# SmartConnect v1.0

## *LogiCORE IP Product Guide*

**Vivado Design Suite**

**PG247 December 20, 2017**

XILINX
ALL PROGRAMMABLE™

# Table of Contents

# Introduction

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

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

The AXI SmartConnect is a Hierarchical IP block that is added to a Vivado® IP integrator block design in the Vivado Design Suite.

***Note:*** AXI SmartConnect is not available for direct (standalone) instantiation from the Xilinx IP catalog for use directly in a RTL design.

AXI SmartConnect is a drop-in replacement for the AXI Interconnect v2 core. SmartConnect is more tightly integrated into the Vivado design environment to automatically configure and adapt to connected AXI master and slave IP with minimal user intervention.

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

**Notes:**

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

*Chapter 1*

# Overview

## Feature Summary

- Up to 16 Slave Interfaces (SI) and up to 16 Master Interfaces (MI) per instance.
- Instances of SmartConnect can be cascaded to interconnect a larger number of masters/slaves or for organizing the interconnect topology.
- AXI Protocol compliant. Each SI and MI of SmartConnect can be connected to a master or slave IP interface of type AXI3, AXI4 or AXI4-Lite.
  - Transactions between interfaces of different protocol types are automatically converted by SmartConnect.
  - Burst transactions are automatically split, as required, to remain AXI compliant.
- Interface Data Widths (bits):
  - AXI4 and AXI3: 32,64,128,256, 512 or 1024.
  - AXI4-Lite: 32 or 64-bit.
- Transactions between interfaces of different data widths are automatically converted by SmartConnect.
- Supports multiple clock domains (the IP provides one clock pin per domain).
  - Transactions between interfaces in different clock domains are automatically converted by SmartConnect.
- Address width: Up to 64 bits:
  - SmartConnect decodes up to 256 total address range segments.
- User defined signals up to 512 bits wide per channel.
  - User signals on any AXI channel are propagated regardless of internal transaction conversions.
- ID width: Up to 32 bits.
  - Automatic re-mapping/compression of wide input ID signals.
- Support for Read-only and Write-only masters and slaves, resulting in reduced resource utilization.

- Supports multiple outstanding transactions:
  - Supports connected masters with multiple reordering depth (ID threads).
  - Supports write response reordering, Read data reordering, and Read Data interleaving.
  - Multi-threaded traffic (propagation of ID signals) is supported regardless of internal transaction conversions, including data width conversion and transaction splitting.
  - Optional single ordering mode (per SI and MI). Stores ID values internally instead of propagating to the slave, resulting in reduced resource utilization.
- *Single-Slave per ID* method of cyclic dependency (deadlock) avoidance.
  - For each ID thread issued by a connected master, the SmartConnect allows one or more outstanding transactions to only one slave device for Writes and one slave device for Reads, at a time.
- Multiple parallel pathways along all AXI channels when connected to multiple masters and multiple slaves:
  - Each AXI channel has independent destination-side arbitration. Transfers from two or more source endpoints to separate destination endpoints can occur concurrently, for any AXI channel.
  - Round-robin arbitration for each of the AW, AR, R and B channels. (W-channel transfers follow the same order as AW-channel arbitration, per AXI protocol rules.)
- Supports back-to-back transfers (100% duty cycle) on any AXI channel:
  - Single data-beat transactions can traverse the SmartConnect at the same bandwidth as multi-beat bursts.
- Supports TrustZone security for each connected slave:
  - If configured as a secure address segment, only secure AXI accesses are permitted according to the AXI `arprot` or `awprot` signal.
  - Any non-secure accesses are blocked and the AXI SmartConnect core returns a `decerr` response to the connected master.
- Internally resynchronized reset:
  - One `aresetn` input per IP.

# Applications

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

# AXI SmartConnect Core Limitations

These limitations apply to the AXI SmartConnect core:

- SmartConnect unconditionally packs all multi-beat bursts to fill the interface data-width.

  SmartConnect SI interfaces accept *narrow* bursts, in which the `arsize` or `awsize` signal indicates data units which are smaller than the interface data-width. But such bursts are always propagated through the SmartConnect and its MI interfaces fully packed. The *modifiable bit* of the AXI `arcache` or `awcache` signal does not prevent packing.

- SmartConnect converts all WRAP type bursts into INCR type. SmartConnect SI interfaces accept all protocol-compliant WRAP bursts, beginning at any target address. But such bursts are always converted to a single INCR burst beginning at the *wrap address*. This may increase response latency of unaligned read wrap bursts.
- SmartConnect does not support FIXED type bursts. Any FIXED burst transaction received at the SmartConnect SI is blocked and a DECERR response is returned to the master.
- SmartConnect does not propagate original ID values from endpoint masters. IDs received at an SI interface are re-mapped to a smaller (or equal) number of bits for more resource-efficient management of multi-threaded traffic.
- SmartConnect appends ID bits to differentiate among multiple masters, when propagating transactions to the MI. Values of master identification bits are assigned by IP integrator and cannot be controlled or predicted by the user.
- The AXI SmartConnect core does not support discontinued AXI3 features:
  - **Atomic locked transactions**. This feature was retracted by the AXI4 protocol. A locked transaction is changed to a non-locked transaction and propagated by the MI.
  - **Write interleaving**. This feature was retracted by AXI4 protocol. AXI3 master devices must be configured as if connected to a slave with a Write interleaving depth of one.
- All arbitration on all AXI channels is round-robin. SmartConnect does not support fixed priority arbitration.
- AXI4 Quality of Service (`arqos` and `awqos`) signals do not influence arbitration priority. QoS signals are propagated from SI to MI.
- SmartConnect neither propagates nor generates the AXI4 `arregion` or `awregion` signal.

