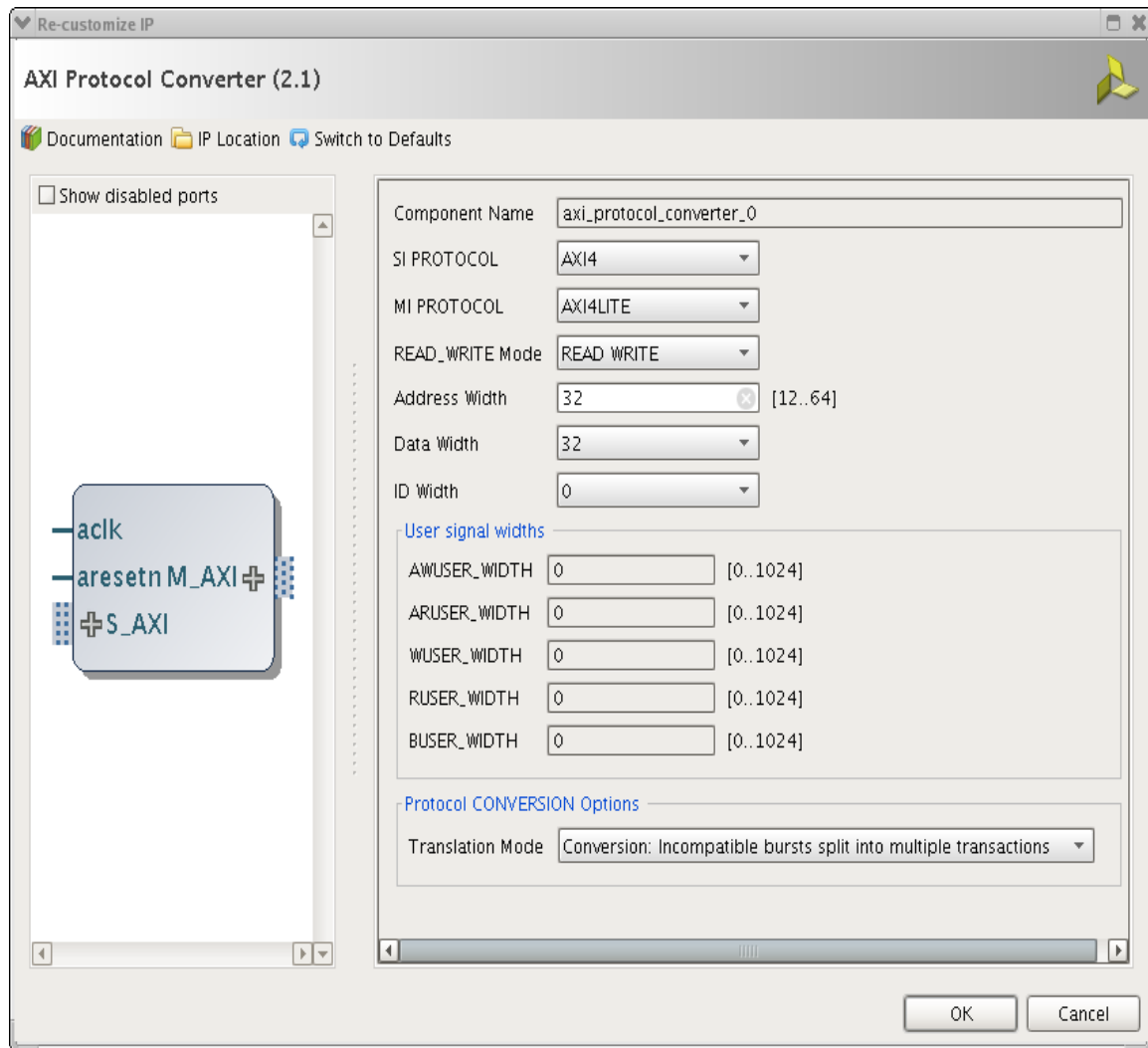


Figure 4-12: AXI Protocol Converter

SI Protocol

- Description: Protocol of SI.
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

MI Protocol

- Description: Protocol of MI.
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4LITE

READ_WRITE Mode

- Description: Enables read channels and/or write channels. Automatically set by tools based on system connectivity
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Address Width

- Description: Width of all `addr` signals. Automatically set by tools based on system connectivity.
- Format/Range: Integer (12-64)
- Default Value: 32

Data Width

- Description: Data width of the Write and Read datapaths. Automatically set by tools based on system connectivity
- Format/Range: If SI PROTOCOL or MI PROTOCOL is AXI4LITE: Integer (32, 64); otherwise: Integer (32, 64, 128, 256, 512, 1024)
- Default Value: 32

ID Width

- Description: Width of all ID signals on each AXI interface configured in AXI4 or AXI3 protocol. Automatically set by tools based on system connectivity.
- Format/Range: If SI PROTOCOL = AXI4LITE: Integer (0); otherwise: Integer (0-32)
- Default Value: 0

Protocol CONVERSION Options

- Translation Mode

This parameter is used only when translating from AXI4 to AXI3 or from AXI4 or AXI3 to AXI4LITE.

- Description:
 - 0 = Unprotected: Master must be well-behaved. For conversion to AXI4LITE, all transactions received on the SI must be single-beat transfers (AWLEN or ARLEN = 0). For conversion from AXI4 to AXI3, all bursts received on the SI must be ≤ 16 beats (AWLEN or ARLEN ≤ 15).
 - 2 = Conversion: Incompatible bursts split into multiple transactions (AXI4-to-AXI3 only)
- Format/Range: Integer (0, 2)
- Default Value: 2

