

## Introduction

The AXI Interconnect IP (axi\_interconnect) connects one or more AXI memory-mapped Master devices to one or more memory-mapped Slave devices. The AXI interfaces conform to the AMBA® AXI version 4 specification from ARM®, including the AXI4-Lite control register interface subset.

**Note:** The AXI Interconnect IP is intended for memory-mapped transfers only; AXI4-Stream transfers are not applicable. IP with AXI4-Stream interfaces are generally connected to one another, and to DMA IP.

The AXI Interconnect IP is provided as an encrypted, non-licensed (free) pcore in the Xilinx Platform Studio software.

## Features

The AXI Interconnect IP contains the following features:

- AXI protocol compliant (AXI3, AXI4, and AXI4-Lite), and includes:
  - Burst lengths up to 256 for incremental (INCR) bursts
  - Converts AXI4 bursts >16 beats when targeting AXI3 slaves by splitting transactions.
  - Generates REGION outputs for slaves with multiple address decode ranges
  - Propagates USER signals on each channel, if any; independent USER signal width per channel (optional)
  - Propagates Quality of Service (QoS) signals, if any; not used by the AXI Interconnect (optional)

LogiCORE™ Facts	
Core Specifics	
Core Name	axi_interconnect
Supported Devices <sup>(1)</sup>	Virtex®-6, Spartan®-6
Design Tool Requirements	
Xilinx Implementation Tools	ISE® Design Suite 12.3 Xilinx Platform Studio (XPS)
Verification	N/A
Simulation	N/A
Synthesis	N/A
Resources Used	TBD
Provided with Core	
Documentation	See <a href="#">References, page 44</a>
Design File Formats	Verilog (encrypted)
Reference Designs Application Notes	N/A
Support	
Provided by <a href="http://www.xilinx.com">www.xilinx.com</a>	

1. For a complete listing of supported devices, see the release notes for this core. Also, refer to [www.xilinx.com/ise/embedded/ddsupport.htm](http://www.xilinx.com/ise/embedded/ddsupport.htm) for derivative device support.

- Interface data widths:
  - AXI4: 32, 64, 128 or 256 bits
  - AXI4-Lite: 32 bits 32-bit address width

(Features are continued on the following page)

## Features (*Continued*)

- Connects to 1-16 masters to 1-16 slaves:
  - When connecting one master to one slave, the AXI Interconnect can optionally perform address range checking. Also, it can perform any of the normal data-width, clock-rate, or protocol conversions and pipelining.
  - When connecting one master to one slave and not performing any conversions or address range checking, the AXI Interconnect is implemented as wires, with no resources, no delay and no latency.
- Built-in data-width conversion:
  - Each master and slave connection can independently use data widths of 32, 64, 128, or 256 bits wide:
    - The internal crossbar can be configured to have a native data-width of 32, 64, 128, or 256 bits
    - Data-width conversion is performed for each master and slave connection that does not match the crossbar native data-width
  - When converting to a wider interface (up-sizing), data is packed (merged) optionally, when permitted by address channel control signals (CACHE modifiable bit is asserted)
  - When converting to a narrower interface (down-sizing), burst transactions can be split into multiple transactions if the maximum burst length would otherwise be exceeded
- Built-in clock-rate conversion:
  - Each master and slave connection can use independent clock rates
  - Synchronous integer-ratio (N:1 and 1:N) conversion to the internal crossbar native clock-rate
  - Asynchronous clock conversion (uses more storage and incurs more latency than synchronous conversion)
  - The AXI Interconnect exports reset signals re-synchronized to the clock rate of each connected master and slave
- Built-in AXI4-Lite protocol conversion:
  - The AXI Interconnect can connect to any mixture of AXI4 and AXI4-Lite masters and slaves.
  - The AXI Interconnect saves transaction IDs and restores them during response transfers, when connected to an AXI4-Lite slave.
    - AXI4-Lite slaves do not need to sample or store IDs.
    - No additional adaptation logic is required between the AXI Interconnect and an AXI4-Lite slave.
  - The AXI Interconnect detects illegal AXI4-Lite transactions from AXI4 masters, such as any transaction that results in a burst of more than one word. The AXI Interconnect generates a protocol-compliant error response to the master, and does not propagate the illegal transaction to the AXI4-Lite slave.
  - Write and Read transactions are single-threaded to AXI4-Lite slaves, propagating only a single address at a time, which typically nullifies the resource overhead of separate write and read address signals.
- Built-in AXI3 protocol conversion:
  - The AXI Interconnect splits burst transactions of more than 16 beats from AXI4 masters into multiple transactions of no more than 16 beats when connected to an AXI3 slave.
- Optional register-slice pipelining:
  - Available on each AXI channel connecting to each master and each slave.

- Facilitates timing closure by trading-off frequency vs. latency.
- One latency cycle per register-slice, with no loss in data throughput under all AXI handshaking conditions.
- Optional data-path FIFO buffering:
  - Available on Write and Read data pathways connecting to each master and each slave.
  - 32-deep LUT-RAM based
  - 512-deep block RAM based
- Shared-Address, Multiple-Data (SAMD) crossbar:
  - 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 may occur independently and concurrently, provided AXI ordering rules are met.
  - Sparse crossbar data pathways 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.
- Supports multiple outstanding transactions:
  - Supports masters with multiple reordering depth (ID threads).
  - Supports up to 16-bit wide ID signals (system-wide).
  - 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.
- “Single-Slave per ID” method of cyclic dependency (deadlock) avoidance:
  - For each ID thread issued by a connected master, the master can have outstanding transactions to only one slave for Writes and one slave for Reads, at any 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 master that has reached its acceptance limit, or is targeting a slave that has reached its issuing limit, or is trying to access a slave in a manner that risks deadlock, is temporarily disqualified from arbitration, so that other masters may be granted arbitration
- Supports TrustZone security for each connected slave as a whole:
  - If configured as a secure slave, only secure AXI accesses are permitted
  - Any non-secure accesses are blocked and the AXI Interconnect returns a DECERR response to the master
- Support for Read-only and Write-only masters and slaves, resulting in reduced resource utilization.

## AXI Interconnect Limitations

- The AXI Interconnect does not support AXI3 atomic locked transactions; this feature was retracted by AXI4 protocol. A locked transaction is changed to a non-locked transaction and propagated to the slave.
- The AXI Interconnect does not support AXI3 Write interleaving; this feature was retracted by AXI4 protocol. AXI3 masters must be configured as if connected to a Slave with Write interleaving depth of one.
- AXI4 QoS signals do not influence arbitration priority. QoS signals are propagated from masters to slaves.
- The AXI Interconnect does not convert multi-beat bursts into multiple single-beat transactions when connected to an AXI4-Lite slave.
- The AXI Interconnect does not support low-power mode or propagate the AXI C channel signals.
- The AXI Interconnect does not time-out if the destination of any AXI channel transfer stalls indefinitely. All AXI slaves must respond to all received transactions, as required by AXI protocol.
- The AXI Interconnect provides no address remapping.
- The AXI Interconnect provides no built-in conversion to non-AXI protocols, such as APB.
- The AXI Interconnect does not have clock-enable (ACLKEN) inputs. Consequently, the use of ACLKEN is not supported among memory mapped AXI interfaces in Xilinx systems.

**Note:** (The ACLKEN signal is supported for Xilinx AXI4-Stream interfaces.)

## Definitions, Acronyms, and Abbreviations

The following table provides a list of acronyms, abbreviations and specific definitions used in this document.

**Table 1: Definitions, Acronyms, and Abbreviations**

Item	Description
master device	An IP having a master AXI interface that connects to the Slave Interface (SI) of the AXI Interconnect.
slave device	An IP having a slave AXI interface that connects to the Master Interface (MI) of the AXI Interconnect.
master interface (generic)	An interface of an IP or module that generates out-bound AXI transactions. On AXI master interfaces, <code>AWVALID</code> , <code>ARVALID</code> , and <code>WVALID</code> are outputs, and <code>RVALID</code> and <code>BVALID</code> are inputs.
slave interface (generic)	An interface of an IP or module that receives in-bound AXI transactions. On AXI slave interfaces, <code>AWVALID</code> , <code>ARVALID</code> , and <code>WVALID</code> are inputs, and <code>RVALID</code> and <code>BVALID</code> are outputs.
SI	AXI Interconnect Slave Interface: Vectored AXI slave interface receiving in-bound AXI transactions from all connected master devices.
MI	AXI Interconnect Master Interface: Vectored AXI master interface generating out-bound AXI transactions to all connected slave devices.
SI slot	Slave Interface Slot: A slice of the AXI Interconnect's Slave Interface vector signals that connect to a single master device.
MI slot	Master Interface Slot: A slice of the AXI Interconnect's Master Interface vector signals that connect to a single slave device.
SI-side	Refers to a module interface closer to the AXI Interconnect's SI.
MI-side	Refers to a module interface closer to the AXI Interconnect's MI.
Crossbar	Module at the center of the AXI Interconnect 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 located between the SI and crossbar
MI hemisphere	Conversion and storage modules of the AXI Interconnect located between the crossbar and MI.
Up-sizer	Data width conversion function in which the data path width gets wider when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction).
Down-sizer	Data width conversion function in which the data path width gets narrower when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction).

## Functional Description

The following figure shows the top-most AXI Interconnect block diagram.

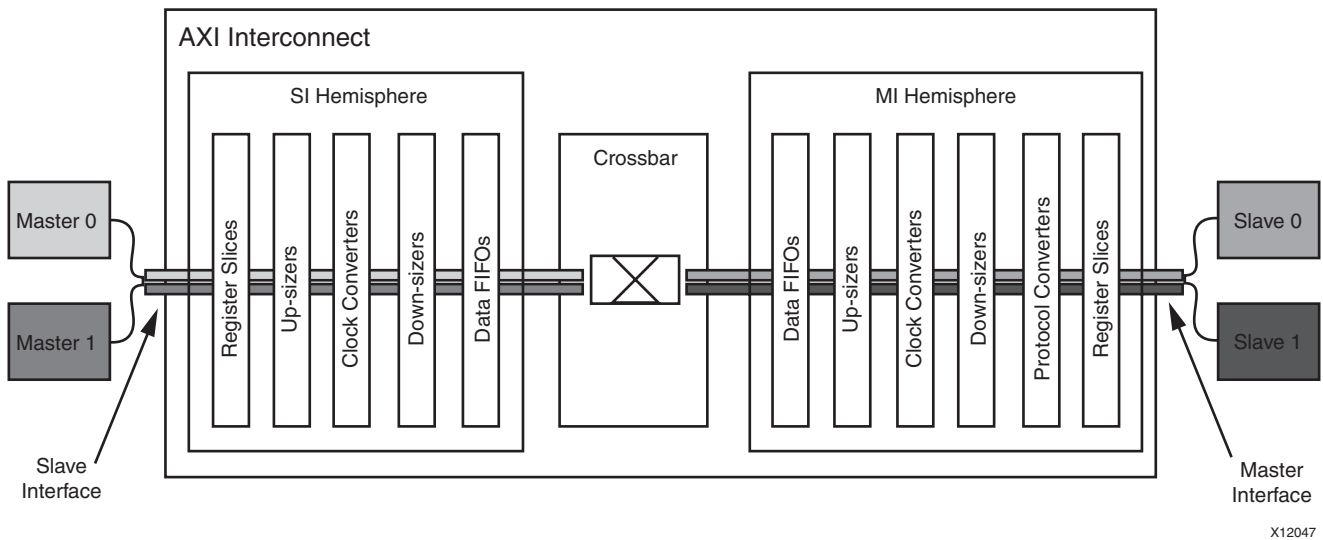


Figure 1: AXI Interconnect Diagram

The AXI Interconnect consists of the SI, the MI and the functional units that comprise the AXI channel pathways between them. The SI receives Write and Read transaction requests from connected master devices; the MI connects to slave devices. At the center is the “crossbar” that routes traffic on all the AXI channels between the various devices connected to the SI and MI. The AXI Interconnect also comprises other functional units located between the crossbar and each of the interfaces that perform various conversion and storage functions. The crossbar effectively splits the AXI Interconnect down the middle between the SI-related functional units (SI “hemisphere”) and the MI-related units (MI “hemisphere”). The following figure is a block diagram of the AXI Interconnect IP functional units.

The following subsection describes the use models for the AXI Interconnect.

### Use Models

The AXI Interconnect IP connects one or more AXI memory-mapped master devices to one or more memory-mapped slave devices. The following subsections describe the following use cases:

- [Pass Through](#)
- [N-to-1 Interconnect](#)
- [1-to-N Interconnect](#)
- [N-to-M Interconnect \(Sparse Crossbar Message Routing\)](#)

## Pass Through

When there is one master device and one slave device only connected to the AXI Interconnect, and the AXI Interconnect is not performing any optional conversion functions or pipelining, the AXI Interconnect degenerates into direct wire connections with no latency and consuming no logic resources. The following figure shows the Pass Through diagram.

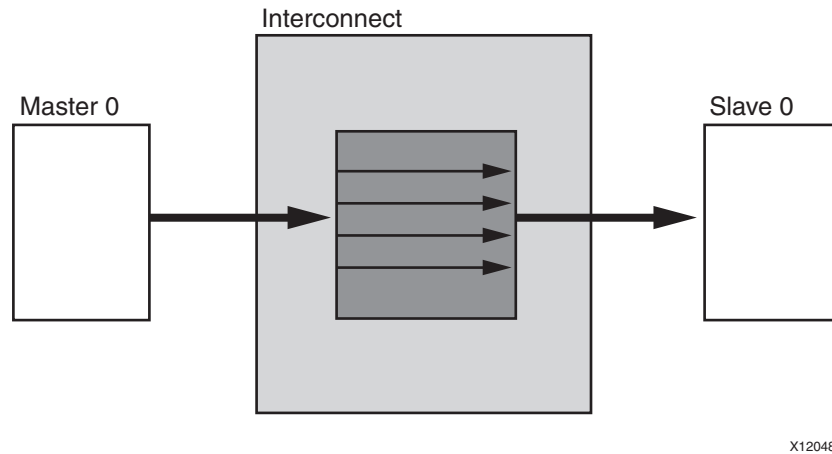


Figure 2: Pass-through AXI Interconnect Use Case

- AXI4-Lite slave adaptation
- AXI-3 slave adaptation
- Pipelining, such as a register slice or data channel FIFO

In these cases, the AXI Interconnect contains no arbitration, decoding, or routing logic. There could be latency incurred, depending on the conversion being performed.