- SmartConnect does not support independent reset domains. If any master or slave device connected to SmartConnect is reset, then all connected devices must be reset concurrently.
- SmartConnect does not support low-power mode or propagate the AXI C channel signals.
- SmartConnect does not time out if the destination of any AXI channel transfer stalls indefinitely. All connected AXI slaves must respond to all received transactions, as required by AXI protocol.
- SmartConnect provides no address remapping.
- SmartConnect does not include conversion or bridging to non-AXI protocols, such as APB.

# Licensing and Ordering Information

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

# Product Specification

## Standards

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

## Performance

For details about performance, visit Performance and Resource Utilization.

## Resource Utilization

For details about resource utilization, visit Performance and Resource Utilization.

# Port Descriptions

This section lists the interface signals for the AXI SmartConnect core.

In Table 2-1 through Table 2-3, the *Default* column shows whether the input signal is required (REQ) or, if not, its default value if left unconnected. Signal connections are required only for the SIs and MIs that are used. Signals which are not used in a particular protocol configuration are indicated by *d/c* (do not care).

## Slave Interface I/O Signals

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

### *Slave Interface I/O Signals*

*Table 2-1:* **AXI SmartConnect Core Slave I/O Signals**

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| snn_axi_awid | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–32] | Write Address Channel Transaction ID. |
| snn_axi_awaddr | Input | REQ | [2–64] | Write Address Channel Address. |
| snn_axi_awlen | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | AXI4: 8<br>AXI3: 4 | Write Address Channel Burst Length (0–255). |
| snn_axi_awsize | Input | AXI3, AXI4:<br>See Description<br>AXI4-Lite: d/c | 3 | Write Address Channel Transfer Size code (0–7). When not connected, the data transfer size for all transactions is assumed to be the full interface data width. |
| snn_axi_awburst | Input | AXI3, AXI4: 0b01<br>AXI4-Lite: d/c | 2 | Write Address Channel Burst Type code (0–2). When not connected, all transactions are assumed to be increment burst type (INCR). |
| snn_axi_awlock | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | AXI4: 1<br>AXI3: 2 | Write Address Channel Atomic Access Type:<br>0=No locked access<br>1=Exclusive Access<br>2, 3= Not supported |
| snn_axi_awcache | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | 4 | Write Address Channel Cache Characteristics. |
| snn_axi_awprot | Input | 0b000(1) | 3 | Write Address Channel Protection Bits. |

*Table 2-1:* **AXI SmartConnect Core Slave I/O Signals *(Cont'd)***

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| snn_axi_awqos[(2)] | Input | AXI4: 0<br>AXI4-Lite: d/c | 4 | AXI4 Write Address Channel Quality of Service. |
| snn_axi_awuser | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–512][(3)] | User-defined AW Channel signals. |
| snn_axi_awvalid | Input | REQ | 1 | Write Address Channel Valid. |
| snn_axi_awready | Output | | 1 | Write Address Channel Ready. |
| snn_axi_wid | Input | AXI3: 0<br>AXI4, AXI4-Lite: d/c | [1–32] | Write Data Channel Transaction ID for AXI3 masters. |
| snn_axi_wdata | Input | REQ | [32, 64, 128, 256, 512, 1024] | Write Data Channel Data. |
| snn_axi_wstrb | Input | all ones | [32, 64, 128, 256, 512, 1024] / 8 | Write Data Channel Byte Strobes. |
| snn_axi_wlast | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | 1 | Write Data Channel Last Data Beat. |
| snn_axi_wuser | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–512][(3)] | User-defined W Channel signals; must be an integer number of bits per byte of `wdata`. |
| snn_axi_wvalid | Input | REQ | 1 | Write Data Channel Valid. |
| snn_axi_wready | Output | | 1 | Write Data Channel Ready. |
| snn_axi_bid | Output | | [1–32] | Write Response Channel Transaction ID. |
| snn_axi_bresp | Output | | 2 | Write Response Channel Response Code (0–3). |
| snn_axi_buser | Output | | 1-512[(3)] | User-defined B Channel signals. |
| snn_axi_bvalid | Output | | 1 | Write Response Channel Valid. |
| snn_axi_bready | Input | REQ | 1 | Write Response Channel Ready. |
| snn_axi_arid | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–32] | Read Address Channel Transaction ID. |
| snn_axi_araddr | Input | REQ | [2–64] | Read Address Channel Address. |
| snn_axi_arlen | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | AXI4: 8<br>AXI3: 4 | Read Address Channel Burst Length code (0–255). |
| snn_axi_arsize | Input | AXI3, AXI4:<br>See Description<br>AXI4-Lite: d/c | 3 | Write Address Channel Transfer Size code (0-7). When not connected, the data transfer size for all transactions is assumed to be the full interface data width. |
| snn_axi_arburst | Input | AXI3, AXI4: 2b01<br>AXI4-Lite: d/c | 2 | Write Address Channel Burst Type code (0-2). When not connected, all transactions are assumed to be increment burst type (INCR). |