AXI Data FIFO Core

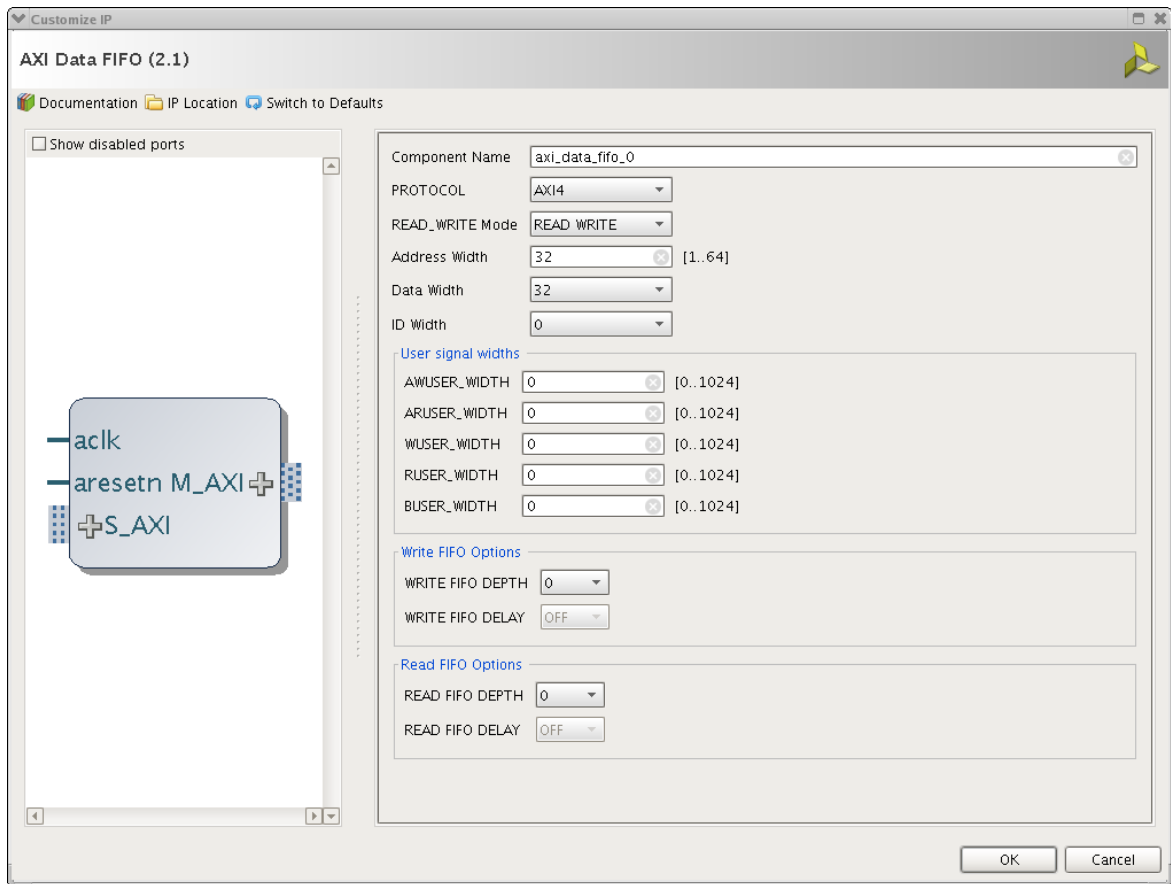


Figure 4-13: AXI Data FIFO Core

PROTOCOL

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3)
- Default Value: AXI4

READ_WRITE Mode

- Description: Enables read channels and/or write channels. Automatically set by tools based on system connectivity
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Address Width

- Description: Width of all `addr` signals. Automatically set by tools based on system connectivity.
- Format/Range: Integer (12-64)
- Default Value: 32

Data Width

- Description: Data width of the Write and Read datapaths. Automatically set by tools based on system connectivity
- Format/Range: Integer (32, 64, 128, 256, 512, 1024)
- Default Value: 32

ID Width

- Description: Width of all ID signals propagated by the core. Automatically set by tools based on system connectivity.
- Format/Range: Integer (0-32)
- Default Value: 0

User Signal Widths

- AWUSER_WIDTH
 - Description: Width of `awuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- ARUSER_WIDTH
 - Description: Width of `aruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- WUSER_WIDTH
 - Description: Width of `wuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

- RUSER_WIDTH
 - Description: Width of `ruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- BUSER_WIDTH
 - Description: Width of `buser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

Write FIFO Options

- WRITE FIFO DEPTH
 - Description:
 - 0 = No FIFO,
 - 32 = LUT-RAM FIFO
 - 512 = Block RAM FIFO
 - Format/Range: Integer (0, 32, 512)
 - Default Value: 0
- WRITE FIFO DELAY
 - Description: Enable Packet-mode FIFO (available when `WRITE_FIFO_DEPTH = 512`)
 - Format/Range: Integer (0, 1)
 - Default Value: 0

Read FIFO Options

- READ FIFO DEPTH
 - Description:
 - 0 = No FIFO,
 - 32 = LUT-RAM FIFO
 - 512 = Block RAM FIFO
 - Format/Range: Integer (0, 32, 512)

- Default Value: 0
- READ FIFO DELAY
 - Description: Enable Packet-mode FIFO (available when READ_FIFO_DEPTH = 512)
 - Format/Range: Integer (0, 1)
 - Default Value: 0

AXI Register Slice Core Settings Tab

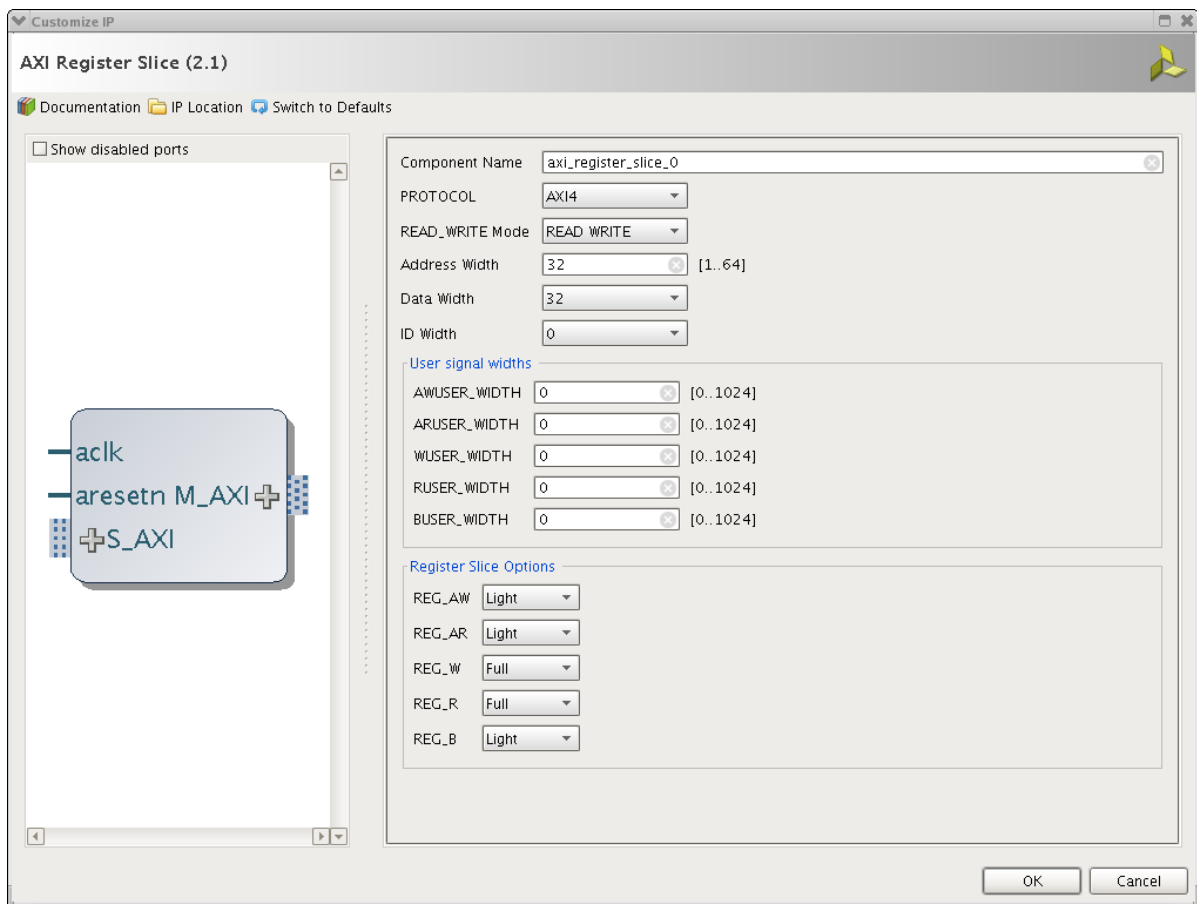


Figure 4-14: AXI Register Slice Core

PROTOCOL

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

READ_WRITE Mode

- Description: Enables read channels and/or write channels. Automatically set by tools based on system connectivity
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Address Width