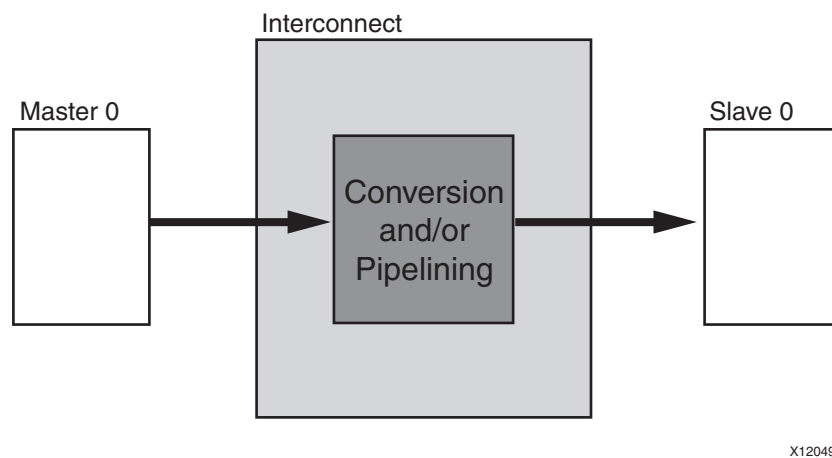


Figure 3: 1-to-1 Conversion AXI Interconnect Use Case

## N-to-1 Interconnect

A common degenerate configuration of AXI Interconnect is when multiple master devices arbitrate for access to a single slave device, typically a memory controller.

In these cases, address decoding logic might be unnecessary and omitted from the AXI Interconnect (unless address range validation is needed). Conversion functions, such as data width and clock rate conversion, can also be performed in this configuration as shown in the following figure.

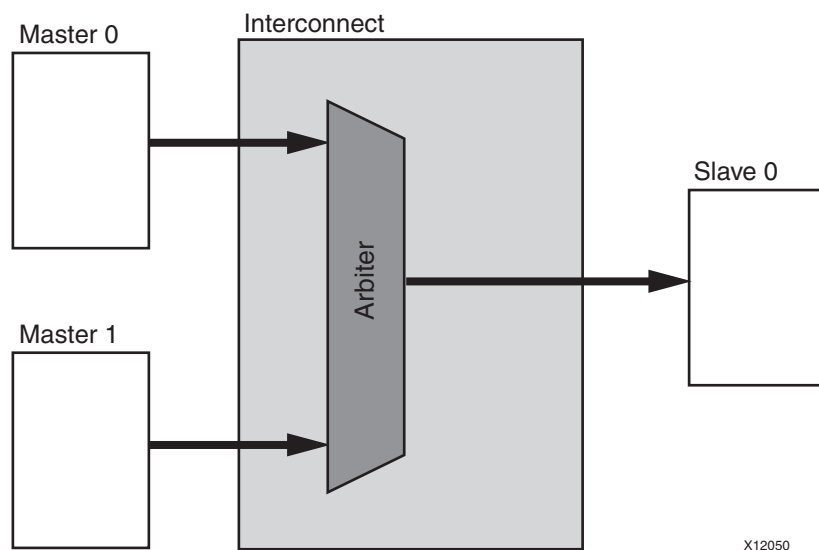


Figure 4: N-to-1 AXI Interconnect

## 1-to-N Interconnect

Another degenerative configuration of the AXI Interconnect is when a single master device, typically a processor, accesses multiple memory-mapped slave peripherals. In these cases, arbitration (in the address and Write data paths) is not performed, as shown in the following figure.

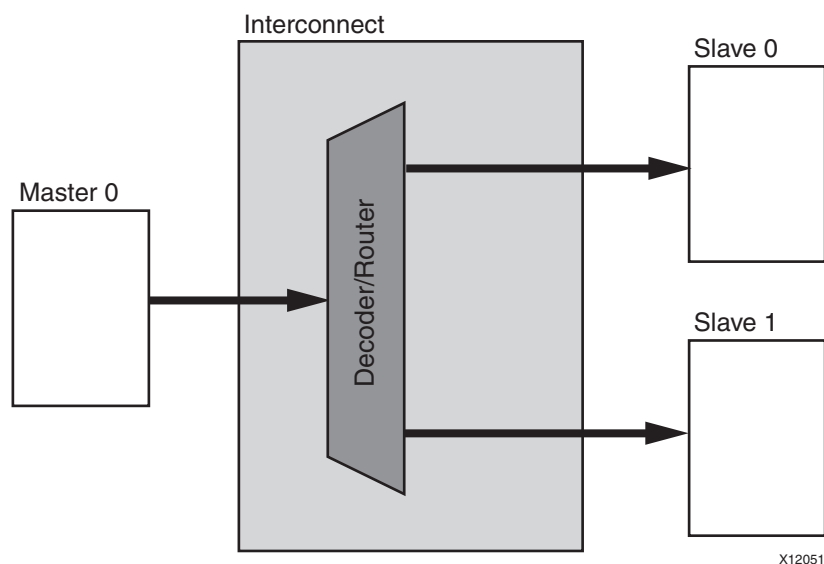
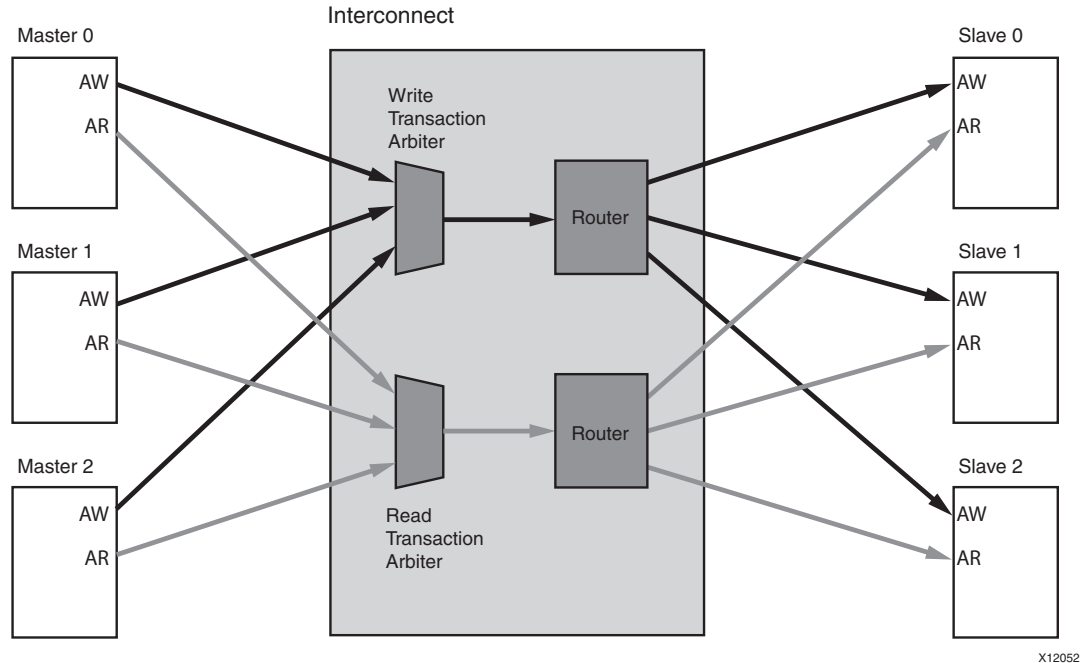


Figure 5: 1-to-N AXI Interconnect Use Case

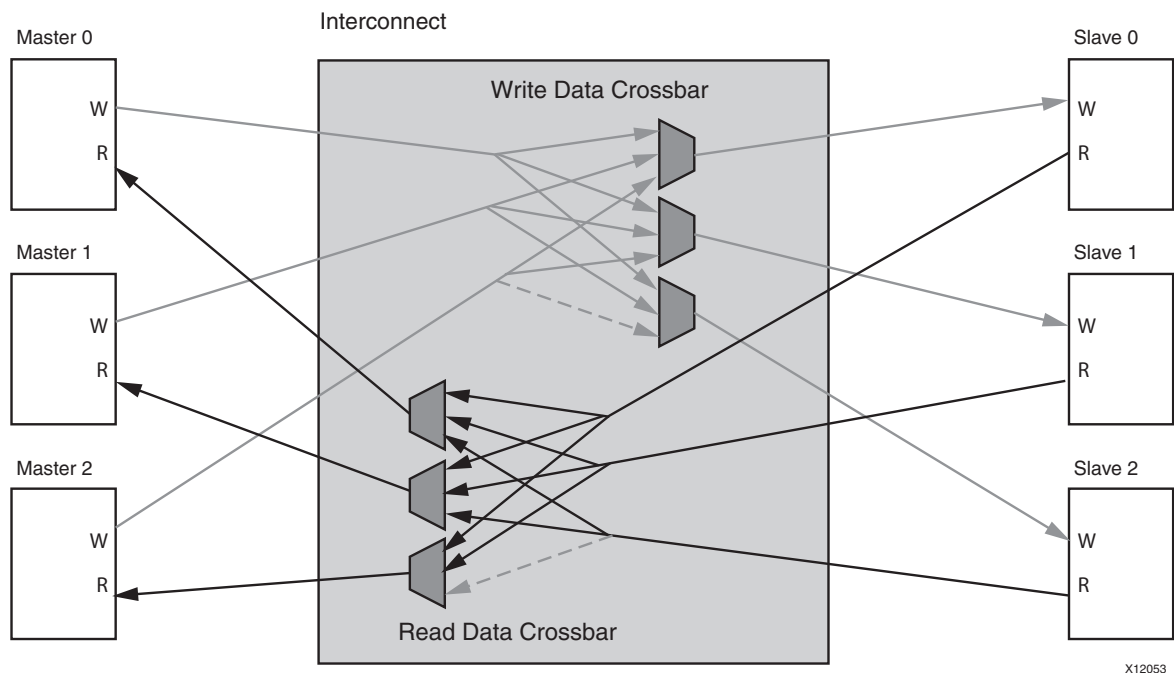


## N-to-M Interconnect (Sparse Crossbar Message Routing)

The N-to-M use case of the AXI Interconnect features a Shared-Address Multiple-Data (SAMD) topology, consisting of sparse data crossbar connectivity, with single-threaded Write and Read address arbitration, as shown in Figure 6 and Figure 7.



**Figure 6: Shared Write and Read Address Arbitration**



**Figure 7: 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.

The Write address channels among all SI slots (if > 1) feed into a central address arbiter, which grants access to one SI slot at a time, as 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.

## AXI Interconnect Functionality

The following subsections describe the functionality within the AXI Interconnect IP.

- [Top-Level Slave/Master Interface](#)
- [Width Conversion](#)
- [Width Conversion Transactions](#)
- [Pipelining](#)
- [Peripheral Register Slices](#)
- [Data Path FIFOs](#)
- [Use of ID Signals](#)
- [Multiple Address Range Support](#)
- [Cyclic Dependency Avoidance](#)
- [Error Signaling](#)

### Top-Level Slave/Master Interface

The top-level interface consist 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.
- For each signal comprising a vectored AXI interface on the core, its natural width is multiplied by the number 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 `AWLEN` signal carries an 8-bit value indicating the number of data beats in a Write transaction. If the AXI Interconnect is configured with 2 SI slots, then the `S_AXI_AWLEN` signal would be a total of 16 bits wide.
- The effective widths of the `WDATA`, `WSTRB`, and `RDATA` signals are also configurable per MI/SI. The width of each of these signals on the vectored SI or MI is the maximum supported width of the signal (currently 256 bits for `DATA` signals) multiplied by the number of slots. The unused high-order bits for each narrower slot are tied off (for inputs) or left unconnected (for outputs) within the AXI Interconnect, and are trimmed by the implementation tools. Each AXI interface signal therefore allocates the same physical width across all slots.

For example, if the AXI Interconnect core is configured with two SI slots, one with data width 32 bits and one with data width 128 bits, each of the `WDATA` and `RDATA` signals on the SI of the core would be a total of 512 bits wide, as shown in [Figure 8, page 11](#).

Specifically:

- “slot 0” would use WDATA[31:0]
- “slot 1” would use WDATA[383:256], which would connect to WDATA[127:0] of the “Master 1” IP).
- WDATA[255:32] would remain tied-off or unconnected inside the AXI Interconnect, as would WDATA[511:384].

Similar to I/O signals, many configuration parameters on the AXI Interconnect are also formatted as vectors across all the SI or MI slots. Vectored parameters are formatted as follows:

- Parameters that define Boolean conditions, such as the TrustZone security indicator (C\_M\_AXI\_SECURE), are formatted as bit vectors with one bit per slot.
- Parameters that define any numerical value, regardless of value range, are formatted as bit vectors with 32 bits per slot.
- Base and high addresses are an exception and are formatted as 64-bits per slot.

In the example, the value of the vectored parameter that defines the effective data widths of the slots on the SI (C\_S\_AXI\_DATA\_WIDTH) would be 0x0000008000000020, where 0x20 represents 32-bits for slot 0, and 0x80 represents 128-bits for slot 1.

Parameter values are little-endian, as are I/O signals, and so the value corresponding to slot 0 appears at the right-hand Least Significant Bit (LSB) end of the parameter vector. The following figure shows a vectored Slave/Master interface.