*Table 2-1:* **AXI SmartConnect Core Slave I/O Signals *(Cont'd)***

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| snn_axi_arlock | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | AXI4: 1<br>AXI3: 2 | Read Address Channel Atomic Access Type:<br>0=No locked access<br>1=Exclusive Access<br>2, 3= Not supported |
| snn_axi_arcache | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | 4 | Read Address Channel Cache Characteristics. |
| snn_axi_arprot | Input | 0b000(1) | 3 | Read Address Channel Protection Bits. |
| snn_axi_arqos(2) | Input | AXI4: 0<br>AXI4-Lite: d/c | 4 | AXI4 Read Address Channel Quality of Service. |
| snn_axi_aruser | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–512](3) | User-defined AR Channel signals. |
| snn_axi_arvalid | Input | REQ | 1 | Read Address Channel Valid. |
| snn_axi_arready | Output | | 1 | Read Address Channel Ready. |
| snn_axi_rid | Output | | [1–32] | Read Data Channel Transaction ID. |
| snn_axi_rdata | Output | | [32, 64, 128, 256, 512, 1024] | Read Data Channel Data. |
| snn_axi_rresp | Output | | 2 | Read Data Channel Response Code (0–3). |
| snn_axi_rlast | Output | | 1 | Read Data Channel Last Data Beat. |
| snn_axi_ruser | Output | | [1–512](3) | User-defined R Channel signals. |
| snn_axi_rvalid | Output | | 1 | Read Data Channel Valid. |
| snn_axi_rready | Input | REQ | 1 | Read Data Channel Ready. |

**Notes:**

1. AXI protocol requires master devices to drive their `awprot/arprot` outputs. If the `awprot/arprot` signals are left undriven, it defaults to all zeros and the transaction is interpreted as secure.
2. Although the `awqos/arqos` signals are defined only by the AXI4 protocol specification, this SmartConnect IP core also propagates QoS signals for any SI configured as AXI3.
3. When connected to another SmartConnect instance (cascaded), the width of the user signal for each channel is 1024, and the signal carries proprietary SmartConnect control fields in addition to the user-defined signal value.

### Master Interface I/O Signals

Table 2-2 lists the Master Interface signals for the AXI SmartConnect core. In the *Signal Name* column "nn" represents a two-digit sequence number (with leading zero) with range 00 ≤ nn ≤ N–1, where N refers to the total number of configured Master Interfaces, which is the number of slave devices connected to the AXI SmartConnect core. Each row in the table therefore defines N interface signals. When a range of values is specified in the Width column, the signal width is determined by the tools based on system connectivity.

*Table 2-2:* **AXI SmartConnect Core Master I/O Signals**

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| mnn_axi_awid | Output | | [1–32] | Write Address Channel Transaction ID. |
| mnn_axi_awaddr | Output | | [2–64] | Write Address Channel Address. |
| mnn_axi_awlen | Output | | AXI4: 8<br>AXI3: 4 | Write Address Channel Burst Length code. (0–255). |
| mnn_axi_awsize | Output | | 3 | Write Address Channel Transfer Size code (0–7). |
| mnn_axi_awburst | Output | | 2 | Write Address Channel Burst Type. This signal (if enabled) will always be driven to 2b01 (INCR burst type). |
| mnn_axi_awlock | Output | | AXI4: 1<br>AXI3: 2 | Write Address Channel Atomic Access Type<br>(0, 1). |
| mnn_axi_awcache | Output | | 4 | Write Address Channel Cache Characteristics. |
| mnn_axi_awprot | Output | | 3 | Write Address Channel Protection Bits. |
| mnn_axi_awqos(1) | Output | | 4 | Write Address Channel Quality of Service. |
| mnn_axi_awuser | Output | | [1–512](2) | User-defined AW Channel signals. |
| mnn_axi_awvalid | Output | | 1 | Write Address Channel Valid. |
| mnn_axi_awready | Input | REQ | 1 | Write Address Channel Ready. |
| mnn_axi_wid | Output | | [1–32] | Write Data Channel Transaction ID for AXI3 slaves. |
| mnn_axi_wdata | Output | | [32, 64, 128, 256, 512, 1024] | Write Data Channel Data. |
| mnn_axi_wstrb | Output | | [32, 64, 128, 256, 512, 1024] / 8 | Write Data Channel Data Byte Strobes. |
| mnn_axi_wlast | Output | | 1 | Write Data Channel Last Data Beat. |