- Description: Width of all `addr` signals. Automatically set by tools based on system connectivity.
- Format/Range: Integer For AXI4 or AXI3: Integer (12-64); for AXI4-Lite: Integer (1-64)
- Default Value: 32

Data Width

- Description: Data width of the Write and Read datapaths. Automatically set by tools based on system connectivity
- Format/Range: For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)
- Default Value: 32

ID Width

- Description: Width of all ID signals propagated by the core. Automatically set by tools based on system connectivity.
- Format/Range: Integer (0-32)
- Default Value: 0

User Signal Widths

- AWUSER_WIDTH
 - Description: Width of `awuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- ARUSER_WIDTH
 - Description: Width of `aruser` signals (if any). Automatically set by tools based on system connectivity

- Format/Range: Integer (0-1024)
- Default Value: 0
- WUSER_WIDTH
 - Description: Width of `wuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- RUSER_WIDTH
 - Description: Width of `ruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- BUSER_WIDTH
 - Description: Width of `buser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

Register Slice Options

- REG_AW
 - Description: Mode of channel register slice
 - Format/Range: String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)
 - Default Value: Light
- REG_AR
 - Description: Mode of channel register slice
 - Format/Range: String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)
 - Default Value: Light
- REG_W
 - Description: Mode of channel register slice
 - Format/Range: String (Bypass, Full, Light, SI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)

- Default Value: Light if PROTOCOL = AXI4LITE, otherwise Full
- REG_R
 - Description: Mode of channel register slice
 - Format/Range: String (Bypass, Full, Light, MI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)
 - Default Value: Light if PROTOCOL = AXI4LITE, otherwise Full
- REG_B
 - Description: Mode of channel register slice
 - Format/Range: String (Bypass, Full, Light, MI_Reg, SLR Crossing, SLR TDM Crossing, Multi SLR Crossing)
 - Default Value: Light

AXI Register Slice — SLR Crossings Tab

The properties in this tab are enabled only when Multi SLR Crossing is selected for any AXI channel.

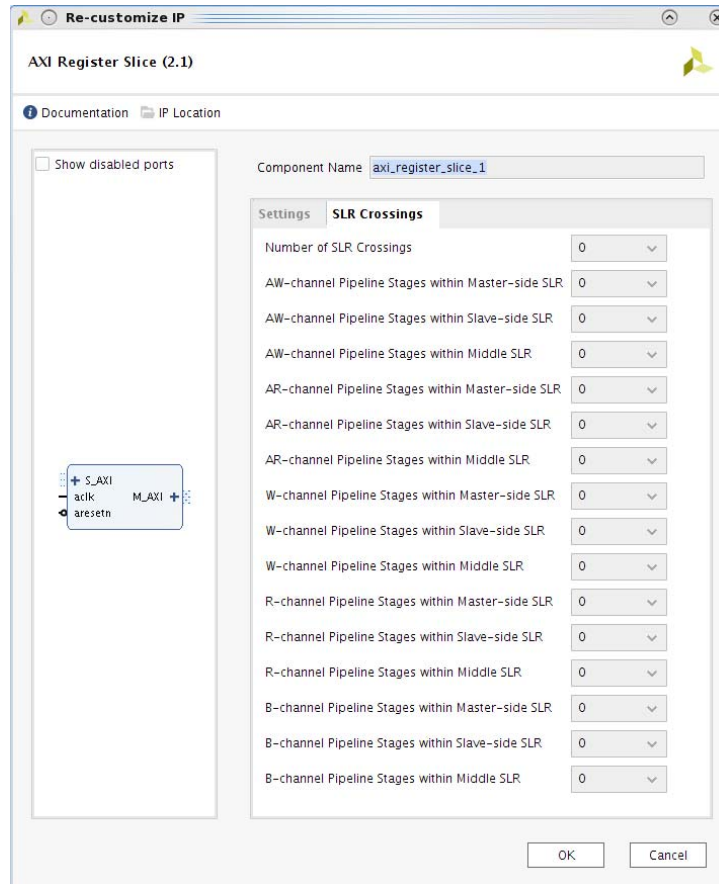


Figure 4-15:

- Number of SLR Crossings
 - Description: Select the number of SLR boundaries to be crossed within the core.
 - Format/Range: Integer (0-2).
 - Default Value: 0
- AW/AR/W/R/B-channel Pipeline Stages within Master-side SLR
 - Description: Select the number of additional pipeline stages to insert within the master-side SLR (between the SLR boundary and the SI interface).
 - Format/Range: Integer (0-4).
 - Default Value: 0
- AW/AR/W/R/B-channel Pipeline Stages within Slave-side SLR

- Description: Select the number of additional pipeline stages to insert within the slave-side SLR (between the SLR boundary and the MI interface).
- Format/Range: Integer (0-4).
- Default Value: 0
- AW/AR/W/R/B-channel Pipeline Stages within Middle SLR
 - Description: Select the number of additional pipeline stages to insert within the middle SLR (between the two SLR boundaries). Enabled only when Number of SLR Crossings is > 1.
 - Format/Range: Integer (0-4).
 - Default Value: 0