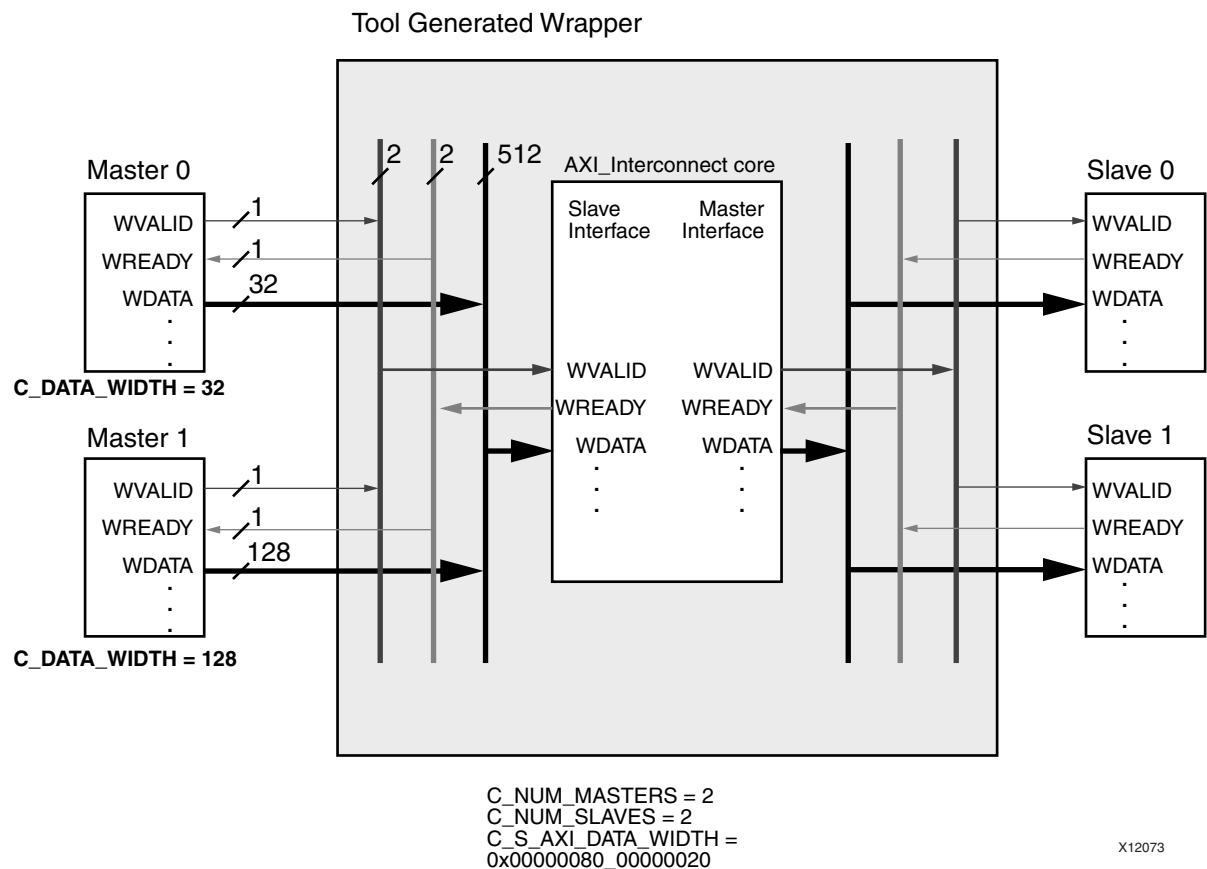


Figure 8: Vectored Slave/Master Interface

## Width Conversion

The AXI Interconnect core has a parametrically-defined, internal, native data-width, which supports 32, 64, 128, and 256 bits. The AXI data channels that span the crossbar are sized to the “native” width of the AXI Interconnect, as specified by the `C_INTERCONNECT_DATA_WIDTH` parameter.

When any SI slots or MI slots are sized differently, the AXI Interconnect inserts width conversion units to adapt the slot width to the AXI Interconnect native width before transiting the crossbar to the other hemisphere.

The width conversion functions differ depending on whether the data path width gets wider (“up-sizing”) or more narrow (“down-sizing”) 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 the SI to the AXI Interconnect native width) or the MI hemisphere (translating from the AXI Interconnect native width to the MI).

MI and SI slots have an associated individual parametric data-width value. The AXI Interconnect adapts each MI and SI slot automatically to the internal native data-width as follows:

- When the data width of an SI slot is wider than the internal native data width of the AXI Interconnect, a down-sizing conversion is performed along the pathways of the SI slot.
- When the internal native data width of the AXI Interconnect is wider than that of an MI slot, a down-sizing conversion is performed along the pathways of the MI slot.
- When the data width of an SI slot is narrower than the internal native data width of the AXI Interconnect, an up-sizing conversion is performed along the pathways of the SI slot.
- When the internal native data width of the AXI Interconnect is narrower than that of an MI slot, an up-sizing conversion is performed along the pathways of the MI slot.

Typically, the data-width of the AXI Interconnect is matched to the smaller of the widest SI slot or the widest MI slot in the system design.

The following subsections describe the down-sizing and up-sizing behavior.

### ***Down-sizing***

Down-sizers used in pathways connecting wide master devices are equipped to split burst transactions that might exceed the maximum AXI burst length (even if such bursts are never actually needed).

When the data width on the SI side is wider than that on the MI side and the transfer size of the transaction is also wider than the data width on the MI side, then down-sizing is performed and, in the transaction issued to the MI side, the number of data beats is multiplied accordingly.

- For writes, data serialization occurs
- For reads, data merging occurs

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, and data transfers pass through unchanged except for byte-lane steering. This applies to both writes and reads.

When down-sizing, the AXI Interconnect 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). In such cases, the AXI Interconnect splits the transaction automatically into multiple conforming burst transactions.

- When a down-sized Write transaction results in splitting, the AXI Interconnect coalesces the multiple Write responses from the slave device and issues one Write response back to the master device, accumulating the worst-case occurring error condition (`BRESP`).

- When a down-sized Read transaction results in splitting, the AXI Interconnect sets the `RRESP` for each output data beat (to the master device) according to the worst-case error condition among the input data beats being merged.

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

Refer to [Table 2, page 15](#) for the various size conversions.

### **Up-sizing**

For upszers in the SI hemisphere, 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.
- For reads, data serialization occurs.

When the `AW/ARCACHE[1]` bit is de-asserted, 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.”

For upszers in the MI hemisphere, typically only expanders are used. (An exception is that MI-side upszers must pack when `C_M_AXI_SUPPORTS_NARROW_BURST = 0` for a MI slot; see [AXI Interconnect Parameter Usage, page 34](#).) This applies to both Writes and Reads.

Up-sizing never results in transaction splitting. Refer to [Table 2, page 15](#) for the various size conversions.

### **WIDTH Conversion Transaction Transformations**

The following table uses these following designators when describing properties, signals, and derived equations:

- `si` = Slave Interface
- `cb` = Interconnect (crossbar) core
- `mi` = Master Interface

The following tables list SI-hemisphere transformations when:

- Relative `DWidth` compares `si.DW` to `cb.DW`
- MI-hemisphere transformations when Relative `DWidth` compares `cb.DW` to `mi.DW`

### **Supporting Equations**

The following are the width conversion equations which are enumerated in [Table 2, page 15](#).

1. When width conversion results in a change in transaction length, the output `SIZE` is always the same as the output `DATA_WIDTH`.
2.  $si.DW = C\_S\_AXI\_DATA\_WIDTH$
3.  $cb.DW = C\_INTERCONNECT\_DATA\_WIDTH$
4.  $mi.DW = C\_M\_AXI\_DATA\_WIDTH$
5.  $si.Bytes = si.DW^{[2]} / 8$
6.  $cb.Bytes = cb.DW^{[3]} / 8$

7.  $mi.Bytes = mi.DW^{[4]} / 8$
8.  $cb.ByteMask = cb.Bytes^{[5]} - 1$
9.  $mi.ByteMask = mi.Bytes^{[6]} - 1$
10.  $si.SIZE = S\_AXI\_AWSIZE$  or  $S\_AXI\_ARSIZE$ , as applicable
11.  $cb.SIZE = si.SIZE$  if  $(cb.LEN=si.LEN)$ , else  $\log_2(cb.Bytes^{[6]})$
12.  $mi.SIZE = cb.SIZE$  if  $(mi.LEN=cb.LEN)$ , else  $\log_2(mi.Bytes^{[7]})$
13.  $si.SizeMask = (2^{**}si.SIZE^{[10]}) - 1$
14.  $cb.SizeMask = (2^{**}cb.SIZE^{[11]}) - 1$
15.  $mi.SizeMask = (2^{**}mi.SIZE^{[12]}) - 1$
16.  $cb.AlignedStart = si.ADDR \& \sim cb.ByteMask^{[8]}$
17.  $cb.AlignedEnd = ((si.ADDR \& \sim si.SizeMask^{[13]}) + (si.LEN * 2^{**}si.SIZE^{[10]})) \& \sim cb.ByteMask^{[9]}$
18.  $cb.upsized\_LEN = (cb.AlignedEnd^{[17]} - cb.AlignedStart^{[16]}) / cb.Bytes^{[6]}$
19.  $mi.AlignedStart = cb.ADDR \& \sim mi.ByteMask^{[9]}$
20.  $mi.AlignedEnd = ((cb.ADDR \& \sim cb.SizeMask^{[13]}) + (cb.LEN * 2^{**}cb.SIZE^{[11]})) \& \sim mi.ByteMask^{[9]}$
21.  $mi.upsized\_LEN = (mi.AlignedEnd^{[20]} - mi.AlignedStart^{[19]}) / mi.Bytes^{[4]}$
22.  $si.conv\_ratio = (2^{**}si.SIZE^{[10]}) / cb.Bytes^{[8]}$
23.  $cb.conv\_ratio = (2^{**}cb.SIZE^{[10]}) / mi.Bytes^{[9]}$
24.  $si.downsize\_LEN = (si.LEN+1) * si.conv\_ratio - 1^{[22]}$
25.  $cb.downsize\_LEN = (cb.LEN+1) * cb.conv\_ratio - 1^{[23]}$
26.  $cb.AlignedAdjustment = (si.ADDR \& si.SizeMask^{[13]} \& \sim cb.ByteMask^{[8]}) / cb.Bytes^{[6]}$
27.  $mi.AlignedAdjustment = (cb.ADDR \& cb.SizeMask^{[14]} \& \sim mi.ByteMask) / mi.Bytes^{[9]}$
28.  $si.burst\_bytes = 2^{**}si.SIZE^{[10]} * (si.LEN+1)$
29.  $cb.burst\_bytes = 2^{**}cb.SIZE^{[11]} * (cb.LEN+1)$
30.  $si.burst\_mask = si.burst\_bytes^{[28]} - 1$
31.  $cb.burst\_mask = cb.burst\_bytes^{[29]} - 1$
32.  $si.wrap\_address = si.ADDR \& \sim si.burst\_mask^{[30]}$
33.  $cb.wrap\_address = cb.ADDR \& \sim cb.burst\_mask^{[31]}$
34.  $si.wrap1\_LEN = (si.burst\_bytes^{[28]} - (si.ADDR \& si.burst\_mask^{[30]})) / cb.Bytes - 1^{[8]}$
35.  $cb.wrap1\_LEN = (cb.burst\_bytes^{[29]} - (cb.ADDR \& cb.burst\_mask^{[31]})) / mi.Bytes - 1^{[7]}$
36.  $si.wrap2\_LEN = (si.ADDR \& si.burst\_mask^{[30]}) / cb.Bytes - 1^{[6]}$
37.  $cb.wrap2\_LEN = (cb.ADDR \& cb.burst\_mask^{[31]}) / mi.Bytes - 1^{[7]}$

**Note:** "x%y" denotes x modulo y.

## Width Conversion Transactions

Table 2: Width Conversion Transactions

Relative DWidth	Conditions	Output Trans	Output LEN	Output ADDR	Output BURST
<b>INCR Bursts</b>					
si.DW [2] = cb.DW [3]	always	1	No change	No change	INCR
cb.DW [3] = mi.DW [4]	always	1	No change	No change	INCR
si.DW [2] > cb.DW [3]	if (2**si.SIZE [10] <= cb.Bytes [6])	1	No change	No change	INCR
	else if (si.downsize_LEN [24] <= 255)	1	si.downsize_LEN [24] - cb.AlignedAdjustment [26]	No change	INCR (1)
	else	ceil ((si.downsize_LEN+1[24]) / 256)	first = 255 - cb.AlignedAdjustment [26]; last = si.downsize_LEN [24] % 256; others = 255	first = si.ADDR; others = (cb.ADDR[i-1] & ~si.SizeMask [13]) + (256*cb.Bytes [6])	INCR (1)
cb.DW[3] > mi.DW [4]	if (2**cb.SIZE [11] <= mi.Bytes [7])	1	No change	No change	INCR
	else if (cb.downsize_LEN [25] <= 255)	1	cb.downsize_LEN [25] - mi.AlignedAdjustment [27]	No change	INCR (1)
	else	ceil ((cb.downsize_­ LEN+1[25]) / 256)	first = 255 - mi.AlignedAdjustment [27]; last = cb.downsize_LEN [25] % 256; others = 255	first = cb.ADDR; others = (mi.ADDR[i-1] & ~cb.SizeMask [14]) + (256*mi.Bytes [24])	INCR (1)
si.DW [2] < cb.DW [3]	si.CACHE[1])	1	cb.upsample_LEN [18]	No change	INCR (1)
	else	1	No change	No change	INCR
cb.DW [3] < mi.DW [4]	if ((C_M_AXI_SUPPORTS_­ NARROW_BURST=0)	1	mi.upsample_LEN [21]	No change	INCR (1)
	else	1	No change	No change	INCR

1. When width conversion results in a change in transaction length, the output SIZE is always the same as the output DATA\_WIDTH.

Table 2: Width Conversion Transactions (Cont'd)

Relative DWidth	Conditions	Output Trans	Output LEN	Output ADDR	Output BURST
<b>WRAP Bursts</b>					
si.DW <sup>[2]</sup> = cb.DW <sup>[3]</sup>	always	1	No change	No change	WRAP
cb.DW <sup>[3]</sup> = mi.DW <sup>[4]</sup>	always	1	No change	No change	WRAP
si.DW <sup>[2]</sup> > cb.DW <sup>[3]</sup>	if (2**si.SIZE <sup>[10]</sup> <= cb.Bytes)	1	No change	No change	WRAP
	else if (si.downsize_LEN <sup>[24]</sup> <= 15)	1	si.downsize_LEN <sup>[24]</sup>	No change	WRAP <sup>(1)</sup>
	else if ((si.ADDR & si.burst_mask <sup>[30]</sup> ) == 0)	1	si.wrap1_LEN <sup>[34]</sup>	si.ADDR	INCR <sup>(1)</sup>
	else	2	first = si.wrap1_LEN <sup>[34]</sup> ; second = si.wrap2_LEN <sup>[36]</sup>	first = si.ADDR; second = si.wrap_address <sup>[32]</sup>	INCR <sup>(1)</sup>
cb.DW <sup>[3]</sup> > mi.DW <sup>[4]</sup>	if (2**cb.SIZE <sup>[11]</sup> <= mi.Bytes)	1	No change	No change	WRAP
	else if (cb.downsize_LEN <sup>[25]</sup> <= 15)	1	cb.downsize_LEN <sup>[25]</sup>	No change	WRAP <sup>(1)</sup>
	else if ((cb.ADDR & cb.burst_mask <sup>[30]</sup> ) == 0)	1	cb.wrap1_LEN <sup>[35]</sup>	cb.ADDR	INCR <sup>(1)</sup>
	else	2	first = cb.wrap1_LEN <sup>[35]</sup> ; second = cb.wrap2_LEN <sup>[37]</sup>	first = cb.ADDR; second = cb.wrap_address <sup>[33]</sup>	INCR <sup>(1)</sup>
si.DW <sup>[2]</sup> < cb.DW <sup>[3]</sup> , Write	if (si.CACHE[1])	1	ceil((si.LEN+1) * (2**si.SIZE <sup>[10]</sup> ) / cb.Bytes) - 1	si.wrap_address <sup>[32]</sup> + (ceil((si.ADDR & si.burst_mask <sup>[30]</sup> ) / cb.Bytes) * cb.Bytes) % si.burst_bytes <sup>[28]</sup>	If (cb.LEN>0) then WRAP: else INCR <sup>(1)</sup>
	else	1	No change	No change	WRAP
si.DW <sup>[2]</sup> < cb.DW <sup>[3]</sup> , Read	if (si.CACHE[1])	1	ceil((si.LEN+1) * (2**si.SIZE <sup>[10]</sup> ) / cb.Bytes <sup>[6]</sup> ) - 1	si.wrap_address <sup>[33]</sup> + (int((si.ADDR & si.burst_mask <sup>[30]</sup> ) / cb.Bytes <sup>[6]</sup> ) * cb.Bytes <sup>[6]</sup> )	(If cb.LEN>0) then WRAP: else INCR <sup>(1)</sup>
	else	1	No change	No change	WRAP

1. When width conversion results in a change in transaction length, the output SIZE is always the same as the output DATA\_WIDTH.