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

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| mnn_axi_wuser | Output | | [1–512](2) | User-defined W Channel signals. |
| mnn_axi_wvalid | Output | | 1 | Write Data Channel Valid. |
| mnn_axi_wready | Input | REQ | 1 | Write Data Channel Ready. |
| mnn_axi_bid | Input | AXI3, AXI4: REQ<br>AXI4-Lite: d/c | [1–32] | Write Response Channel Transaction ID. |
| mnn_axi_bresp | Input | 0b00 | 2 | Write Response Channel Response Code (0–3). |
| mnn_axi_buser | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–512](2) | User-defined B Channel signals. |
| mnn_axi_bvalid | Input | REQ | 1 | Write Response Channel Valid. |
| mnn_axi_bready | Output | | 1 | Write Response Channel Ready. |
| mnn_axi_arid | Output | | [1–32] | Read Address Channel Transaction ID. |
| mnn_axi_araddr | Output | | [2–64] | Read Address Channel Address. |
| mnn_axi_arlen | Output | | AXI4: 8<br>AXI3: 4 | Read Address Channel Burst Length code (0–255). |
| mnn_axi_arsize | Output | | 3 | Read Address Channel Transfer Size code (0–7). |
| mnn_axi_arburst | Output | | 2 | Read Address Channel Burst Type. This signal (if enabled) will always be driven to 2b01 (INCR burst type). |
| mnn_axi_arlock | Output | | AXI4: 1<br>AXI3: 2 | Read Address Channel Atomic Access Type (0,1). |
| mnn_axi_arcache | Output | | 4 | Read Address Channel Cache Characteristics. |
| mnn_axi_arprot | Output | | 3 | Read Address Channel Protection Bits. |
| mnn_axi_arqos(1) | Output | | 4 | AXI4 Read Address Channel Quality of Service. |
| mnn_axi_aruser | Output | | [1–512](2) | User-defined AR Channel signals. |
| mnn_axi_arvalid | Output | | 1 | Read Address Channel Valid. |
| mnn_axi_arready | Input | REQ | 1 | Read Address Channel Ready. |
| mnn_axi_rid | Input | AXI3, AXI4: REQ<br>AXI4-Lite: d/c | [1–32] | Read Data Channel Transaction ID. |
| mnn_axi_rdata | Input | REQ | [32, 64, 128, 256, 512, 1024] | Read Data Channel Data. |
| mnn_axi_rresp | Input | 0b00 | 2 | Read Data Channel Response Code (0–3). |

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

| Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| mnn_axi_rlast | Input | AXI3, AXI4: REQ<br>AXI4-Lite: d/c | 1 | Read Data Channel Last Data Beat. |
| mnn_axi_ruser | Input | AXI3, AXI4: 0<br>AXI4-Lite: d/c | [1–512](2) | User-defined R Channel signals. |
| mnn_axi_rvalid | Input | REQ | 1 | Read Data Channel Valid. |
| mnn_axi_rready | Output | | 1 | Read Data Channel Ready. |

**Notes:**

1. Although the QOS signals are defined only by the AXI4 protocol specification, this SmartConnect IP core also propagates QOS signals for any MI configured as AXI3.
2. When connected to another SmartConnect instance (cascaded), the width of the user signal for each channel is 1024, and the signal carries proprietary SmartConnect control fields in addition to the user-defined signal value.

*Table 2-3:* **AXI SmartConnect Core Global Port Signals**

| Port Signal Name | Direction | Default | Width | Description (Range) |
|---|---|---|---|---|
| aclk | Input | REQ | 1 | SmartConnect clock input. |
| aclk1…aclkn | Input | REQ when enabled | 1 | Clock inputs for additional interface clock domains. |
| aclken, aclken1…aclkenn | Input | 1 | 1 | Clock enable input associated with each clock domain. |
| aresetn | Input | REQ | 1 | SmartConnect Reset (active-Low). |

# Register Space

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

# Designing with the Core

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

## AXI SmartConnect Core Functionality

These subsections describe the functionality within the AXI SmartConnect core.

### Address Decode

The SmartConnect core must determine which MI is the target of each transaction by decoding the address of each AW and AR channel transfer received at an SI. This address decode involves only those upper order address bits needed to distinguish between MIs, 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 (up to the width of the address signal port of the connected slave). It is visible to any connected monitors, even though the high-order address bits are typically not reused by the slave device.

In some cases, there may be multiple, possibly disjointed, address ranges that define when a single slave device is accessed. The address decode logic in the SmartConnect core includes the multiple ranges that determine the selection of each MI.

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

**IMPORTANT**: All address ranges must be a power of two. The base address of all ranges must be aligned to (an integer multiple of) their size. There must be no overlap among all address ranges across all MI. These rules are enforced by the tools.

Each SI maintains an address map table containing only those address segments configured for the connected master device in the IP integrator Address Editor.

When all address segments associates with an MI (connected slave) are omitted (unmapped) from a master's address space, SmartConnect prevents that master from propagating any transactions to that MI. If a subset (aperture) of a slave's address segment is mapped to a master's address space, then only transactions with addresses that fall within the mapped aperture are permitted to propagate to the MI; attempts to access excluded portions of address segments will result in DECERR responses.

SmartConnect performs no address re-mapping. The address segment(s) corresponding to each MI are seen the same way by all SIs that can access the MI (*flat* address space). Masters with an address width less than the full address width spanned by all accessible slaves, can only access those address segments mapped at the low end of the address space.

## Use of ID Signals

The transaction ID signals that propagate from SIs to MIs (`awid` and `arid`) and back again (`bid` and `rid`), control both the routing of response transfers, and the ordering of AXI transfer propagation within the SmartConnect.

Endpoint master devices can optionally output `awid` and `arid` signals that the master device can use to select among multiple threads of transactions (as though the master IP core was comprised of multiple master devices internally). The *reordering depth* is the total number of ID values that can be generated by a master, as determined by the width of the master's ID signals. Master devices with a reordering depth of one do not need ID signals on their interface. The width of ID signals may vary among SIs.