AXI MMU – Settings Tab

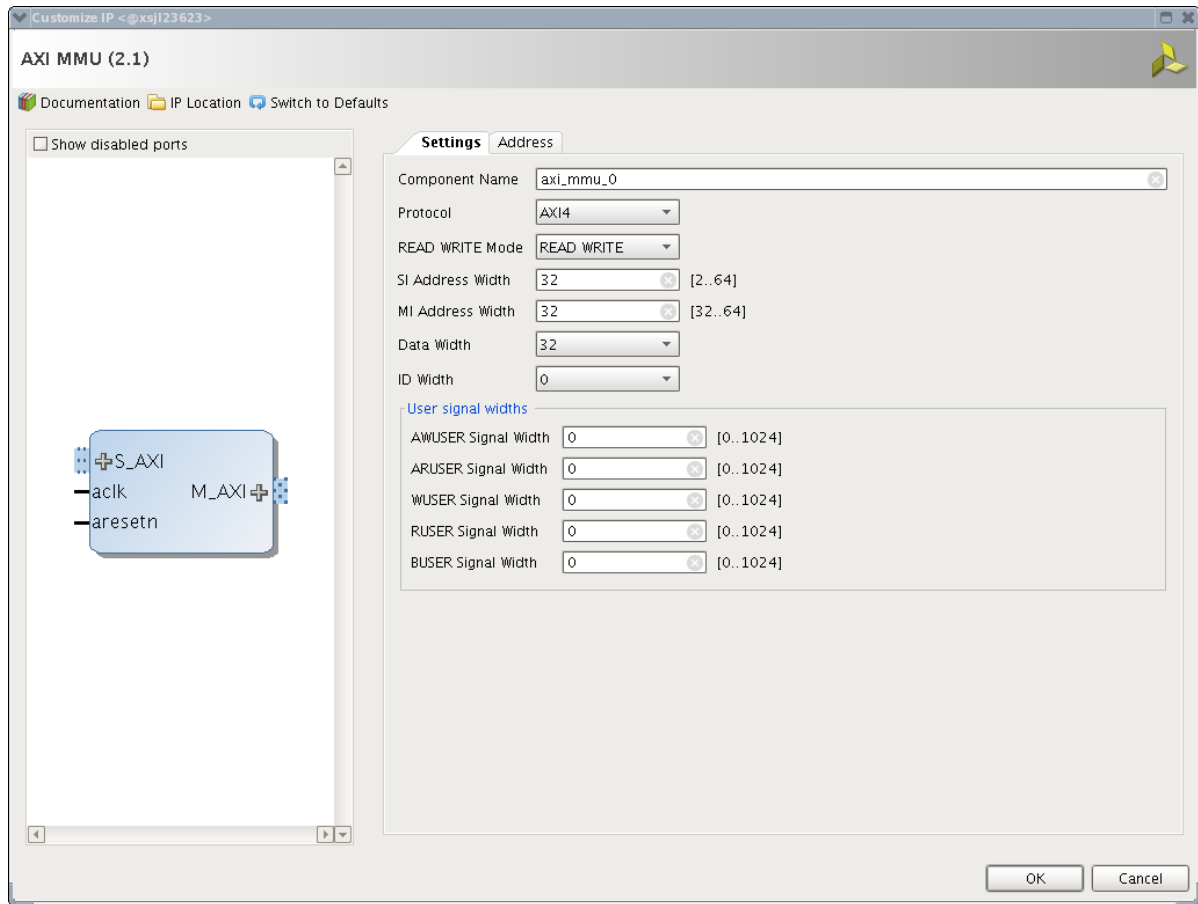


Figure 4-16: AXI MMU – Settings Tab

PROTOCOL

- Description: Protocol of all interfaces. Automatically set by tools based on system connectivity
- Format/Range: String (AXI4, AXI3, AXI4LITE)
- Default Value: AXI4

READ_WRITE Mode

- Description: Enables read channels and/or write channels for the whole IP. Automatically set by tools based on the master device connected to the S_AXI interface
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

SI Address Width

- Description: Width of read/write addr signals received on the SI interface. Automatically set by tools based on system connectivity.
- Format/Range: Integer (1-64)
- Default Value: 32

MI Address Width

- Description: Width of read/write addr signals issued on the MI interface. Automatically set by tools based on system connectivity.
- Format/Range: Integer (1-64); must be at least as wide as the SI Address Width
- Default Value: 32

Data Width

- Description: Data width of the Write and Read datapaths. Automatically set by tools based on system connectivity
- Format/Range: For AXI4 or AXI3: Integer (32, 64, 128, 256, 512, 1024); for AXI4-Lite: Integer (32, 64)
- Default Value: 32

ID Width

- Description: Width of all ID signals (if any) propagated by the core. Automatically set by tools based on system connectivity.
- Format/Range: Integer (0-32)
- Default Value: 0

User Signal Widths

- AWUSER_WIDTH
 - Description: Width of awuser signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

- ARUSER_WIDTH
 - Description: Width of `aruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- WUSER_WIDTH
 - Description: Width of `wuser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- RUSER_WIDTH
 - Description: Width of `ruser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0
- BUSER_WIDTH
 - Description: Width of `buser` signals (if any). Automatically set by tools based on system connectivity
 - Format/Range: Integer (0-1024)
 - Default Value: 0

AXI MMU – Address Tab

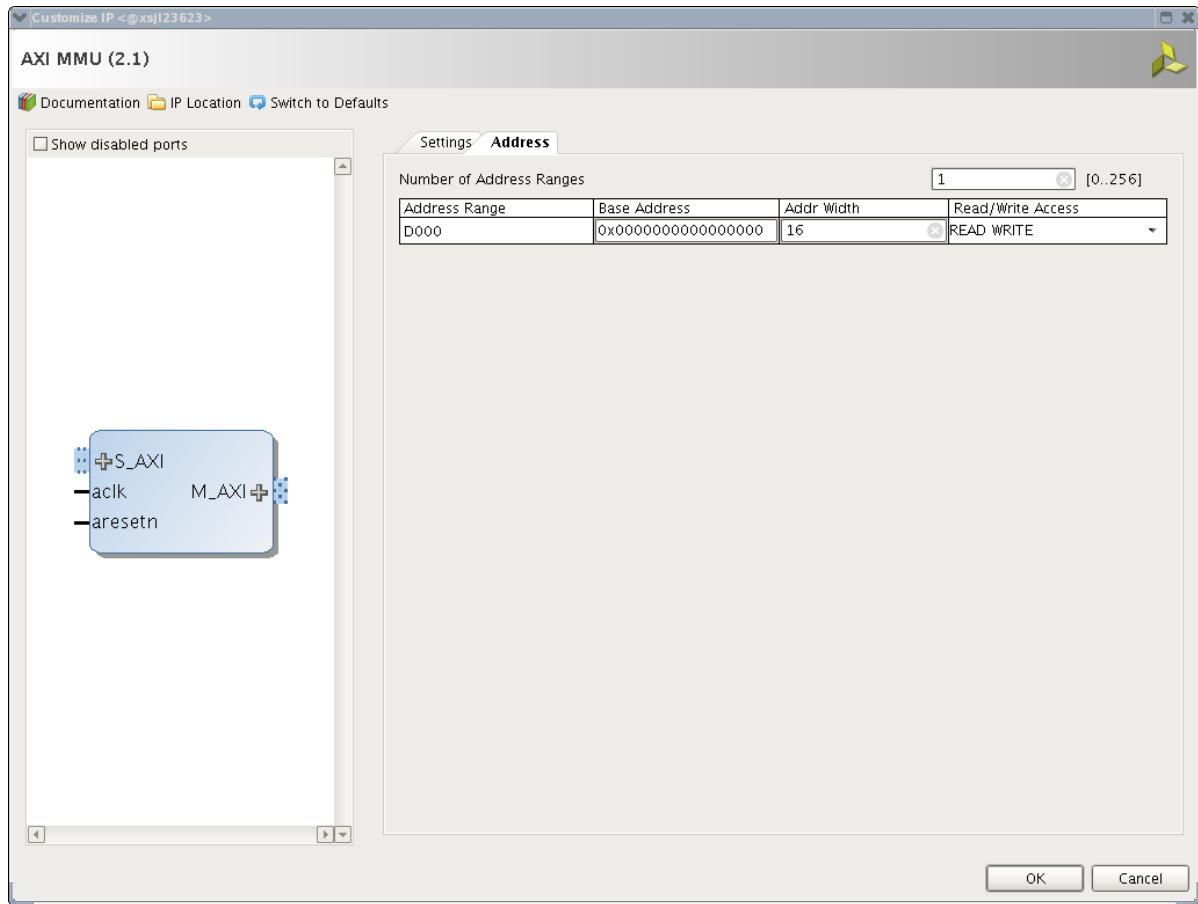


Figure 4-17: AXI MMU – Address Tab

Number of Address Ranges

- Description: Total number of Address Ranges (if any) for all endpoint slaves accessible through the MMU.
- Format/Range: Integer (0-256); 0 = reject all transactions
- Default Value: 1

Base Addr

- Description: Base address of each address range. All low-order bits of base address in the range [ADDR_WIDTH-1: 0] must be zero.
- Format/Range: Bit-string, 64-bits; each value must be less than $2^{SI_ADDR_WIDTH}$
- Default Value for each range "Dddd": $ddd * 0x10000$. (For unused ranges, Base Addr is ignored.)