Table 2: Width Conversion Transactions (Cont'd)

Relative DWidth	Conditions	Output Trans	Output LEN	Output ADDR	Output BURST
cb.DW <sup>[3]</sup> < mi.DW <sup>[4]</sup> , Write	if ((C_M_AXI_SUPPORTS_NARROW_BURST=0)	1	$\text{ceil}((\text{cb.LEN}+1) * (2^{**}\text{cb.SIZE}^{[11]}) / \text{mi.Bytes}^{[7]}) - 1$	$\text{cb.wrap\_address}^{[33]} + (\text{ceil}((\text{cb.ADDR} \& \text{cb.burst\_mask}^{[31]}) / \text{mi.Bytes}^{[7]}) * \text{mi.Bytes}^{[7]}) \% \text{cb.burst\_bytes}^{[29]}$	If (mi.LEN>0) then WRAP: else INCR <sup>(1)</sup>
	else	1	No change	No change	WRAP
cb.DW <sup>[3]</sup> < mi.DW <sup>[4]</sup> , Read	if ((C_M_AXI_SUPPORTS_NARROW_BURST=0) & cb.CACHE[1])	1	1. $\text{ceil}((\text{cb.LEN}+1) * (2^{**}\text{cb.SIZE}^{[11]}) / \text{mi.Bytes}^{[7]}) - 1$	$\text{cb.wrap\_address}^{[33]} + (\text{int}((\text{cb.ADDR} \& \text{cb.burst\_mask}^{[31]}) / \text{mi.Bytes}^{[7]}) * \text{mi.Bytes}^{[7]})$	If (mi.LEN>0) then WRAP: else INCR <sup>(1)</sup>
	else	1	No change	No change	WRAP
<b>FIXED Bursts</b>					
si.DW <sup>[2]</sup> = cb.DW <sup>[3]</sup>	always	1	No change	No change	FIXED
cb.DW <sup>[3]</sup> = mi.DW <sup>[4]</sup>	always	1	No change	No change	FIXED
si.DW <sup>[2]</sup> > cb.DW <sup>[3]</sup>	if ( $2^{**}\text{si.SIZE}^{[11]} \leq \text{cb.Bytes}^{[6]}$ )	1	No change	No change	FIXED
	else	si.LEN+1	$\text{all} = \max(\text{si.conv\_ratio}^{[22]} - \text{cb.AlignedAdjustment}^{[26]} - 1, 0)$	all = si.ADDR	INCR <sup>(1)</sup>
cb.DW <sup>[3]</sup> > mi.DW <sup>[4]</sup>	if ( $2^{**}\text{cb.SIZE}^{[10]} \leq \text{mi.Bytes}^{[7]}$ )	1	No change	No change	FIXED
	else	cb.LEN+1	$\text{all} = \max(\text{cb.conv\_ratio}^{[23]} - \text{mi.AlignedAdjustment}^{[27]} - 1, 0)$	all = cb.ADDR	INCR <sup>(1)</sup>
si.DW <sup>[2]</sup> < cb.DW <sup>[3]</sup>	always	1	No change	No change	FIXED
cb.DW <sup>[3]</sup> < mi.DW <sup>[4]</sup>	always	1	No change	No change	FIXED

1. When width conversion results in a change in transaction length, the output SIZE is always the same as the output DATA\_WIDTH.

## Clock Conversion

Clock conversion comprises the following:

- A clock-rate reduction module performs integer (N:1) division of the clock rate from its input (SI) side to its output (MI) side.
- A clock-rate acceleration module performs integer (1:N) multiplication of clock rate from its input (SI) to output (MI) side.
- An asynchronous clock conversion module performs either reduction or acceleration of clock-rates by passing the channel signals through an asynchronous FIFO.

For both the reduction and the acceleration modules, the sample cycle for the faster clock domain is determined automatically. Each module is applicable to all five AXI channels.

The MI and SI each have a vector of clock inputs in which each bit synchronizes all the signals of the corresponding interface slot. The AXI Interconnect core has its own native clock input. The AXI Interconnect adapts the clock rate of each MI and SI slot automatically to the native clock rate of the AXI Interconnect.

Typically, the native clock input of the AXI Interconnect is tied to the same clock source as used by the highest frequency SI or MI slot in the system design, such as the MI slot connecting to the main memory controller.

## Pipelining

Under some circumstances, AXI Interconnect throughput is improved by buffering data bursts. This is commonly the case when the data rate at a SI or MI slot differs from the native data rate of the AXI Interconnect due to data width or clock rate conversion. To accommodate the various rate change combinations, data burst buffers can be inserted optionally at the various locations.

Additionally, a two-deep register slice (skid buffer) can be inserted optionally on each of the five AXI channels at each SI or MI slot to help improve system timing closure.

## Peripheral Register Slices

At the outer-most periphery of both the SI and MI, each channel of each interface slot can be optionally buffered by a register slice. These are provided mainly to improve system timing at the expense of one latency cycle.

Peripheral register slices are always synchronized to the SI or MI slot clock.

## Data Path FIFOs

To accommodate the data flow rate change combinations between the SI and MI sides of the AXI Interconnect, data path buffers can be inserted optionally at the following four locations:

- The SI-side Write data FIFO is located before the Write data router of each SI slot.
- The MI-side Write data FIFO is located after the Write data multiplexer of each MI slot.
- The MI-side Read data FIFO is located before (on the MI side of) the Read data router of each MI slot.
- The SI-side Read data FIFO is located after (on the SI side of) the Read data multiplexers of each SI slot, including the MUX between multiple ID-thread pathways for multi-threaded SI slots.

Data FIFOs are synchronized to the AXI Interconnect native clock. The width of each data FIFO matches the AXI Interconnect native data width.

## Use of ID Signals

The ID signals that propagate from master-to-slave devices (*AWID* and *ARID*) and back again (*BID* and *RID*) identify the original source of each transaction, and therefore, how slave device responses are to be routed back to the originating master devices across the interconnect topology of the system.

End-point master device 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 was comprised of multiple master devices internally. The “thread 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. Transactions belonging to the same thread must be returned in order, whereas transactions among different threads can be returned out-of-order. Master devices with a thread depth of one have no ID signals on their interface.

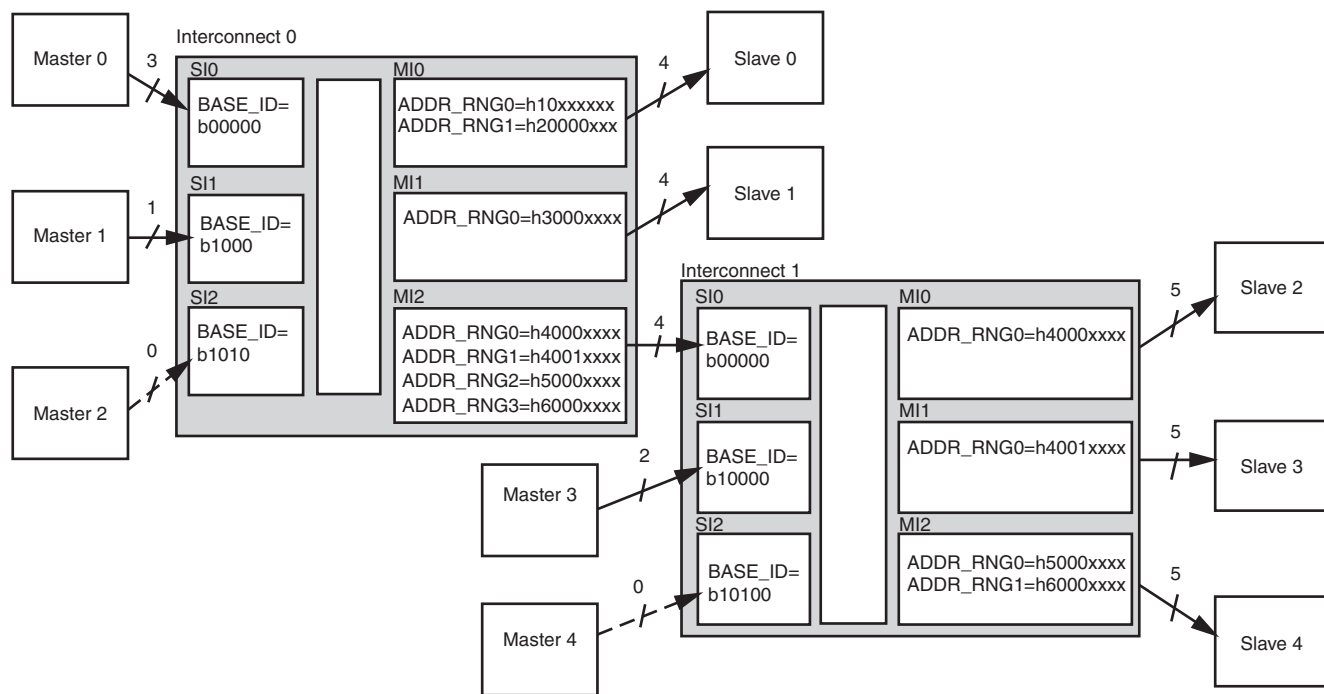
ID values among all the master devices must be made unique before propagating to any slave device. The AXI Interconnect prefixes a constant unique “master ID” value to the *AWID* and *ARID* signals sampled at each SI slot (if any).

A *BASE\_ID* parameter associated with each SI slot allows the AXI Interconnect to assign master IDs at compile time. End-point master devices are not required to drive their assigned master ID on their own ID outputs, and therefore do not need to be aware of their own assigned master ID values.

When a SI slot is connected hierarchically to another AXI Interconnect, all ID signals produced by the upstream AXI Interconnect are treated as though they are the thread ID bits of a connected master device. As with other master devices, the downstream AXI Interconnect prefixes the ID signals sampled from a hierarchical SI slot with a unique master ID, causing the ID width to grow as it propagates forward across a hierarchical AXI Interconnect topology. All responses matching that master ID are routed back to the upstream AXI Interconnect.

Figure 9, page 20 shows an example of connecting two AXI Interconnects hierarchically, as follows:

- MI slot #2 of AXI Interconnect 0 connects to SI slot 0 of AXI Interconnect 1. The end-point slave devices #2-4 have address ranges as defined for MI slot #s 0-2 of AXI Interconnect 1. Notice how these same ranges are represented as a series of address regions for MI slot 2 of AXI Interconnect 0.
- The arrows represent the ID signals propagated from each master device. AXI Interconnect 0 produces 4 bits of ID output, which is the smallest width possible to keep the master IDs unique. For example, when Master 0 issues a transaction, the output ID would be its master ID (1'b0) followed by 3 bits of ID sampled from the master device.
- All transactions from Master 2 would have ID value 4'b1010 (no variable thread bits from the master device).
- When a transaction from Master 0-2 targets Slaves #s 2-4, the 4-bit ID value produced by AXI Interconnect 0 is prefixed with an additional 1'b0 by AXI Interconnect 1, which has an output ID width of 5.



X12068

Figure 9: Hierarchical AXI Interconnects

## Multiple Address Range Support

The AXI Interconnect 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 only involves 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, and 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 Interconnect includes the multiple ranges that determine selection of each MI slot. Differentiation between the multiple address ranges is typically required by the functionality of the connected slave device also. That would typically mean some of the decode logic implemented by the AXI Interconnect is replicated in the slave device. The AXI4 protocol defines the new `REGION` signal that can be used to encode the results of which address range is being decoded by the AXI Interconnect. The AXI Interconnect generates these `REGION` outputs for use by slave devices with multiple address decode ranges so that the range decode logic does not need to be replicated in the slave device.

Whenever a transaction address received on the SI does not match any of the ranges being decoded by the AXI Interconnect, the transaction is trapped and handled by a decode error module within the AXI Interconnect. An exception is if the AXI Interconnect has only one MI slot and only one address range. Then, the `C_RANGE_CHECK` parameter determines whether address decoding and associated decode error traps are implemented or whether all transactions are propagated to the MI slot.

## Cyclic Dependency Avoidance

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

### How Deadlock Occurs

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

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 it were different. However, the AXI Interconnect 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 Interconnect 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 Interconnect to avoid deadlock is a “Single Slave per ID” method that does not impact the performance of the transactions of most critical concern, which are the pipelining of multiple reads and writes, by multiple master devices, to a performance-critical slave devices, such as a memory controller.

The “Single Slave per ID” method imposes the restriction that each ID thread issued by each master device can have outstanding transactions (of each type) to one slave device at a time only. However, slaves devices are still permitted to queue multiple outstanding transactions from multiple master devices.

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

The “Single Slave per ID” restriction applies to all transaction threads whenever the AXI Interconnect is configured as anything more complex than a 1-to-1 pass-through. In addition to preventing deadlock, this restriction also guarantees in-order completion of all Write transactions at the SIs, even if different slave devices are targeted by a transaction thread in successive transactions.