AXI transaction ordering rules are as follows:

- There are no rules regarding the relative ordering between write transactions and read transactions.
- Transactions (of each direction) belonging to the same thread must be returned in order.
- Transactions (of each direction) among different threads can be returned out-of-order.

All response transfers on R or B channels of an SI contain rid or bid values that match the `arid` or `awid` (if present) of the original commands issued by the master on the AW or AR channel of the SI. However, the SmartConnect does not preserve these original ID values when propagating transactions to the MI. Depending on the configuration of the SmartConnect, thread ID values may either be re-mapped (to a more compact ID space) or suppressed entirely and stored internally.

When an SI is configured in *single-ordered* mode, all IDs received on the SI are stored and retrieved internally. The connected master may still issue multiple outstanding transactions, and the transaction IDs (if any) may have any value. However, all transactions issued by the master are propagated and returned in order. No ID information is propagated to the slave, resulting in the most resource-efficient implementation.

If an SI is configured to propagate IDs, then transaction ID values received from the connected master are dynamically re-mapped to an ID of equal or smaller width. This reduces the total number of ID threads monitored by the SmartConnect, resulting in a more resource-efficient implementation.

If there are multiple SIs, the SmartConnect makes ID values among all SIs unique before propagating to the MI. The SmartConnect core appends a constant unique *master-ID* value to the re-mapped thread-ID from the SI (if any). The master-ID values are assigned by the tools. Master IP in the Xilinx IP catalog or packaged by the user may include metadata indicating the master's reordering depth, which is the number of ID threads that should be tracked by SmartConnect for the connected SI. By default, the SI is configured in *single-ordered* mode.

## Transaction Arbitration

For each MI that can be accessed by multiple SIs, round-robin arbiters in the MI select the read command (AR channel) and write command (AW channel) to issue from the MI. A new read/write command can be issued each clock cycle, regardless of whether re-arbitrating a command from a different SI, or issuing a subsequent command from the same SI (back-to-back command arbitration). Commands presented to different MIs can be arbitrated and issued concurrently.

For each SI that accesses multiple MIs, round-robin arbiters in the SIs select among the R-channel and B-channel responses returned from the various MIs. These arbiters also support back-to-back transfers. Along the R-channel, the response arbiters support interleaving of read data beats among multiple MIs, without waiting for the end of the read burst (`rlast`) before re-arbitrating.

## Cyclic Dependency Avoidance

When there is more than one transaction ID (issued by one or more master devices) on which multiple outstanding transactions can be issued, and there is more than one connected slave device that can queue multiple transactions, and any of the slave devices can respond out-of-order on either the R or B channel, there is a potential cyclic dependency (deadlock) risk.

### How Deadlock Occurs

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

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 reorders the Read response, which is allowable because the transaction IDs received on transactions from different masters are different. However, the SmartConnect core must not 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 reorder). But the SmartConnect must not 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 SmartConnect uses the *Single Slave per ID* method to avoid deadlock. This method does not impact the performance of the transactions of most critical concern. These are the pipelining of multiple Reads and Writes by multiple master devices, to a shared performance-critical slave device, such as a memory controller.

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

By imposing this rule in the example shown in the previous section, the Read transaction from M0 to S1 in step 2 is stalled until S0 completes its response to M0. Similarly, the transaction from M1 to S0 in step 4 is stalled until S1 completes its response to M1. However the transactions proceed, under these conditions, interdependencies that could cause deadlock are avoided.

As well as preventing deadlock, the *Single Slave per ID* rule also guarantees in-order completion of all Write transactions at the SIs, even if different MIs are targeted by a transaction thread in successive transactions. For example, a master device writes to a direct memory access (DMA) descriptor in memory, then writes (using the same thread-ID) to a control register in a DMA engine which subsequently reads that descriptor. Because SmartConnect does not allow the second Write to propagate to the DMA slave device until the first Write completes (Write response received from the memory controller), there is no risk that the DMA reads steal descriptor data from memory. Each master device is therefore guaranteed in-order completion of transactions to various slave devices, in the same direction, and on the same ID thread. Therefore, under those conditions, master devices do not need to condition subsequent Write transactions on receiving Write responses for prior transactions.

**IMPORTANT:** *AXI protocol provides no method of ensuring 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 SmartConnect are as follows:

- **Address decode error**: No eligible MI mapped to the address of the transaction, according to the address mapping for the SI (and whether the target slave supports read or write transactions).
- An address segment with the SECURE parameter enabled is targeted by a transaction in which `awprot[1]` or `arprot[1]` is set (unsecure).
- A transaction is received in which the bust type (`arburst` or `awburst`) indicates a FIXED burst.

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

The SmartConnect does not detect the following error conditions:

- The response ID received at the MI does not match any outstanding command ID value. This is indicative of a slave malfunction or system connectivity error that violates AXI protocol.
- AXI4 protocol violations caused by connected endpoint IP. The AXI protocol checker IP should be deployed to debug such conditions.
- Write data interleaving from an AXI3 master configured with a write reordering depth greater than one. All Write data is routed in the same order as AW command are issued. W-channel `wid` inputs are ignored by SmartConnect.
- The tools generally enforce design rules which prevent erroneous configurations at compile time. Therefore, no error detection is provided by the AXI SmartConnect for these configuration errors:
  - Parameter value range violations.
  - Address range overlap, non-binary size or base value misalignment.