Addr Width

- Description: Number of address bits representing the address space (in bytes) covered by each address range. The size of each address range is $2^{**}ADDR_WIDTH$ bytes.
- Format/Range: Integer (0-SI_ADDR_WIDTH); 0 indicates an unused range
- Default Value: 16

READ/WRITE Access

- Description: Enables each address range for read and/or write transactions
- Format/Range: String (READ_WRITE, READ_ONLY, WRITE_ONLY)
- Default Value: READ_WRITE

Output Generation

The AXI4 Interconnect deliverables are organized in the directory `<project name>/<project name>.srcs/sources_1/ip/<component name>` and is designated as the `<ip source dir>`.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Constraining the Core

In general, the only constraints required when deploying any of these cores is the specification of a clock period for each of the clock signals connected to all `ac1k` inputs of the core.

Required Constraints

Except for Register Slice configured for SLR Crossing, there are no constraints required for any cores covered in this Product Guide.

Floorplanning Constraints for Register Slice SLR Crossing Modes

When using the Register Slice core in either the SLR Crossing, SLR TDM Crossing or Multi SLR Crossing mode, constraints can be applied to explicitly floorplan the submodules of the core into adjacent SLRs. This will ensure that the SLR crossing(s) will take place between the intended flop-to-flop, unit-fanout, internal wires across all payload and handshake pathways within the core.

After synthesis, all logic and registers that should be placed into the master-side SLR (where the AXI master connected to the SI interface is located) will contain the cell name pattern `*slr_master*`. All logic and registers that should be placed into the slave-side SLR (where the AXI slave connected to the MI interface is located) will contain the cell name pattern `*slr_slave*`. When spanning 3 SLRs, all logic and registers that should be placed into the

middle SLR will contain the cell name pattern `*slr_middle*`. Constraints that combine the instance name of the Register Slice and any of these sub-module name patterns can then be used to group all cells in the core into their respective PBLOCKS for floorplanning.

In the following example, a Register Slice instance named `my_reg` is configured in Multi SLR Crossing mode (all channels) to cross two SLR boundaries that exist in the target device. One of the boundaries exists between row Y4 (the top of the lower SLR) and row Y5 (the bottom of the middle SLR). The other boundary exists between row Y9 (the top of the middle SLR) and row Y10 (the bottom of the upper SLR).

```
create_pblock lower_slr
add_cells_to_pblock [get_pblocks lower_slr] [get_cells -hierarchical -filter
"NAME=~*my_reg*slr_master*"]
resize_pblock [get_pblocks lower_slr] -add {CLOCKREGION_X0Y0:CLOCKREGION_X5Y4}
create_pblock center_slr
add_cells_to_pblock [get_pblocks center_slr] [get_cells -hierarchical -filter
"NAME=~*my_reg*slr_middle*"]
resize_pblock [get_pblocks center_slr] -add {CLOCKREGION_X0Y5:CLOCKREGION_X5Y9}
create_pblock upper_slr
add_cells_to_pblock [get_pblocks upper_slr] [get_cells -hierarchical -filter
"NAME=~*my_reg*slr_slave*"]
resize_pblock [get_pblocks upper_slr] -add {CLOCKREGION_X0Y10:CLOCKREGION_X5Y14}
```

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

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

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].



IMPORTANT: For cores targeting 7 series or Zynq®-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.

Synthesis and Implementation

Asynchronous Clock-Domain-Crossing Constraints

The AXI Clock Converter IP core and the AXI Data Width Converter IP core generate IP-level XDC constraint files, whether they are used stand-alone or as part of an AXI Interconnect core. When either is configured to perform asynchronous clock-domain-crossing, the generated XDC file contains **set_max_delay** constraints to prevent the resynchronized pathways from causing timing violations during static timing analysis. Otherwise, the XDC file is empty. If generated, the constraints use the periods of the connected clocks, defined at the system level, to derive their delay values. In order for these IP-level constraints to work properly, the following rules apply to your system-level timing constraints whenever the IP core performs asynchronous clock conversion:

1. Each of the nets connected to the `s_axi_aclk` and `m_axi_aclk` ports of the IP core must have exactly one clock defined on it, using either
 - a. a **create_clock** command on a top-level clock pin specified in your system XDC file, or
 - b. a **create_generated_clock** command, typically generated automatically by an IP core producing a derived clock signal, such as `clk_wiz`.
2. The `s_axi_aclk` and `m_axi_aclk` ports of the IP core should not be connected to the same clock source.

For more details about synthesis and implementation, see in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Timing Closure of AXI Memory-Mapped Connections Across SLRs in SSI Devices

This section describes how to apply the AXI Register Slice IP to pipeline the AXI pathways crossing between two SLR regions when targeting an SSI FPGA.

Assume two IP cores (A and B) with an AXI Memory-Mapped point-to-point connection that is known to cross from one Super Logic Region (SLR) to another:

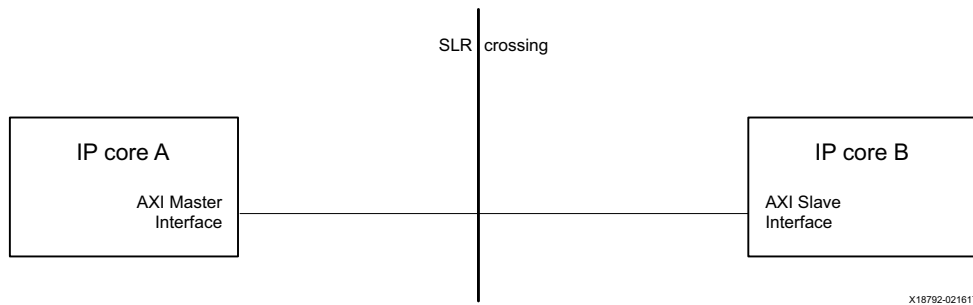


Figure 4-18: AXI Memory-Mapped Connections Across SLRs in SSI Devices

To facilitate timing closure of those AXI Memory-Mapped interfaces, crossing the SLRs with flop-to-flop paths is helpful. This can be accomplished, in one of two ways, by using the Vivado IP Integrator design entry:

Alternative 1: Using an instance of AXI Register Slice in Each SLR

1. Add two AXI Register Slice IP cores in sequence to the AXI interface connection of IP cores A and B:

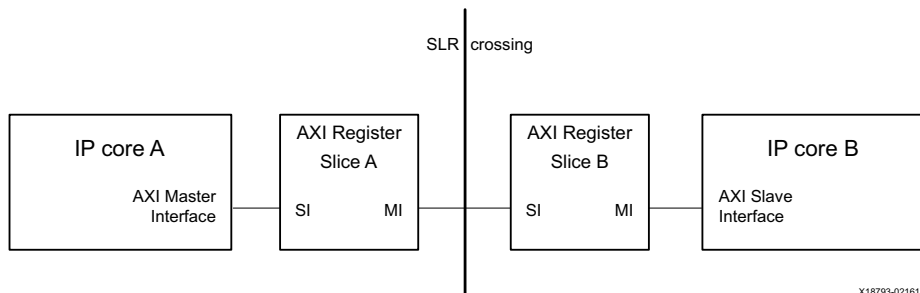


Figure 4-19: Adding Register Slice Cores

2. Double-click the first AXI Register Slice IP core A (which is connected to the AXI master interface of the source IP core A) and apply the following configuration options:

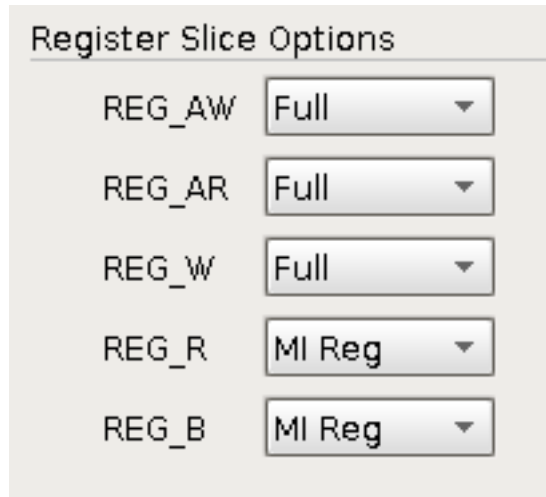


Figure 4-21: AXI Register Slice A Options

3. Add an XDC constraint to the design which places this AXI Register Slice IP core A in the same SLR as the source IP core A.
4. Double-click the second AXI Register Slice IP core B (which is connected to the AXI slave interface of the destination IP core B) and apply the following configuration options:

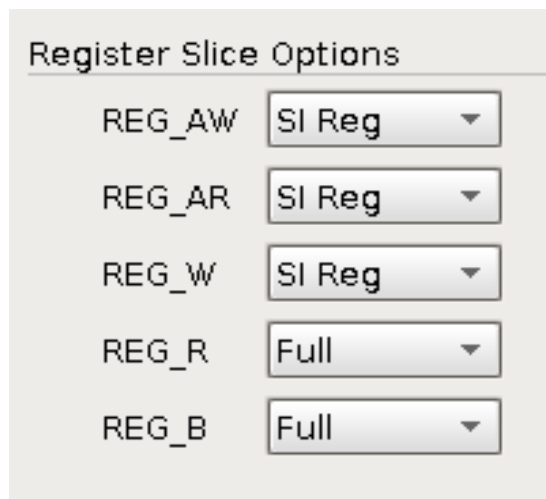


Figure 4-23: AXI Register Slice B Options

5. Add an XDC constraint to the design which places this AXI Register Slice IP core B in the same SLR as the destination IP core B.

See [AXI Register Slices in Chapter 3](#) for a description of the register options of the AXI Register Slice IP.

Alternative 2: Using a single instance of AXI Register Slice to span multiple SLRs

1. Add an AXI Register Slice IP core into the AXI interface connection of IP cores A and B.
2. Double-click the AXI Register Slice core and configure all Register Slice Options to one of the following alternative modes, as best suited to your application:
 - a. SLR Crossing: Pipelines only the crossing pathways between two adjacent SLRs.
 - b. SLR TDM Crossing pipelines: Only the crossing pathways between 2 adjacent SLRs using half the number of cross-SLR wires, but requiring a double-frequency `clock` input.
 - c. Multi-SLR Crossing pipelines a pathway that crosses two or more SLRs, and can further add pipeline stages to span the distance between the SLR boundaries and the connected endpoint IP.
3. Add an XDC constraint to the design which places the submodules within the AXI Register Slice core into the same SLRs as the connected endpoint IP.

See [Floorplanning Constraints for Register Slice SLR Crossing Modes](#) for more details.

Example Design

The AXI Interconnect core does not provide an example design.

Each of the AXI Infrastructure IP cores, except AXI Interconnect core, can generate an example design demonstrating its basic functionality when connected to a simple AXI master (traffic generator) and AXI slave. The example design system is customized to match the configuration settings you apply to the IP core. A test bench is provided to simulate the example design. The example design can also be implemented and analyzed using Vivado® Design Suite debug feature.

Upgrading

This appendix contains information about migrating a design from ISE® to the 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 included.

See the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

AXI Interconnect v2.1 core is not backward-compatible to any of the v1.x versions of the AXI Interconnect core. There is no migration automation available from those earlier versions.

All configurations of the AXI Interconnect v1.x core supported by the CORE Generator™ tool and XPS tool flows are supported by the AXI Interconnect v2.1 core.

Migration from CORE Generator System AXI Interconnect v1.x Core

Address Range

In the CORE Generator tool, AXI Interconnect v1.x core supported only N:1 connectivity and therefore assumed the M00_AXI interface was mapped to the entire address space, as defined by the ADDR_WIDTH parameter. The corresponding parameter settings in AXI Interconnect v2.0 core are:

- M00_A00_BASE_ADDR = 0
- M00_A00_ADDR_WIDTH = ADDR_WIDTH

Migration from XPS AXI Interconnect v1.x Core

Address Range

In XPS, AXI Interconnect v1.x core used `BASE_ADDR` and `HIGH_ADDR` to represent all address ranges.

In AXI Interconnect v2.1 core, set `Mmm_Aaa_ADDR_WIDTH` to the number of address bits corresponding to each range, which is $\log_2(\text{HIGH_ADDR} - \text{BASE_ADDR} + 1)$.

Range Check

There is no `RANGE_CHECK` parameter on AXI Interconnect v2.1 core. Range checking is enabled except when `NUM_SI = 1` and `NUM_MI = 1`.

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.

Parameter Changes

Protocol Converter Core

For the parameter Translation Mode, the value 1 (Protection) that was available in the v2.0 core is obsolete. The core no longer supports producing an error response back to a master that is not well-behaved when converting from AXI4 or AXI3 to AXI4-Lite, or from AXI4 to AXI3. During the upgrade to v2.1, any previously-configured core set to Translation Mode = 1 will automatically get set to Translation Mode = 2 (Conversion).

Port Changes

There have been no port changes between v2.0 and v2.1 of any core covered in this document.

Debugging

A common system malfunction that you can encounter when designing with any IP core with AXI interfaces can occur when custom IP (or non-production version of an IP) violates AXI protocol rules. Xilinx AXI IP cores do not contain any logic to guard against AXI protocol violations incurred by IP cores to which they are connected.

One of most common symptoms of an AXI protocol violation in a system is an apparent lock-up of a connected core. The AXI Interconnect core and the contained Crossbar core are especially vulnerable to protocol violations incurred by connected IP cores. When such a lock-up condition occurs, it often appears that an AXI channel transfer (VALID/READY handshake) completes on one interface of the Interconnect, but the resultant transfer is never issued on the expected output interface. Other possible symptoms include output transfers that appear to violate AXI transaction ordering rules.