For example, a master device writes to a DMA descriptor in memory, then writes to a control register in a DMA engine that subsequently reads that descriptor. Because the AXI Interconnect does not allow the second Write to propagate to the DMA slave device until the first Write completes (Write completion received from the memory controller), there is no risk that the DMA could read 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 thread. Therefore, under those conditions, master devices do not need to condition subsequent Write transactions on receiving Write responses for prior transactions.

**Note:** AXI protocol provides no means to assure 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 Interconnect are:

- 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. The AXI Interconnect returns a `DECERR` and the transaction is not propagated to any MI slot. However, address decode errors are not trapped when the AXI Interconnect is configured to have only one MI slot and one address range, and the `C_RANGE_CHECK` parameter is set to 0, in which case the base and high addresses associated with the MI slot are ignored (transactions for all addresses are then propagated to the MI).
- AXI4-Lite burst length violation: Transaction length >1 data beat when targeting an AXI4-Lite slave device. The AXI Interconnect returns a `DECERR` and the transaction is not propagated to the MI slot.
- AXI4-Lite data size violation: Transaction data transfer size wider than 4 bytes when targeting an AXI4-Lite slave device. The AXI Interconnect returns a `DECERR` and the transaction is not propagated to the MI slot.
- TrustZone access violation: The AXI Interconnect returns a `DECERR` and the transaction is not propagated to the MI slot. TrustZone transaction blocking and error detection logic is implemented whenever the `C_M_AXI_SECURE` configuration parameter is set, even when the AXI Interconnect has only one SI and one MI slot.
- Narrow burst violation: Transaction length > 1 data beat and data transfer size less than the SI slot data-width for an SI slot that has `C_S_AXI_SUPPORTS_NARROW_BURST = 0`. The AXI Interconnect returns a `DECERR` and the transaction is not propagated to the MI slot.
- For unrecognized response ID errors, no `READY` response is issued from the AXI Interconnect to the slave device, and the entire response (Write response or Read data burst) is permanently blocked by the AXI Interconnect. This can cause the problematic slave device and any master device expecting to receive the response to hang.
- The AXI Interconnect does not trap AXI4 protocol violations, which are the responsibility of the endpoint IP. In addition, the AXI Interconnect neither supports nor traps Write data interleaving (all Write data is routed according to write transaction order; `WID` is not sampled at the SI).

Other static erroneous conditions are trapped by Xilinx software design tools at configuration time. Therefore, no error detection logic are provided by the AXI Interconnect for the following configuration errors:

- Non-integer clock ratios when not configured for asynchronous clocking
- Parameter value range violations
- Address or ID range overlap, non-binary size or base value misalignment.

## AXI Protocol Converters

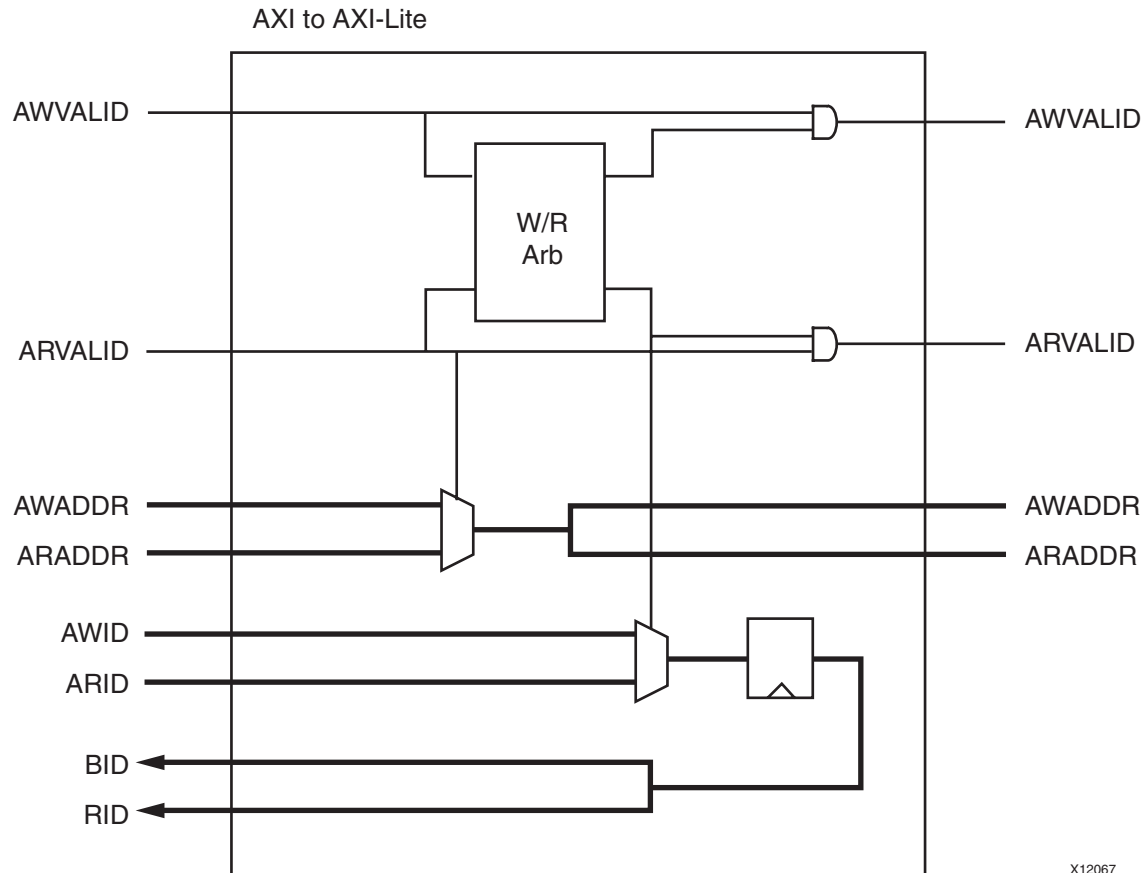
The following subsections describe AXI protocol converters:

- [AXI4-Lite Slave Conversion](#)
- [AXI3 Slave Converter](#)

### AXI4-Lite Slave Conversion

Each MI slot connected to an AXI4-Lite slave device routes through an AXI4-Lite conversion block. The conversion block single-threads all transactions, including single-threaded Round-Robin arbitration between Write and Read transactions. In most cases, Write and Read addresses are multiplexed onto a single bus, which is then duplicated onto the `AWADDR` and `ARADDR` signals of the MI slot. In most cases these duplicated signals are trimmed during back-end design implementation, so that the resources used by AXI4-Lite slaves are roughly the same as if there were only one address bus.

The transaction ID (`AWID` or `ARID`) is stripped and stored in the conversion block, and retrieved during response transfers as `BID` or `RID`. The following figure illustrates the AXI4-Lite conversion logic.



X12067

Figure 10: AXI4-Lite Conversion Logic

## AXI3 Slave Converter

A set of AXI3 slave conversion modules is instantiated at each MI slot connected to an AXI3 slave device, if the MI slot is accessible by one or more AXI4 SI slots.

This module receives an AW or AR transfer (command) on its slave interface and produces one or more commands on its MI, similar to the address channel down-sizer module. The data transfer `SIZE` is never modified by the AXI3 converter. Any time a burst longer than 16 data beats is received, the command is split into a number of shorter burst transactions.

## I/O Signals

This section lists the AXI Interconnect signals.

In the following tables, the “Default” column shows if the input signal is required (REQ) or, if not, its default value if left unconnected. Signal connections are required only for the SI and MI slots that are used. Values in the Default column also provides the protocol mode of the slot: in the context of this table, “Full” indicates AXI4 and AXI3, or “Lite” for AXI4-Lite. Input signals that are not sampled (do not care) for AXI4-Lite are indicated by “d/c”.

### Slave Interface I/O Signals

In the Width column “N” refers to the total number of SI slots, which is the number of master devices connected to the AXI Interconnect.

**Table 3: Slave I/O Signals**

Signal Name	Direction	Default	Width	Description (Range)
S_AXI_ARESET_OUT_N	Output		N*1	Reset output (active-low) re-synchronized to each slot's clock. (This is not a signal defined by AXI protocol)
S_AXI_ACLK	Input	REQ	N*1	Clock
S_AXI_AWID	Input	Full: 0 Lite: d/c	N*C_AXI_ID_WIDTH	Write Address Channel Transaction ID
S_AXI_AWADDR	Input	REQ	N*C_AXI_ADDR_WIDTH	Write Address Channel Address
S_AXI_AWLEN	Input	Full: 0 Lite: d/c	N*8	Write Address Channel Burst Length (0-255)
S_AXI_AWSIZE	Input	Full: REQ* Lite: d/c	N*3	Write Address Channel Transfer Size code (0-7)
S_AXI_AWBURST	Input	Full: REQ* Lite: d/c	N*2	Write Address Channel Burst Type code (0-2).
S_AXI_AWLOCK	Input	Full: 0 Lite: d/c	N*2	Write Address Channel Atomic Access Type (0, 1)
S_AXI_AWCACHE	Input	Full: 0*** Lite: d/c	N*4	Write Address Channel Cache Characteristics
S_AXI_AWPROT	Input	0b000**	N*3	Write Address Channel Protection Bits

\* 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.

\*\* 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.

\*\*\* Xilinx recommends that master devices drive their `AW/RCACHE` outputs to 0b0011 to allow the AXI Interconnect to pack data while performing width conversion and to allow store-and-forward in data-path FIFOs.



**Table 3: Slave I/O Signals (Cont'd)**

Signal Name	Direction	Default	Width	Description (Range)
S_AXI_AWQOS	Input	Full: 0 Lite: d/c	N*4	Write Address Channel Quality of Service
S_AXI_AWUSER	Input	Full: 0 Lite: d/c	N*C_AXI_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_WDATA	Input	REQ	N*C_S_AXI_DATA_WIDTH	Write Data Channel Data
S_AXI_WSTRB	Input	all-ones	N*C_S_AXI_DATA_WIDTH/8	Write Data Channel Byte Strobes
S_AXI_WLAST	Input	Full: 0 Lite: d/c	N*1	Write Data Channel Last Data Beat
S_AXI_WUSER	Input	Full: 0 Lite: d/c	N*C_AXI_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*C_AXI_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*C_AXI_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	Full: 0 Lite: d/c	N*C_AXI_ID_WIDTH	Read Address Channel Transaction ID.
S_AXI_ARADDR	Input	REQ	N*C_AXI_ADDR_WIDTH	Read Address Channel Address.
S_AXI_ARLEN	Input	Full: 0 Lite: d/c	N*8	Read Address Channel Burst Length code(0-255).
S_AXI_ARSIZE	Input	Full: REQ* Lite: d/c	N*3	Read Address Channel Transfer Size code (0-7).
S_AXI_ARBURST	Input	Full: REQ* Lite: d/c	N*2	Read Address Channel Burst Type (0-2).
S_AXI_ARLOCK	Input	Full: 0 Lite: d/c	N*2	Read Address Channel Atomic Access Type (0, 1).
S_AXI_ARCACHE	Input	Full: 0*** Lite: d/c	N*4	Read Address Channel Cache Characteristics.
S_AXI_ARPROT	Input	0b000**	N*3	Read Address Channel Protection Bits.
S_AXI_ARQOS	Input	Full: 0 Lite: d/c	N*4	Read Address Channel Quality of Service.
S_AXI_ARUSER	Input	Full: 0 Lite: d/c	N*C_AXI_ARUSER_WIDTH	User-defined AR Channel signals.
S_AXI_ARVALID	Input	REQ	N*1	Read Address Channel Valid.

\* 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.

\*\* 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.

\*\*\* Xilinx recommends that master devices drive their *AW/RCACHE* outputs to 0b0011 to allow the AXI Interconnect to pack data while performing width conversion and to allow store-and-forward in data-path FIFOs.

Table 3: Slave I/O Signals (Cont'd)

Signal Name	Direction	Default	Width	Description (Range)
S_AXI_ARREADY	Output		N*1	Read Address Channel Ready.
S_AXI_RID	Output		N*C_AXI_ID_WIDTH	Read Data Channel Transaction ID.
S_AXI_RDATA	Output		N*C_S_AXI_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*C_AXI_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.

\* Xilinx recommends that AXI4 master devices drive their *AW/R*SIZE and *AW/R*BURST outputs. Typically, a master device drives an *AW/R*SIZE value that corresponds to its interface data width, unless application requirements dictate otherwise. Typically a master device drives its *AW/R*BURST output to 0b01, which indicates an incremental (INCR) burst.

\*\* AXI protocol requires master devices to drive their *AW/R*PROT output. If the *AW/R*PROT signals are left undriven, it would default to all-zeros and the transaction would be interpreted as secure.

\*\*\* Xilinx recommends that master devices drive their *AW/R*CACHE outputs to 0b0011 to allow the AXI Interconnect to pack data while performing width conversion and to allow store-and-forward in data-path FIFOs.

## Master Interface Signals

In the Width column of the following table, “M” refers to the total number of Master Interface (MI) slots, which is the number of slave devices connected to the AXI Interconnect.

Table 4: Master I/O Signals

Signal Name	Direction	Default	Width	Description (Range)
M_AXI_ARESET_OUT_N	Output		M*1	Reset output (active-low) re-synchronized to each slot's clock. (This is not a signal defined by AXI protocol.)
M_AXI_ACLK	Input	REQ	M*1	Clock.
M_AXI_AWID	Output		M*C_AXI_ID_WIDTH	Write Address Channel Transaction ID.
M_AXI_AWADDR	Output		M*C_AXI_ADDR_WIDTH	Write Address Channel Address.
M_AXI_AWLEN	Output		M*8	Write Address Channel Burst Length code. (0-255).
M_AXI_AWSIZE	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		M*2	Write Address Channel Atomic Access Type (0, 1).
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	Write Address Channel address region index.
M_AXI_AWQOS	Output		M*4	Write Address Channel Quality of Service.
M_AXI_AWUSER	Output		M*C_AXI_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*C_AXI_ID_WIDTH	Write Data Channel Transaction ID for AXI3 slaves (copied from S_AXI_AWID).