## Width Conversion

Each of the SIs and MIs on the AXI SmartConnect core can be connected to a master or slave endpoint with a data width of 32, 64, 128, 256, 512, or 1024 bits. When a transaction at an SI targets an MI with a different data width, width conversion is automatically performed along the pathway. The internal data pathways that connect SIs to MIs vary in width.

All data width conversions support the propagation of ID signals. Out-of-order response transfers resulting from multi-threaded traffic (if enabled) is managed by the width converters.

The width conversion transformations differ depending on whether the datapath width widens (upsizing) or narrows (downsizing) when moving from the SI toward the MI.

### AXI Downsizer

When the SI is wider than the MI, downsizing is performed and, in the transaction issued to the MI, the number of data beats is multiplied up accordingly.

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

  During merging, the read error response code (`rresp`) for each output data beat produced on the SI is set to the worst-case error condition encountered among the input data beats being merged, according to the following descending precedence order: DECERR, SLVERR, OKAY, EXOKAY.

On the AW or AR command channel, SmartConnect factors up the length of each burst and detects when the resulting burst length would exceed the maximum burst limit (256 data beats for AXI4, 16 for AXI3, or 1 for AXI4Lite). In such cases, the core splits the transaction automatically into multiple conforming burst transactions.

- Exclusive Access is not supported through downsizers when burst lengths require splitting. If the `awlock` or `arlock` signal indicates an Exclusive Access write or read transaction, and downsizing results in splitting, then the core changes the `lock` signal in all resulting output transactions to indicate Normal Access (0).

- When a downsized Write transaction results in splitting, the core coalesces the multiple Write responses received at the MI and issues one Write response on the SI. The core sets the error response code (BRESP) to the worst-case error condition encountered among the multiple input responses, according to the following descending precedence order: DECERR, SLVERR, OKAY (EXOKAY cannot occur in a split transaction).

### AXI Upsizer

When the MI is wider than the SI, upsizing is performed, and in the resulting transaction issued to the MI side, the number of data beats is reduced accordingly.

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

  The AXI SmartConnect core replicates the `rresp` from each MI-side (wide) input read data beat onto the `rresp` of each of the resulting SI-side (narrow) output data beats.

Transactions always remain fully packed when upsizing both writes and reads. Data packing is not disabled in response to the *modifiable* bit (`awcache`[1] or `arcache`[1]) of the address transfer. Upsizing does not cause transaction splitting.

## User Defined Signal Propagation

User-defined signals on the W and R channels (`wuser` and `ruser`) are always formatted in terms of user bits per byte of data. As data is serialized or merged during downsizing or upsizing, the bit-lanes associated with each byte of data travel with the data bytes. When an SI and MI have the same number of user bits per byte, the total widths of their `wuser` and `ruser` signals remain in proportion to their data widths, and all user bits will propagate through the intervening width conversion. When the user bits per byte differ between the SI and MI, padding or truncation of high-order user bit positions is performed on a per-byte-lane basis. Propagation of `wuser` and `ruser` signals by SmartConnect is suitable for transporting byte-wise parity information between a data source that generates parity and a data destination that detects parity errors. The SmartConnect neither generates nor detects parity.

The propagation of user bits on R and W channels of an AXI4 memory-mapped interface, when traversing width conversion, is not prescribed by the AXI4 protocol specification. However, it is defined for the AXI4-Stream protocol. Xilinx SmartConnect uses the same transformation for R and W channel width conversion as prescribed in the AXI4-Stream specification.

Width conversion does not affect the propagation of user signals on the AR, AW and B channels. However, when transactions are split as a result of downsizing or AXI3 protocol conversion, the entire user signal received on the AR or AW channel is replicated in all resulting transfers on the MI. Conversely, when multiple B-channel transfers received on the MI are consolidated as a result of a split write transaction, only the user signal received on the last of the consolidated B transfers is propagated to the SI; user information received on earlier B transfers is discarded.

## Protocol Conversion

Each of the SIs and MIs on the AXI SmartConnect core can be individually connected to masters and slaves of protocol types AXI4, AXI3, or AXI4-Lite. When a slave of type AXI3 or AXI4-Lite is connected to an MI of the SmartConnect, protocol conversion logic is automatically included along the internal pathway.

### *Conversion to AXI4-Lite*

AXI4 or AXI3 master devices can issue transactions through the SmartConnect to an AXI4-Lite slave. SmartConnect automatically inserts the required protocol conversion logic.

The transaction ID (`awid` or `arid`) received at the SI is stripped and stored internally, and retrieved during response transfers as bid or rid.

SmartConnect converts AXI4/AXI3 bursts into a sequence of single-beat transactions for AXI4-Lite slaves.