RECOMMENDED: *Xilinx strongly recommends that you use the available protocol checker IP core to test for AXI protocol compliance before deploying any custom IP or IP with custom modifications.*

The clock conversion IP core, including Data Width Converter configured in FIFO Mode, can perform either synchronous or asynchronous clock conversion. When configured for synchronous conversion, it is required that the SI and MI clocks remain edge-aligned at all times. When using a Clock Wizard IP core to generate the AXI clocks, make sure the actual output clock frequencies maintain an integer-ratio relationship (1:2, 1:3... 1:16), as the actual frequency can differ from the requested frequency for each clock.

If the active edges of the SI and MI clocks drift apart during operation, you might observe a malfunction of the clock conversion IP core, including dropped transfers or lock-up. Xilinx clock conversion IP cores do not contain any logic to guard against misalignment of clock edges when configured in synchronous conversion mode. To help isolate a clock edge misalignment, reconfigure the clock conversion IP core to perform asynchronous conversion. If system functionality is restored, you should investigate the SI and MI clock waveforms for misalignment (or continue using asynchronous conversion, if acceptable).

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

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI Interconnect Core, the [Xilinx 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.

Documentation

This product guide is the main document associated with the AXI Interconnect Core. This guide, along with documentation related to all products that aid in the design process, can be found on the [Xilinx Support web page](#) or by using the Xilinx Documentation Navigator.

Download the Xilinx 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 a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

You can locate Answer Records for this core by using the Search Support box on the main [Xilinx support web page](#). To maximize your search results, use proper 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 the AXI Interconnect Core

AR: [54453](#)

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

Technical Support

Xilinx provides technical support in the [Xilinx Support web page](#) for this LogiCORE™ IP product when used as described in the product documentation. Xilinx 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 contact Xilinx Technical Support, navigate to the [Xilinx Support web page](#).

Debug Tools

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

Vivado Design Suite Debug Feature

The 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 Xilinx devices.

The Vivado logic analyzer is used with the logic debug 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) [\[Ref 9\]](#).

Interface Debug

The core has no accessible registers.

Debugging Guidance for AXI Interconnect Cores in the IP Integrator

IP integrator provides additional system and IP automation facilities which can, from time to time, cause you to encounter new warnings or error messages. Typically such warnings and errors are presented when the configuration of one IP core in a design is incompatible with the configuration of an IP core to which it has been connected. In many cases, the AXI Interconnect core will automatically attempt to resolve differences in the configuration of the master and slave IP cores to which it has been connected by adding a conversion IP core within the interconnect instance itself. To see which IP cores have been automatically added, use the **Expand hierarchy** buttons in the IP integrator canvas to explore within the AXI Interconnect instance's hierarchy.

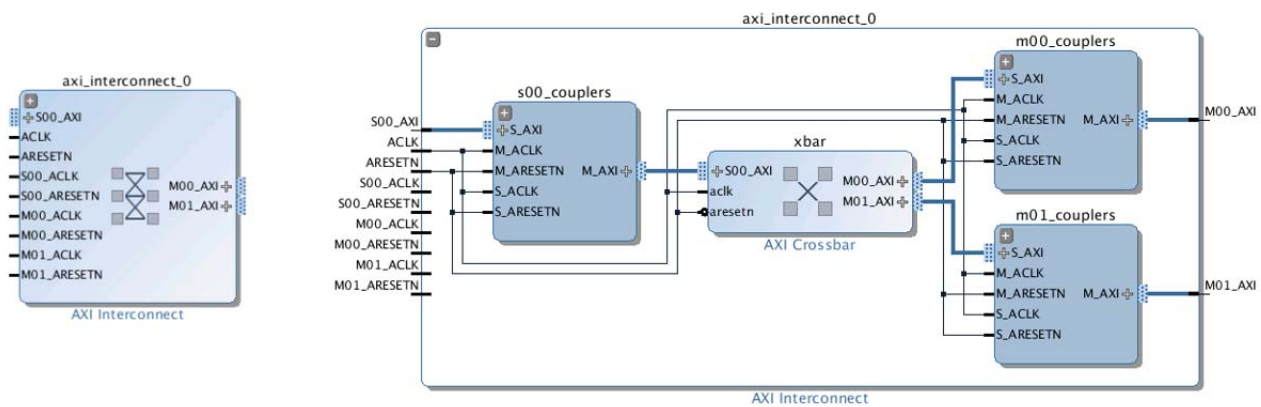


Figure B-1: Vivado IP Integrator Expand Hierarchy with AXI Interconnect Core

Note: IP integrator supports only "read-only" viewing of the contents of the AXI Interconnect core. It is possible to open the configuration Vivado IDE of each IP core within the AXI Interconnect instance to see its detailed configuration but the values cannot be changed.

If unexpected converter IP core instances are discovered while inspecting the AXI Interconnect core hierarchy, it is recommended to review the interface properties of the AXI Master or AXI slave connected to the AXI Interconnect core. The AXI interface properties of an IP interface can be seen by first selecting the interface on the IP core then opening the IP integrator **Block Interface Properties** panel (select **Window > Properties** from the Vivado menu bar if the properties panel is not currently shown).

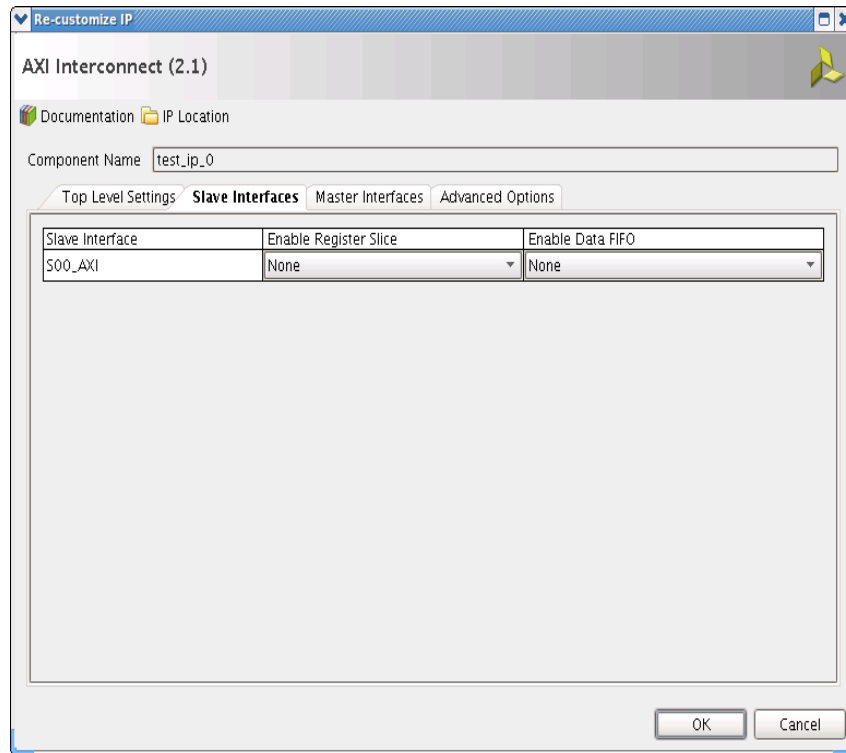


Figure B-2: Vivado IP Integrator Block Interface Properties Window Showing AXI Interface Properties

The interface properties of the AXI Master or Slave should be compared with the interface properties of the corresponding AXI Crossbar instance within the AXI Interconnect core. Certain differences in the values of the interface properties lead to the automatic addition of converter IP cores within the AXI Interconnect core. For example, if the DATA_WIDTH property differs between the AXI Master and the AXI Crossbar, a data width converter IP core will be inserted. If a conversion IP core is not desired, the configuration Vivado IDE of the AXI Master or AXI Slave IP core should be used to change the configuration of the IP core such that its interface properties are consistent with those found on the connected AXI Crossbar interface.