**Table 4: Master I/O Signals (Cont'd)**

Signal Name	Direction	Default	Width	Description (Range)
M_AXI_WDATA	Output		M*C_M_AXI_DATA_WIDTH	Write Data Channel Data.
M_AXI_WSTRB	Output		M*C_M_AXI_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*C_AXI_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	Full: REQ Lite: d/c	M*C_AXI_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	Full: 0 Lite: d/c	M*C_AXI_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*C_AXI_ID_WIDTH	Read Address Channel Transaction ID.
M_AXI_ARADDR	Output		M*C_AXI_ADDR_WIDTH	Read Address Channel Address.
M_AXI_ARLEN	Output		M*8	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		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	Read Address Channel address region index.
M_AXI_ARQOS	Output		M*4	Read Address Channel Quality of Service.
M_AXI_ARUSER	Output		M*C_AXI_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	Full: REQ Lite: d/c	M*C_AXI_ID_WIDTH	Read Data Channel Transaction ID.
M_AXI_RDATA	Input	REQ	M*C_M_AXI_DATA_WIDTH	Read Data Channel Data.
M_AXI_RRESP	Input	0b00	M*2	Read Data Channel Response Code (0-3).
M_AXI_RLAST	Input	Full: REQ Lite: d/c	M*1	Read Data Channel Last Data Beat.
M_AXI_RUSER	Input	Full: 0 Lite: d/c	M*C_AXI_RUSER_WIDTH	User-defined R Channel signals.
M_AXI_RVALID	Input	REQ	M*1	Read Data Channel Valid.
M_AXI_RREADY	Output		M*1	Read Data Channel Ready.

## Global Ports

Table 5: Global Port Signals

Port Signal Name	Direction	Default	Width	Description (Range)
INTERCONNECT_ACLK	Input	REQ	1	Interconnect native clock input.
INTERCONNECT_ARESETN	Input	REQ	1	Global Reset (Active-Low).

## Design Parameters

The following subsections list the design parameters and conventions used to describe those parameters.

### Conventions in the Parameter Summary Tables

The following conventions are used in the Core, Inherent Slave and Inherent Master Parameters table (Table 6, Table 7, Table 8):

In the Format (Range) column:

- “N” represents the value of C\_NUM\_SLAVE\_SLOTS.
- “M” represents the value of C\_NUM\_MASTER\_SLOTS.
- Braces { } denote a replication count for the value that follows.
- “Bit1” represents a 1-bit value, “Bit32” represents a 32-bit value, and “Bit64” a 64-bit value.

For example, “[N] Bit32” is used to indicate a parameter consisting of the concatenation of 32-bit values for each SI slot.

- Core parameters influence HDL compilation, unless indicated otherwise by footnote N.

### Interconnect Core Parameters

Table 6: Interconnect Core Parameters

Parameter Name	Default Value	Format/Range	Description
C_NUM_SLAVE_SLOTS (T)	1	Integer (1-16)	Number of SI slots.
C_NUM_MASTER_SLOTS (T)	1	Integer (1-16)	Number of MI slots.
C_FAMILY	REQ	string (virtex6, spartan6)	FPGA Family.
C_AXI_ID_WIDTH (T)	1	Integer (1-16)	Width of all ID signals propagated by the AXI Interconnect.
C_AXI_ADDR_WIDTH (C.)	32	Integer (32)	Width of all ADDR signals for all SI slots and MI slots.
C_S_AXI_IS_INTERCONNECT (T)	{N}0b0	{N} Bit1	Used to determine whether CDAM logic is implemented: 0 = connects to an end-point master device, 1 = connects to another AXI Interconnect.

#### Notes:

U = User specified.

T = Tool generated (EDK originates the information and sets the value.)

C = Constant

O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)

N = Not used by the HDL of the core.

**Table 6: Interconnect Core Parameters (Cont'd)**

Parameter Name	Default Value	Format/Range	Description
C_INTERCONNECT_DATA_WIDTH (U)	32	Integer (32, 64, 128, 256)	Data width of the internal interconnect Write and Read data paths.
C_INTERCONNECT_ACLK_RATIO (O)	1	Integer (1-16)	Clock frequency ratio of the internal AXI Interconnect with regard to all SI and MI slots. (Slowest clock input should have ratio =1).
C_AXI_SUPPORTS_USER_SIGNALS (U)	0	Integer	Indicates whether or not to propagate the USER signals (all 5 channels) across the AXI Interconnect. 0 = not propagate 1 = propagate
C_AXI_AWUSER_WIDTH (U)	1	Integer (1-256)	Width of AWUSER signals for all AXI4 SI slots and MI slots.
C_AXI_ARUSER_WIDTH (U)	1	Integer (1-256)	Width of ARUSER signals for all AXI4 SI slots and MI slots.
C_AXI_WUSER_WIDTH (U)	1	Integer (1-256)	Width of WUSER signals for all AXI4 SI slots and MI slots.
C_AXI_RUSER_WIDTH (U)	1	Integer (1-256)	Width of RUSER signals for all AXI4 SI slots and MI slots.
C_AXI_BUSER_WIDTH (U)	1	Integer (1-256)	Width of BUSER signals for all AXI4 SI slots and MI slots.
C_AXI_CONNECTIVITY (T)	all-ones	{M} Bit32 (N) Bit1	Sparse crossbar connectivity from each SI slot (N) to each MI slot (M). 0 = no pathway required 1 = pathway required
C_INTERCONNECT_CONNECTIVITY_MODE (C, N)	1	Integer (1)	Defines the connectivity mode: 1 = Shared Address Multiple Data (SAMD)
C_RANGE_CHECK (U, I)	0	Integer (0,1)	Determines whether address decode is performed when only 1 MI slot with only 1 address range is present 0 = Pass all transactions (no DECERR) when only 1 MI slot/range. 1 = Always check address ranges (DECERR if no match to MI slot).

**Notes:**

U = User specified.

T = Tool generated (EDK originates the information and sets the value.)

C = Constant

O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)

N = Not used by the HDL of the core.

## Slave Interface Parameters

The following parameters relate to the slave interface of the AXI Interconnect.

**Table 7: Slave Parameters**

Parameter Name	Default Value	Format/Range	Description
C_S_AXI_PROTOCOL <sup>(M)</sup>	{N} 0x00000000	{N} Bit32	AXI protocol of connected master device: 0 = SI slot is AXI4 1 = SI slot is AXI3 2 = SI slot is AXI4-Lite
C_S_AXI_DATA_WIDTH <sup>(M)</sup>	{N}0x00000020	{N} Bit32 (0x00000020, 0x00000040, 0x00000080, 0x00000100)	Effective width of S_AXI_WDATA and S_AXI_RDATA for each SI slot. (Must be 0x20 for AXI4-Lite SI slots)
C_S_AXI_BASE_ID <sup>(I,O)</sup>	{N} 0x00000000	{N}Bit32 (0-0xFFFF)	Base ID of each SI slot (N-1:0).
C_S_AXI_THREAD_ID_WIDTH <sup>(M)</sup>	{N} 0x00000000	{N}Bit32 (0-0x10)	Number of variable low-order ID bits of each SI slot (N-1:0). Each value must be ≤ C_AXI_ID_WIDTH.
C_S_AXI_ACLK_RATIO <sup>(I,O)</sup>	{N}0x00000001	{N} Bit32 (0x1-0x10)	Clock frequency ratio of each SI slot with regard to internal interconnect, if synchronous. (Slowest clock input should have ratio =1.)
C_S_AXI_IS_ACLK_ASYNC <sup>(I,O)</sup>	{N}0b0	{N} Bit1	Indicates if the SI slot clock is synchronous or asynchronous to AXI Interconnect native clock. 0 = SI slot clock is synchronous to AXI Interconnect native clock 1 = SI slot clock is asynchronous to interconnect native clock
C_S_AXI_ARB_PRIORITY <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32 (0x00000000 -0x0000000f)	Arbitration priority among each SI slot. Higher values indicate higher priority. All slots with value 0 participate in round-robin arbitration.
C_S_AXI_WRITE_ACCEPTANCE <sup>(I,U)</sup>	{M}0x00000001	{M} Bit32 (2**0 – 2**5)	Number of data-active Write transactions that each AXI SI slot can generate
C_S_AXI_READ_ACCEPTANCE <sup>(I,U)</sup>	{M}0x00000001	{M} Bit32 (2**0 – 2**5)	Number of active Read transactions that each AXI SI slot can generate
C_S_AXI_SUPPORTS_WRITE <sup>(M)</sup>	{N}0b1	{N}Bit1	Indicates whether each SI slot uses Write-related channels. 0 = Read-only 1 = Use AW, W, and B channels
C_S_AXI_SUPPORTS_READ <sup>(M)</sup>	{N}0b1	{N}Bit1	Indicates whether each SI slot uses Read-related channels. 0 = Write-only 1= Use AR and R channels

### Notes:

I = Inherent parameter available on all connected master devices.  
M = Value is copied from a parameter that exists on the connected master device.  
T = Tool generated (EDK originates the information and sets the value.)  
C = Constant  
O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)  
N = Not used by the HDL of the core.  
U = User specified.

Table 7: Slave Parameters (Cont'd)

Parameter Name	Default Value	Format/Range	Description
C_S_AXI_SUPPORTS_NARROW_BURST <sup>(M)</sup>	{N}0b1	{N} Bit1	Indicates whether the connected master device can produce narrow bursts. 0 = all bursts are same size as data-width and A*CACHE[1]=1 always (does not apply to single-beat transfers). 1 = may produce narrow bursts or de-assert A*CACHE[1].
C_S_AXI_WRITE_FIFO_DEPTH <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32 (0x00000000, 0x00000020, 0x00000200)	Depth of SI-side Write data FIFO (before W channel arbitration) for each SI slot.
C_S_AXI_READ_FIFO_DEPTH <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32 (0x00000000, 0x00000020, 0x00000200)	Depth of SI-side Read data FIFO (after R channel routing) for each SI slot.
C_S_AXI_AW_REGISTER <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32	Insert register slice on AW channel at each SI slot interface. 0 = Bypass 1 = Fully_registered
C_S_AXI_AR_REGISTER <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32	Insert register slice on AR channel at each SI slot interface. 0 = Bypass 1 = Fully_registered
C_S_AXI_W_REGISTER <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32	Insert register slice on W channel at each SI slot interface. 0 = Bypass 1 = Fully_registered
C_S_AXI_R_REGISTER <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32	Insert register slice on R channel at each SI slot interface. 0 = Bypass 1 = Fully_registered
C_S_AXI_B_REGISTER <sup>(I,U)</sup>	{N}0x00000000	{N} Bit32	Insert register slice on B channel at each SI slot interface. 0 = Bypass 1 = Fully_registered

**Notes:**

I = Inherent parameter available on all connected master devices.

M = Value is copied from a parameter that exists on the connected master device.

T = T = Tool generated (EDK originates the information and sets the value.)

C = Constant

O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)

N = Not used by the HDL of the core.

U = User specified.

## Master Interface Parameters

The following parameters relate to the master interface of the AXI Interconnect and to the connected slave device.

**Table 8: Master Interface-Related Parameters**

Parameter Name	Default Value	Format/Range	Description
C_M_AXI_PROTOCOL (S)	{M} 0x00000000	{M} Bit32	AXI protocol of connected slave device: 0 = MI slot is AXI4 1 = MI slot is AXI3 2 = MI slot is AXI4-Lite
C_M_AXI_DATA_WIDTH (S)	{M} 0x00000020	{M} Bit32 (0x00000020, 0x00000040, 0x00000080, 0x00000100)	Effective width of M_AXI_WDATA and M_AXI_RDATA for each MI slot. (Must be 0x20 for AXI4-Lite MI slots.)
C_M_AXI_BASE_ADDR (I,U)	{M} (16)0xffffffff	{M} (16) Bit64	Base address of each range (15:0) of each MI slot (M-1:0). For unused ranges, set base address to 0xffffffff.
C_M_AXI_HIGH_ADDR (I,U)	{M} (16) 0x00000000_00000000	{M} (16) Bit64	High address of each range (15:0) of each MI slot (M-1:0). For unused ranges, set high address to 0x00000000_00000000.
C_M_AXI_ACLK_RATIO (I,O)	{M} 0x00000001	{M} Bit32 (0x1-0x10)	Clock frequency ratio of each MI slot with regard to internal AXI Interconnect, if synchronous. (Slowest clock input should have ratio =1)
C_M_AXI_IS_ACLK_ASYNC (I,O)	{M} 0b0	{M} Bit1	Indicates if the MI slot clock is synchronous or asynchronous to AXI Interconnect native clock. 0 = MI slot clock is synchronous 1 = MI slot clock is asynchronous
C_M_AXI_SUPPORTS_WRITE (S)	{M} 0b1	{M} Bit1	Indicates whether each MI slot uses Write-related channels. 0 = Read-only 1 = Use AW, W, and B channels
C_M_AXI_SUPPORTS_READ (S)	{M} 0b1	{M} Bit1	Indicates whether each MI slot Read-related channels. 0 = Write-only 1 = AR and R channels
C_M_AXI_WRITE_ISSUING (I,U)	{M} 0x00000001	{M} Bit32 (2**0 – 2**5)	Number of data-active Write transactions that each AXI4 MI slot can generate
C_M_AXI_READ_ISSUING (I,U)	{M} 0x00000001	{M} Bit32 (2**0 – 2**5)	Number of active Read transactions that each AXI4 MI slot can generate
C_M_AXI_SECURE (I,U)	{M} 0b0	{M} Bit1	Indicates whether each MI slot connects to a secure slave device (allows TrustZone secure access). 0 = non-secure slave device 1 = secure slave device

### Notes:

- I = Inherent parameter available on all connected master devices.
- S = Value is copied from a parameter that exists on the connected slave device.
- U = User specified.
- T = Tool generated (EDK originates the information and sets the value.)
- C = Constant
- O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)



Table 8: Master Interface-Related Parameters (Cont'd)

Parameter Name	Default Value	Format/Range	Description
C_M_AXI_SUPPORTS_NARROW_BURST (S, O)	{M}0b1	{M} Bit1	Indicates whether the connected slave device is configured to support bursts where the transfer size is narrower. 0 = connected slave device does not tolerate bursts that are a different SIZE than the MI-slot data-width (does not apply to single-beat transfers). 1 = connected slave device supports narrow bursts.
C_M_AXI_WRITE_FIFO_DEPTH (I, U)	{M}0x00000000	{M} Bit32 (0x00000000, 0x00000020, 0x00000200)	Depth of MI-side Write data FIFO (after W channel routing) for each MI slot 0x0 = no FIFO 0x20 = 32-deep LUT-RAM based FIFO 0x200 = 512-deep BRAM based FIFO
C_M_AXI_READ_FIFO_DEPTH (I, U)	{M}0x00000000	{M} Bit32 (0x00000000, 0x00000020, 0x00000200)	Depth of MI-side Read data FIFO (before R channel arbitration) for each MI slot 0x0 = no FIFO 0x20 = 32-deep LUT-RAM based FIFO 0x200 = 512-deep BRAM based FIFO
C_M_AXI_AW_REGISTER (I, U)	{M}0x00000000	{M} Bit32	Insert register slice on AW channel at each MI slot interface. 0 = Bypass 1 = Fully_registered
C_M_AXI_AR_REGISTER (I, U)	{M}0x00000000	{M} Bit32	Insert register slice on AR channel at each MI slot interface. 0=Bypass, 1=Fully_registered
C_M_AXI_W_REGISTER (I, U)	{M}0x00000000	{M} Bit32	Insert register slice on W channel at each MI slot interface: 0 = Bypass 1 = Fully_registered.
C_M_AXI_R_REGISTER (I, U)	{M}0x00000000	{M} Bit32	Insert register slice on R channel at each MI slot interface: 0 = Bypass 1 = Fully_registered
C_M_AXI_B_REGISTER (I, U)	{M}0x00000000	{M} Bit32	Insert register slice on B channel at each MI slot interface: 0 = Bypass 1 = Fully_registered

**Notes:**

I = Inherent parameter available on all connected master devices.  
 S = Value is copied from a parameter that exists on the connected slave device.  
 U = User specified.  
 T = Tool generated (EDK originates the information and sets the value.)  
 C = Constant  
 O = Tool-generated with user override (EDK originates the information and sets the value; user can override.)