**Downsizing to an AXI4-Lite slave:**

By default, when a wide master issues a single-beat transaction to a narrower AXI4-Lite slave, SmartConnect converts the wide transfer to a series of narrow single-beat transactions that span all the address locations in the master's original transfer. If the master intends to write to only one location in the slave without disturbing adjacent locations, there are two alternatives:

1. During the master's write data transfer, disable all WSTRB bits for byte locations not to be overwritten. During downsizing, SmartConnect converts the wide transfer to a series of transactions matching the target slave's data-width. However, SmartConnect suppresses any output transfer in which the WSTRB bits for all byte positions in the transfer are zero. Some control-register slaves do not honor the WSTRB inputs and overwrite any locations that gets accessed. By suppressing null write transfers, SmartConnect writes only to locations where the WSTRB bit-string is non-zero for each output word.
2. During the master's write address transfer, drive a value onto the AWSIZE output that corresponds to the slave's physical data width. Then during downsizing, SmartConnect will convert to only one output transaction to the slave, because there are no other locations being accessed by the master. Except for this application, it is recommended that master endpoints do not drive their AWSIZE or ARSIZE outputs with values indicating transfers narrower than the master's physical data width.

### *AXI4-to-AXI3 Conversion*

When an AXI4 master device issues a write transaction to an AXI3 slave, the SmartConnect produces the required WID output on the MI based on the AWID received at the SI. If a burst longer than 16 data beats is received, the command is split into several shorter burst transactions.

Transaction splitting due to AXI3 conversion is similar to the splitting that may result from downsizing. User-defined signals received on the AW and AR channels are replicated onto all resulting MI-side transactions. Conversely, when multiple B-channel transfers received on the MI are consolidated as a result of a split write transaction, only the user signal received on the last of the consolidated B transfers is propagated to the SI; user information received on earlier B transfers is discarded. Propagation of ID signals and multi-threaded traffic propagation is not limited as a result of transaction splitting. However, Exclusive Access transactions are not supported when transactions are split, and the `awlock` and `arlock` outputs are always forced to zero on the MI when a transaction is split.

### *Other Conversions*

Transactions issued from AXI4-Lite masters to AXI4 or AXI3 slaves, and transactions from AXI3 masters to AXI4 slaves, require no conversion logic to be inserted. Output signals present in the MI-side protocol which are not present in the SI-side protocol are set to their default values.

## Internal Payload Buffering

Transfers on all five AXI channels are buffered on both the SI and MI-sides of the central switching plane to support high throughput and reduce throttling. The depth of buffering and type of storage resource (distributed RAM or BRAM) is determined automatically based on the bandwidth requirements of the connected masters and slaves. The buffers on the SI side of the AW and AR channels can optionally operate in packet mode to avoid full/empty stalls within bursts.

For *Write* packet mode, the issuing of the AW channel transfer from the buffer is delayed until the entire write burst has been stored in the W-Channel buffer (`wlast` is received), therefore avoiding stalling due to a slow write data source. To avoid deadlock, the AW command is alternatively issued whenever the SI receives a number of data beats of the same burst that exceeds the total SI-side buffer capacity.

For *read* packet mode, the issuing of the AR channel transfer is delayed until the R-Channel buffer has enough vacancy to store the entire burst, according to `ARLEN`. (*vacancy* is defined as the amount of free space in the R channel FIFO that has not already been committed by previously issued AR commands.) This avoids stalling due to a slow read destination. The first AR command received when there are no outstanding reads is always issued immediately. Beyond that, delaying of AR channel commands begins only after the data accumulated in the R channel FIFO has reached a designated threshold, and therefore does not contribute to read command latency.

## Pipelining

SmartConnect offers various pipelining and register slice options that can be user configured to tune system trade-offs between area, latency, and timing. Enabling pipelines and register slices can improve timing, but can also introduce latency and consume significant area, especially with large data widths.

Register slices options are available on each SI (see Snn_Entry Advanced Properties) and MI (see Mnn_Exit Advanced Properties) interface of the SmartConnect to isolate its timing from an attached AXI master/slave IP.

Around the buffer and switchboard blocks inside the SmartConnect, there are additional SI (Snn_Buffer Advanced Properties) and MI (Mnn_Buffer Advanced Properties) pipelining options which introduce ranks of FFs that can improve timing internal to the SmartConnect. Options for SLR pipeline ranks are suitable for SLR crossings and general purpose pipelining within an SLR.

# Design Parameters

This section lists the configuration parameters for the AXI SmartConnect core.

*Table 3-1:* **AXI SmartConnect Core Global Parameters**

| Parameter Name | Default Value | Format/Range | Description |
|---|---|---|---|
| NUM_SI | 2 | Integer (1-16) | Number of Slave Interfaces |
| NUM_MI | 1 | Integer (1-16) | Number of Master Interfaces |
| NUM_CLKS | 1 | Integer (1-33) | Number of clock inputs |
| HAS_ARESETN | 1 | 0,1 | Enable aresetn input |
| ADVANCED_PROPERTIES | | String | Contains a Tcl dictionary specifying additional property assignments to control advanced features of the SmartConnect. See Advanced Properties for more information. |

## Advanced Properties

### *General Operation*

Clicking **Show Advanced Properties** (Figure 3-1) presents the advanced properties for viewing and editing. These are only available after the IP integrator diagram containing the SmartConnect IP has been successfully validated.

Successful validation of the IPI diagram containing the SmartConnect is required because the set of available advanced options depends on information that is only available once the SmartConnect instance has self-configured its internal logic to meet the configuration requirements of its attached endpoints. The **Show Advanced Properties** button may be disabled if changes are made to the IP integrator diagram without a subsequent re-validation.

***Note:*** If the properties of the SmartConnect instance (number of masters, number of clocks, etc.) are changed, the diagram should be re-validated before subsequent changes to the advanced properties.