Definitions, Acronyms, and Abbreviations

Table C-1 provides a list of acronyms, abbreviations, and specific definitions used in this document.

Table C-1: Definitions, Acronyms, and Abbreviations

Item	Description
AXI	The generic term for all implemented AXI protocol interfaces.
master device or connected master	An IP core or device (or one of multiple interfaces on an IP core) that generates AXI transactions out from the IP core onto the wires connecting to a slave device.
slave device or connected slave	An IP core or device (or one of multiple interfaces on an IP core) that receives and responds to AXI transactions coming in to the IP core from the wires connecting to a master device.
master interface (generic)	An interface of an IP core or module that generates out-bound AXI transactions and thus is the initiator (source) of an AXI transfer. On AXI master interfaces, AWVALID, ARVALID, and WVALID are outputs, and RVALID and BVALID are inputs.
slave interface (generic)	An interface of an IP core or module that receives in-bound AXI transactions and becomes the target (destination) of an AXI transfer. On AXI slave interfaces, AWVALID, ARVALID, and WVALID are inputs, and RVALID and BVALID are outputs.
SI slot	Slave Interface Slot: A slice of the Slave Interface vector signals of the AXI Interconnect core that connect to a single master device.
MI slot	Master Interface Slot: A slice of the Master Interface vector signals of the AXI Interconnect core that connect to a single slave device.
SI-side	A module interface closer to the SI side of the AXI Interconnect core.
MI-side	A module interface closer to the MI side of the AXI Interconnect core.
Crossbar	Module at the center of the AXI Interconnect core that routes address, data and response channel transfers between various SI slots and MI slots.
SI hemisphere	Conversion and storage modules of the AXI Interconnect core located between on the SI side of the crossbar.
MI hemisphere	Conversion and storage modules of the AXI Interconnect core located on the MI side of the crossbar.
upsizer	Data width conversion function in which the datapath width gets wider when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction).
downsizer	Data width conversion function in which the datapath width gets narrower when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction).

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

References

To search for Xilinx documentation, go to <https://xilinx.com/support>

1. *AXI4-Stream LogiCORE IP Interconnect Product Guide (PG085)*
2. *ARM AMBA AXI Protocol v2.0 Specification (ARM IHI 0022C)*
3. *FIFO Generator LogiCORE IP Product Guide (PG057)*
4. *Vivado Design Suite User Guide: Designing with IP (UG896)*
5. *Vivado Design Suite User Guide: Getting Started (UG910)*
6. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator (UG994)*
7. *Vivado Design Suite User Guide: Logic Simulation (UG900)*
8. *ISE to Vivado Design Suite Migration Guide (UG911)*
9. *Vivado Design Suite User Guide: Programming and Debugging (UG908)*
10. *AXI Bus Functional Models User Guide (UG783)*
11. *Xilinx AXI Reference Guide (UG1037)*
12. *Zynq-7000 All Programmable SoC (Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics (DS187)*
13. *Zynq-7000 All Programmable SoC (Z-7030, Z-7035, Z-7045, and Z-7100): DC and AC Switching Characteristics (DS191)*
14. *SDAccel Platform Reference Design User Guide: Developer Board for Acceleration with KU115 (UG1234)*

15. Large FPGA Methodology Guide: Including Stacked Silicon Interconnect (SSI) Technology (UG872)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/20/2017	2.1	Added configuration modes to AXI Register Slice to cross SLRs in SSI devices.
04/05/2017	2.1	<ul style="list-style-type: none"> Added Registered Input to the AXI Register Slices section in Chapter 3. Added SI_Reg and MI_Reg to parameters in the AXI Register Slice Parameters section in Chapter 3 and to the Register Slice Options section in Chapter 4. Added SI_Reg/MI_Reg description to the note in Table 3-15. Added the Timing Closure of AXI Memory-Mapped Connections Across SLRs in SSI Devices section to Chapter 4. Added Automotive Disclaimer.
04/06/2016	2.1	<ul style="list-style-type: none"> Updated Kintex UltraScale performance information to Tables 2-10 through Table 2-14. Updated Table 2-1 through Table 2-15. Added new AXI MMU section. Modified the ID Width to be 4 for the AXI Clock Converter, AXI Data FIFO, AXI Protocol Converter, and AXI Register Slice sections. Added a note about the table specifications in the AXI Data FIFO, AXI Data Width Converter, AXI Protocol Converter, and AXI Register Slice sections. Updated the Common Configuration items in the AXI Crossbar Performance: SAMD, AXI4 Protocol, AXI Crossbar Performance: SASD, AXI4 Protocol, and AXI Crossbar Performance: SASD, AXI4-Lite Protocol sections. Updated the device part numbers in the Maximum Performance section.
11/18/2015	2.1	Added support for UltraScale+ families.
04/01/2015	2.1	<ul style="list-style-type: none"> Changes were made to the following sections in Chapter 3, Designing with the Core: Address Decode, Transaction Arbitration, AXI Upsizer, and AXI MMU Parameters. Added AXI MMU section.

Date	Version	Revision
10/01/2014	2.1.1	<p>Document only update.</p> <ul style="list-style-type: none"> Removed the following sentence from page 8: "Write and Read transactions are multiplexed to AXI4-Lite slave devices, propagating only a single address at a time, which typically avoids the duplication of logic resources otherwise associated with separate AXI write and read address signals." Added text to the Transaction Arbitration section on page 74. Removed Figure 3-2, Cascading AXI Interconnect Cores, and associated text. Modified text on page 89 in the last sentence of the first paragraph in the Conversion to AXI4-Lite section.
04/02/2014	2.1	<p>Added support for example design generation. Added AXI MMU.</p>
12/18/2013	2.1	<p>Added UltraScale™ architecture support.</p>
10/02/2013	2.1	<ul style="list-style-type: none"> Revision number advanced to 2.1 to align with core version number 2.1. Added support for example design generation. Updated Figures 4-1 through 4-5 in Chapter 4. Added the options Thread ID Width and Synchronization Stages. Updated information in the Migrating and Upgrading appendix. Updated the descriptions of Snn_HAS_REGSLICE, Mnn_HAS_REGSLICE and NUM_MI. Updated Figure B-1. Added Maximum Performance section to Chapter 2.
03/20/2013	1.2	<p>Updated Vivado Design Suite and core version 2.0. Updated Appendix B, Debugging.</p>
12/18/2012	1.1	<p>Updated to 2012.4 Vivado Design Suite. Updated Appendix B, Debugging.</p>
08/17/2011	1.0	<p>Initial Xilinx release as product guide release. This document is based on ds768.</p>

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

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 2012–2017 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. ARM and AMBA are registered trademarks of ARM in the EU and other countries. All other trademarks are the property of their respective owners.