## AXI Interconnect Parameter Usage

The AXI Interconnect provides register slices for each Read and Write channel per master and slave.

### Interface Protocol

Interconnect	Connected Master	Connected Slave
C_S_AXI_PROTOCOL	C_busif_PROTOCOL	
C_M_AXI_PROTOCOL		C_busif_PROTOCOL

The \*\_`PROTOCOL` parameter identifies the interface sub-protocol (AXI4, AXI3, or AXI4-Lite) of the AMBA AXI specification. Typically, the parameter is specified as a constant in the MPD of the connected master and slave IP. However, some IP supports configurable protocol (typically AXI4 vs. AXI4-Lite), which is then user-selectable.

The AXI Interconnect uses the `Protocol` parameter to:

- Insert optional protocol conversion modules
- Impose error detection logic
- In the case of AXI4-Lite, conserve logic resources associated with unused AXI4 interface features

The tools copy these values from the connected masters and slaves onto the AXI Interconnect.

### Data Width

Interconnect	Connected Master	Connected Slave
C_S_AXI_DATA_WIDTH	C_busif_DATA_WIDTH	
C_M_AXI_DATA_WIDTH		C_busif_DATA_WIDTH
C_INTERCONNECT_DATA_WIDTH		

The `C_S_AXI_DATA_WIDTH` and `C_M_AXI_DATA_WIDTH` parameters indicate the widths of the WDATA and RDATA signals on the connected master and slave devices, respectively. The tools copy these values from the connected masters and slaves onto the AXI Interconnect.

The `C_INTERCONNECT_DATA_WIDTH` parameter specifies the native data width of the internal crossbar and can be user-configured independently. Typically, `C_INTERCONNECT_DATA_WIDTH` is matched to the smaller of the widest SI slot or the widest MI slot in the system design.

When the value of:

- `C_S_AXI_DATA_WIDTH` for a SI slot is less than `C_INTERCONNECT_DATA_WIDTH`, an upsizer module is inserted in the pathways from that SI slot in the SI hemisphere of the AXI Interconnect (between the SI and the crossbar).
- `C_S_AXI_DATA_WIDTH` is greater than `C_INTERCONNECT_DATA_WIDTH`, a downsizer module is inserted in the SI hemisphere.
- `C_M_AXI_DATA_WIDTH` differs from `C_INTERCONNECT_DATA_WIDTH`, the appropriate width converter is inserted in the MI hemisphere (between the crossbar and the MI).

Data width converters perform the packing and serialization of data required to adapt the differing data widths, and consequently affect the data bandwidth along the pathways between the SI, crossbar and MI.

Selecting a sufficiently high value of `C_INTERCONNECT_DATA_WIDTH` can avoid loss of data bandwidth.

For example, you could elect to set the `C_INTERCONNECT_DATA_WIDTH` 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 can perform data packing (Write transactions) or serialization (Read transactions) concurrently along multiple SI slot pathways in the SI hemisphere while the crossbar maintains at a data throughput rate higher than any one master device can sustain.

In contrast, selecting a lower value of `C_INTERCONNECT_DATA_WIDTH` can reduce logic resource utilization for less speed-critical designs.

### Clock Frequency

Interconnect	Connected Master	Connected Slave
<code>C_S_AXI_ACLK_RATIO</code>	<code>C_busif_ACLK_RATIO</code> (also <code>C_busif_ACLK_FREQ_HZ</code> )	
<code>C_M_AXI_ACLK_RATIO</code>		<code>C_busif_ACLK_RATIO</code> (also <code>C_busif_ACLK_FREQ_HZ</code> )
<code>C_S_AXI_IS_ACLK_ASYNC</code>	<code>C_busif_IS_ACLK_ASYNC</code>	
<code>C_M_AXI_IS_ACLK_ASYNC</code>		<code>C_busif_IS_ACLK_ASYNC</code>
<code>C_INTERCONNECT_ACLK_RATIO</code>		

The Xilinx Platform Studio (XPS) tools keep track of the various clock signals in an embedded system using the `CLK_FREQ_HZ` property (sometimes represented by a parameter `C_busif_ACLK_FREQ_HZ`) associated with the clock port (ACLK) which is associated with each bus interface.

The AXI Interconnect has a global clock port also, `INTERCONNECT_ACLK`, that synchronizes the crossbar and other internal modules. The tools survey the `CLK_FREQ_HZ` values among all the connected masters and slaves, compare them to the frequency of the `INTERCONNECT_ACLK` port, and determine the clock frequency relationships between each of the SI and MI slots and the crossbar.

The clock frequency relationships are determined in the tools as follows:

- When the relationship is an integer ratio (faster or slower), the tools set the corresponding `IS_ACLK_ASYNC` parameter to zero (synchronous); otherwise, the slot is tagged as asynchronous.
- Among all the synchronous SI and MI slots and the AXI Interconnect itself, the lowest clock frequency clock is assigned as `ACLK_RATIO = 1`. All higher frequency synchronous clocks are assigned a `ACLK_RATIO` value that reflects the multiple of the lowest frequency.
- When a SI slot is asynchronous (`C_S_AXI_IS_ACLK_ASYNC = 1`) or its clock ratio is different than that of the `INTERCONNECT_ACLK` (`C_S_AXI_ACLK_RATIO != C_INTERCONNECT_ACLK_RATIO`), then a clock converter module is inserted in the pathways from that SI slot in the SI hemisphere of the AXI Interconnect (between the SI and the crossbar).
- When a MI slot is asynchronous or has a clock ratio different than the AXI Interconnect, a clock converter module is inserted in the MI hemisphere (between the crossbar and the MI).

Clock converters always introduce additional latency. Passing through a clock converter in both the SI and MI hemispheres when traversing any pathway between a master and slave is wasteful; select clock frequencies to avoid passing through a clock converter in both hemispheres whenever possible.

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

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

## Address Ranges

Interconnect	Connected Master	Connected Slave
<code>C_M_AXI_BASE_ADDR</code>		<code>C_busif_BASEADDR</code> or <code>C_busif_RNGnn_BASEADDR</code>
<code>C_M_AXI_HIGH_ADDR</code>		<code>C_busif_HIGHADDR</code> or <code>C_busif_RNGnn_HIGHADDR</code>
<code>C_AXI_ADDR_WIDTH</code>	<code>C_busif_ADDR_WIDTH</code>	<code>C_busif_ADDR_WIDTH</code>

For all unused address ranges, the corresponding values in `C_M_AXI_BASE_ADDR` are set to all-ones (for as many low-order bits as specified by `C_AXI_ADDR_WIDTH`, or more), and the corresponding values in `C_M_AXI_HIGH_ADDR` are set to all-zeros.

For each used address range:

- The range size ( $\text{HIGH\_ADDR} - \text{BASE\_ADDR} + 1$ ) must be a minimum of 4k
- The range size must be a power of 2
- `BASE_ADDR` must be a multiple of the range size (aligned).
- No overlapping address ranges across the entire address decode table (all MI slots)

All address range restrictions are enforced by the tools used to configure the AXI Interconnect, and are therefore not enforced by Design Rule Checks (DRCs) provided by the IP itself.

When an MI slot connects to another “downstream” AXI Interconnect hierarchically, the address decoder in the upstream Interconnect uses multiple address ranges for that MI slot to represent the collection of address ranges for all the downstream slave devices that are accessible through the downstream Interconnect.

It is important to use separate address regions to represent different end-point slave devices, and that more than one slave device are never combined into the same address decoder region, even if they are mapped to adjacent or nearby regions of the system address map. This allows the Cyclic Dependency Avoidance Method (CDAM) to enforce the policy that each ID thread issued by each master device, can have outstanding Write or Read transactions to only one end-point slave device at a time.

Embedded hardware systems support 32-bit addresses only; therefore, the value of `C_AXI_ADDR_WIDTH` is the constant 32.

**Note:** The AXI Interconnect does not support address re-mapping. Therefore, if any multi-ported slaves (such as multi-ported memory controllers) are used in the system, for which multiple bus-interfaces share the same or overlapping address ranges, those multiple bus-interfaces must be connected to different instances of AXI Interconnect, and those Interconnect instances *must not* be cascaded to each other. The tools enforce the rule that multiple address ranges within the same Interconnect must not overlap.

## ID Ranges

Interconnect	Connected Master	Connected Slave
C_S_AXI_BASE_ID	C_INTERCONNECT_busif_BASE_ID	
C_S_AXI_THREAD_ID_WIDTH	C_busif_SUPPORTS_THREADS, C_busif_THREAD_ID_WIDTH	
C_AXI_ID_WIDTH		C_busif_ID_WIDTH

The ID values route response transfers back to the correct SI slot and to the connected master device. The C\_S\_AXI\_BASE\_ID parameter defines the base (lowest) ID value corresponding to the master device connected to each SI slot.

Master devices supporting a transaction reordering depth > 1 can issue multiple “threads” of transactions by varying a certain number of low-order bits of their ID outputs (log2 reordering depth).

The bit ordering activity is as follows:

- The C\_S\_AXI\_THREAD\_ID\_WIDTH parameter specifies the number of low-order ID bits produced by a master device.
- In the C\_S\_BASE\_ID parameter values, those same low-order bit positions must all be zero.
- The remaining high-order bits of C\_S\_BASE\_ID remain constant for all transactions propagated from each master device, and are considered to be the “Master ID.”

The EDK tools determine the required value of the AXI Interconnect C\_AXI\_ID\_WIDTH parameter of and assign unique MasterID values among all the SI slots based upon the values of the C\_S\_THREAD\_ID\_WIDTH parameters found on each SI slot.

The AXI Interconnect uses the C\_S\_BASE\_ID and C\_S\_THREAD\_ID\_WIDTH parameters to implement the decode logic for routing responses on the R and B channels back to the proper SI slot.

When an SI slot connects to an MI slot of another (up-stream) AXI Interconnect, the range of ID values that can be received on the ID signals of that slot consists of all the ID values on all the SI slots of the up-stream AXI Interconnect (those that have connection pathways to the connected MI slot). ID signals produced by the upstream AXI Interconnect are treated as thread ID bits of a connected master device.

As with other master devices, the downstream AXI Interconnect prefixes the ID signals sampled from a hierarchical SI slot with a unique master ID, causing the ID width to grow as it propagates forward across a hierarchical AXI Interconnect topology. All responses matching that master ID are routed back to the upstream Interconnect.

Similar to address ranges, each ID range comprises an aligned, binary-sized range, as all the low-order bits of the BASE\_ID parameter (as defined by THREAD\_ID\_WIDTH) must be zero. There must be no overlap among any ID ranges across the entire ID decode table (all SI slots). Unlike address ranges, ID ranges have no minimum size.

On the AXI Interconnect:

- When the `C_S_AXI_THREAD_ID_WIDTH` value for a SI slot is zero, the size of the corresponding ID range is a single ID value and no ID signals are sampled from the connected master. However, on the connected master device, the corresponding `C_busif_THREAD_ID_WIDTH` parameter is often used to determine the physical bit-width of the ID signals on the IP interface, and therefore must never be set to a value less than 1.
- The `C_busif_SUPPORTS_THREADS` parameter is provided to indicate when the effective thread width of the master is zero.
- When `C_busif_SUPPORTS_THREADS = 0` on a connected master, `C_S_AXI_THREAD_ID_WIDTH` is set to zero for the corresponding SI slot on the AXI Interconnect.

### Transaction Acceptance and Issuing Limits

Interconnect	Connected Master	Connected Slave
<code>C_S_AXI_WRITE_ACCEPTANCE</code>	<code>C_INTERCONNECT_busif_WRITE_ISSUING</code>	
<code>C_S_AXI_READ_ACCEPTANCE</code>	<code>C_INTERCONNECT_busif_READ_ISSUING</code>	
<code>C_M_AXI_WRITE_ISSUING</code>		<code>C_INTERCONNECT_busif_WRITE_ACCEPTANCE</code>
<code>C_M_AXI_READ_ISSUING</code>		<code>C_INTERCONNECT_busif_READ_ACCEPTANCE</code>

The `C_S_AXI_WRITE_ACCEPTANCE` and `C_S_AXI_READ_ACCEPTANCE` parameters limit the number of current outstanding transactions of each type that the crossbar will accept, per ID thread. For each ID value that can be received at each SI slot (based on the value in `C_S_AXI_THREAD_ID_WIDTH`), the crossbar maintains a separate Write and Read transaction counter. There is no limit on the total number of transactions (across all threads) that the crossbar will accept. Also, 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.

Regarding the acceptance counters:

- A Write transaction is considered to be complete (counter decremented) when a `BVALID/BREADY` handshake completes on the SI.
- A Read transaction is considered to be complete when a `RVALID/RREADY` handshake completes on the SI, in which `RLAST` is asserted

The `C_M_AXI_WRITE_ISSUING` and `C_M_AXI_READ_ISSUING` parameters limit the total number of current outstanding transactions of each type that the crossbar will issue (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 the issuing limits:

- A Write transaction is considered to be complete (counter decremented) when a `BVALID/BREADY` handshake completes on the MI
- A Read transaction is considered to be complete when a `RVALID/RREADY` handshake completes on the MI, in which `RLAST` is asserted

Write or Read transactions received at SI slots that have reached their acceptance limit, or that target a 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 or priority inversions when multiple masters access a shared slave.

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

- ISSUING parameters on connected master devices map to ACCEPTANCE parameters on the AXI Interconnect
- ACCEPTANCE parameters on connected slave devices map to ACCEPTANCE parameters on the Interconnect

**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.

### Arbitration Priority

Interconnect	Connected Master	Connected Slave
C_S_AXI_ARB_PRIORITY	C_INTERCONNECT_busif_ARB_PRIORITY	

Primarily, arbitration is granted based on the relative priority of the associated SI slot. The C\_S\_AXI\_ARB\_PRIORITY parameter can be set to a static priority value between 0 and 15; higher values have higher priority. The following conditions incur arbitration:

- 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.

When Write or Read transactions received at SI slots have reached their acceptance limit, or target a MI slot that has reached its issuing 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 (see [Transaction Acceptance and Issuing Limits](#), page 38.)

Also, any SI slot that attempts to access a slave in a manner that risks deadlock, is temporarily disqualified from arbitration (see [Cyclic Dependency Avoidance](#), page 21.)

### Crossbar Connectivity

Interconnect	Connected Master	Connected Slave
C_AXI_CONNECTIVITY		C_INTERCONNECT_busif_MASTERS
C_INTERCONNECT_CONNECTIVITY_MODE		
C_S_AXI_IS_INTERCONNECT		

The AXI Interconnect supports only the Single-Address/Multiple-Data (SAMD) crossbar connectivity model. Therefore, the C\_INTERCONNECT\_CONNECTIVITY\_MODE parameter is constant.



The SAMD model supports “sparse crossbar” connectivity, which means that the user can specify which masters need to access which slaves. By disabling unused master-slave pathways, data-path multiplexing logic and address decoding logic can be reduced, resulting in reduced FPGA resource utilization and faster timing paths.

The connectivity parameters are set as follows:

- Sparse connectivity information is represented by the `C_AXI_CONNECTIVITY` parameter on the AXI Interconnect.
- The `C_AXI_CONNECTIVITY` value is derived by accumulating the `C_INTERCONNECT_busif_MASTERS` parameters found on all the connected slave devices.
- The `C_INTERCONNECT_busif_MASTERS` parameter on each slave device lists all the master devices that need to access that slave, according to the instance name of each master and the name of the bus-interface on each master.

When using the XPS GUI to establish the bus connectivity of a system, the tools automatically insert `C_INTERCONNECT_busif_MASTERS` parameters in the Microprocessor Hardware System (MHS) design file.

Also, the `C_S_AXI_IS_INTERCONNECT` parameter affects crossbar connectivity. This parameter indicates whether any of the slave interface (SI) slots are being driven by another Interconnect.

Two AXI Interconnects can be cascaded by using the `axi2axi_connector` IP to bridge a MI slot of an upstream AXI Interconnect to a SI slot of a downstream Interconnect.

Under certain conditions, the tools set the value of the `C_S_AXI_IS_INTERCONNECT` parameter automatically when the tools detect an `axi2axi_connector` on an SI slot; no user intervention is required. This parameter is used by the downstream AXI Interconnect to avoid duplication of transaction thread control logic that is performed by the upstream AXI Interconnect, when possible.

**Note:** XPS tools assign consecutive SI slot numbers to connected masters automatically, and consecutive MI slot numbers to connected slaves, according to the order in which the master and slave instances and their bus-interfaces appear in the MHS file. There is no mechanism to allow users to explicitly control the assignment of slot numbers or to preserve slot number assignments across design iterations when master/slave instances and/or their bus-interface connections are added, deleted, or replaced. The AXI Interconnect does not support “holes” in the SI or MI.



## Read-Only and Write-Only Interfaces

**Table 9: Read-Only and Write-Only Parameters**

Interconnect	Connected Master	Connected Slave
C_S_AXI_SUPPORTS_WRITE	C_busif_SUPPORTS_WRITE	
C_S_AXI_SUPPORTS_READ	C_busif_SUPPORTS_READ	
C_M_AXI_SUPPORTS_WRITE		C_busif_SUPPORTS_WRITE
C_M_AXI_SUPPORTS_READ		C_busif_SUPPORTS_READ

Typically, the Read/Write-only parameters are specified as a constant in the MPD of the connected master/slave IP. However, some IP might support configurable read-only or write-only operation, which is then selectable by the user. The following rules apply to disabling the support for slave Reads and Writes:

- When the `SUPPORTS_WRITE` parameter is disabled, the connectivity pathways for the AW, W, and B channels of the corresponding SI or MI slot are disabled, similar to the way the `C_AXI_CONNECTIVITY` parameter disables pathways between specified master/slave pairs.
- When the `SUPPORTS_READ` parameter is disabled, the connectivity pathways for the AR and R channels of the corresponding SI or MI slot are disabled.
- When unused Write and Read channels are disabled, the data-path multiplexing logic and address decoding logic can be reduced, resulting in reduced FPGA resource utilization and faster timing paths.

For slaves that can be configured to support Read and Write transactions, the tools set the transaction values based on the values found in the connected master that access those slaves.

## Register Slices

Interconnect	Connected Master	Connected Slave
C_S_AXI_AW_REGISTER	C_INTERCONNECT_busif_AW_REGISTER	
C_S_AXI_AR_REGISTER	C_INTERCONNECT_busif_AR_REGISTER	
C_S_AXI_W_REGISTER	C_INTERCONNECT_busif_W_REGISTER	
C_S_AXI_R_REGISTER	C_INTERCONNECT_busif_R_REGISTER	
C_S_AXI_B_REGISTER	C_INTERCONNECT_busif_B_REGISTER	
C_M_AXI_AW_REGISTER		C_INTERCONNECT_busif_AW_REGISTER
C_M_AXI_AR_REGISTER		C_INTERCONNECT_busif_AR_REGISTER
C_M_AXI_W_REGISTER		C_INTERCONNECT_busif_W_REGISTER
C_M_AXI_R_REGISTER		C_INTERCONNECT_busif_R_REGISTER
C_M_AXI_B_REGISTER		C_INTERCONNECT_busif_B_REGISTER

Setting a `REGISTER` parameter enables a register slice at the outer-most periphery of the SI or MI. Each channel of each interface slot can be enabled independently. Register slices are provided mainly to improve system timing at the expense of one latency cycle. Register slices do not impact the data throughput rate of the Read data (R) or Write data (W) channels.

## Data Path FIFOs

Interconnect	Connected Master	Connected Slave
C_S_AXI_WRITE_FIFO_DEPTH	C_INTERCONNECT_busif_WRITE_FIFO_DEPTH	
C_S_AXI_READ_FIFO_DEPTH	C_INTERCONNECT_busif_READ_FIFO_DEPTH	
C_M_AXI_WRITE_FIFO_DEPTH		C_INTERCONNECT_busif_WRITE_FIFO_DEPTH
C_M_AXI_READ_FIFO_DEPTH		C_INTERCONNECT_busif_READ_FIFO_DEPTH

Setting a \*\_FIFO\_DEPTH parameter inserts a FIFO buffer on the data channel in either the SI or MI hemisphere, directly adjacent to the crossbar. The value of the parameter sets the depth of the FIFO (0 = no FIFO). The width of each FIFO is always the same as the native width of the crossbar (C\_INTERCONNECT\_DATA\_WIDTH).

## TrustZone Security

Interconnect	Connected Master	Connected Slave
C_M_AXI_SECURE		C_INTERCONNECT_busif_SECURE

Setting the C\_M\_AXI\_SECURE parameter enforces TrustZone security for each designated MI slot (connected slave) as a whole. If configured as a secure slot, only secure AXI accesses are permitted (S\_AXI\_AWPROT[1] or M\_AXI\_AWPROT[1] must be zero). Any non-secure accesses are blocked and the AXI Interconnect returns a DECERR response to the connected master. If the connected slave device provides its own TrustZone security feature, then you would generally not also set the C\_M\_AXI\_SECURE parameter on the Interconnect.

## Narrow Burst Support

Interconnect	Connected Master	Connected Slave
C_S_AXI_SUPPORTS_NARROW_BURST	C_busif_SUPPORTS_NARROW_BURST	
C_M_AXI_SUPPORTS_NARROW_BURST		C_busif_SUPPORTS_NARROW_BURST

Setting the C\_S\_AXI\_SUPPORTS\_NARROW\_BURST parameter to zero for a SI slot indicates that the connected master device is “well-behaved” and never issues any multi-beat transactions in which the SIZE of the data transfers is narrower than the interface data-width (“narrow bursts”). For example, a “well-behaved” master that has 64-bit wide WDATA/RDATA signals will always drive its AWSIZE/ARSIZE to 0x011 (8 bytes) when issuing transactions in which AWLEN/ARLEN > 0 (single-beat transactions may be of any SIZE.) For all connected masters that are not “well-behaved”, the C\_S\_AXI\_SUPPORTS\_NARROW\_BURST parameter must remain set to one (default).

Also, when C\_S\_AXI\_SUPPORTS\_NARROW\_BURST = 0 the AXI Interconnect:

- Ignores the AWCACHE[1] and ARCACHE[1] (“modifiable”) signals received on the corresponding SI slot
- Forces the bits to 1 internally and out to the MI

This allows any upsizers in the pathways driven by the SI slot to fully pack data so that the AXI Interconnect never produces narrow bursts on the MI for any transactions that originate from the “well-behaved” SI slot.

If a narrow burst is received on any SI slot for which `C_S_AXI_SUPPORTS_NARROW_BURST = 0`, the Interconnect returns a DECERR and does not propagate the transaction to the MI.

Setting the `C_M_AXI_SUPPORTS_NARROW_BURST` parameter to zero for a MI slot indicates that the connected slave device can expect to receive no narrow bursts and can therefore omit the associated data packing logic. The AXI Interconnect uses the parameter to enable data packing in upsizers in the MI hemisphere driving the designated MI slots; by default, MI side upsizers do not pack narrow bursts.

The XPS tools set the slave devices that feature configurable support for narrow burst `C_busif_SUPPORTS_NARROW_BURST` parameter to zero automatically when all master devices that can access the slave are well-behaved (have `C_busif_SUPPORTS_NARROW_BURST` set to zero).

## User Signals

Interconnect	Connected Master	Connected Slave
<code>C_AXI_SUPPORTS_USER_SIGNALS</code>	<code>C_busif_SUPPORTS_USER_SIGNALS</code>	<code>C_busif_SUPPORTS_USER_SIGNALS</code>
<code>C_AXI_AWUSER_WIDTH</code>	<code>C_busif_AWUSER_WIDTH</code>	<code>C_busif_AWUSER_WIDTH</code>
<code>C_AXI_ARUSER_WIDTH</code>	<code>C_busif_ARUSER_WIDTH</code>	<code>C_busif_ARUSER_WIDTH</code>
<code>C_AXI_WUSER_WIDTH</code>	<code>C_busif_WUSER_WIDTH</code>	<code>C_busif_WUSER_WIDTH</code>
<code>C_AXI_RUSER_WIDTH</code>	<code>C_busif_RUSER_WIDTH</code>	<code>C_busif_RUSER_WIDTH</code>
<code>C_AXI_BUSER_WIDTH</code>	<code>C_busif_BUSER_WIDTH</code>	<code>C_busif_BUSER_WIDTH</code>

When these parameters are specified on connected master and slave devices, they indicate whether those device interfaces include `USER` signals (on any channels) and, if so, the width of the `USER` signal on each channel.

By surveying the parameter values among connected master and slave devices, the tools determine if the AXI Interconnect needs to propagate `USER` signals between masters and slaves (on any channels) and, if so, the maximum required width of the `USER` signals to propagate on each channel. The tools therefore set the values of the `USER` parameters automatically on the AXI Interconnect based on the values found on the connected masters and slaves. Generally, the various `USER_WIDTH` parameter are used to determine the physical bit-width of the `USER` signals on the IP interface, and therefore must never be set to a value less than 1. When `C_AXI_SUPPORTS_USER_SIGNALS = 1`, at least 1 bit of `USER` signal is propagated on each of the five AXI channels.

## Parameter Rules Summary

In addition to the value ranges described in the parameter tables, the following specific rules apply:

- The `C_S_AXI_DATA_WIDTH` of each slot must be 32 when `C_S_AXI_PROTOCOL` denotes AXI4-Lite. Each `C_M_AXI_DATA_WIDTH` must be 32 when `C_M_AXI_PROTOCOL` denotes AXI4-Lite.
- For each used MI slot, at least one address range must be defined (non-null).
- For each used (non-null) address range, the range size must be a power-of-2 and at least 4096. `BASE_ADDR` must be a multiple of the range size (all low-order bits that select locations within the range must be zero). The low-order bits (within the range) of `HIGH_ADDR` must be all-ones.
- There must be no overlap among any of the address ranges across all MI slots.

- For each used SI slot, parameters `C_S_AXI_BASE_ID` and `C_S_AXI_THREAD_ID_WIDTH` must be defined.
- For each SI slot configured as AXI4-Lite, `C_S_AXI_IS_INTERCONNECT` must be 0 (end-point master device) and `C_S_AXI_THREAD_ID_WIDTH` must be 0. (AXI4-Lite interfaces use no ID signals.)
- For each used (non-null) ID range, all low-order bits of the `BASE_ID` parameter (as defined by `THREAD_ID_WIDTH`), if any, must be zero. In other words, `BASE_ID` must be a multiple of the range size ( $2^{**} \text{THREAD\_ID\_WIDTH}$ ).
- There must be no overlap among any of the ID ranges across all SI slots.
- The upper bound of each ID range ( $\text{BASE\_ID} + 2^{**} \text{THREAD\_ID\_WIDTH} - 1$ ) must not exceed the maximum ID value ( $2^{**} \text{C\_AXI\_ID\_WIDTH} - 1$ ) of the Interconnect.

## References

- *ARM AMBA AXI Protocol v2.0* (document number ARM IHI 0022C)  
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ih0022c/index.html>
- *Product Specification Format Reference Manual* (UG642)
- *AXI Reference Guide* (UG761)
- *AXI-to-AXI Connector IP* (DS803)

## Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

## Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
0921/2010	1.00.a	Initial Xilinx release.

## Notice of Disclaimer

Xilinx is providing this product documentation, hereinafter "Information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.