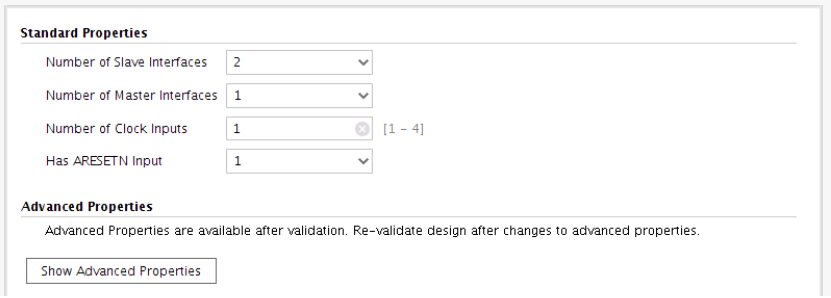


*Figure 3-1:* **Show Advanced Properties**

The advanced properties view consists of three tabs (as shown in Figure 3-2):

- **Functional** - Used to configure functional settings.
- **Timing** - Used for tuning the timing / pipeline properties of the internal logic for Fmax/Area optimization.
- **Clocking** - Used to configure clocking settings.

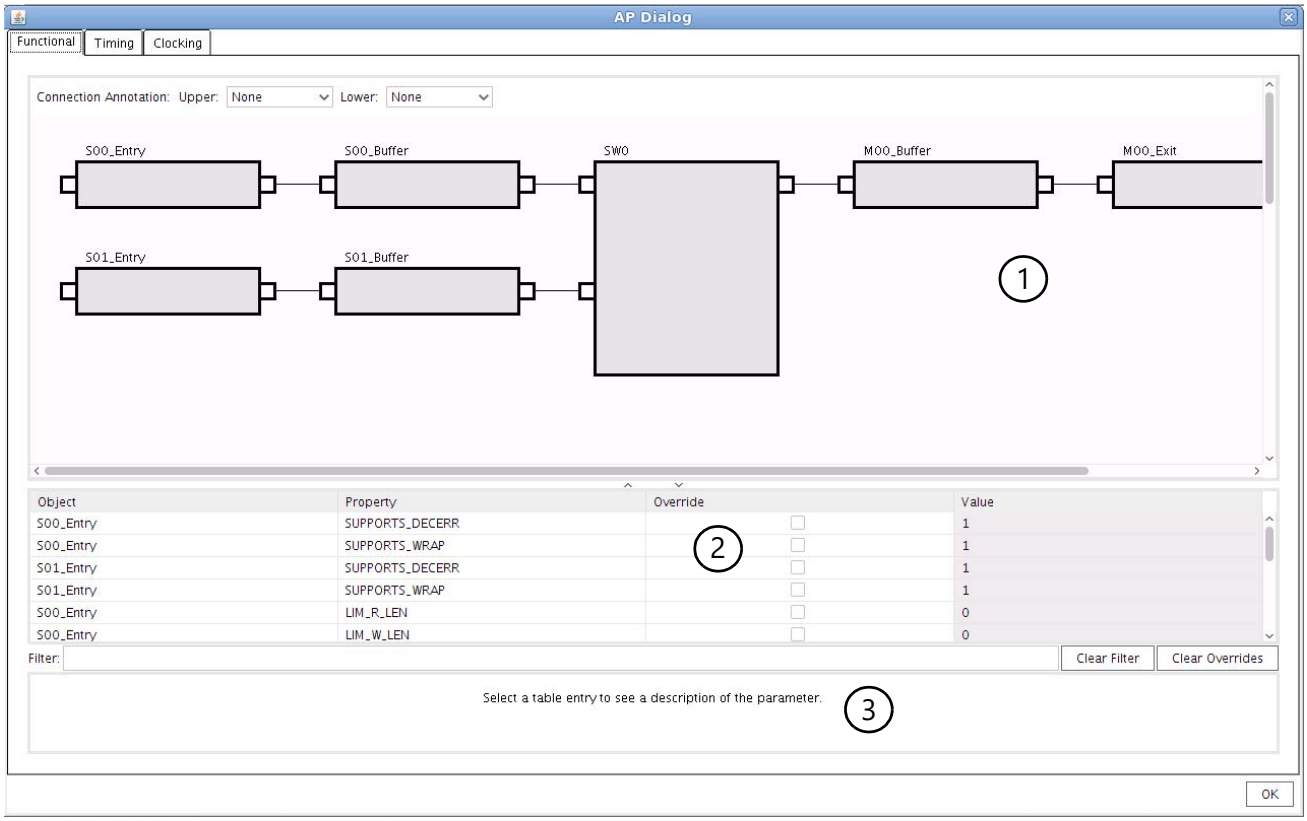


*Figure 3-2:* **Advanced Properties main dialog**

Each tab consists of three elements:

- A schematic which shows a logical view of the SmartConnect's internal topology and any currently applied advanced property overrides (1).
- A table containing any advanced properties within the view (2).
- A documentation pane which displays an overview of any parameter selected in the table (3).

The table listing of available advanced properties can be filtered by object or property name (clicking on an object in the schematic will automatically filter list to properties that apply to the selected object).

Objects in the advanced property schematic view (1) represent entry pipeline logic (Snn_Entry), buffers (Snn_Buffer / Mnn_Buffer), switchboard logic (SW0), and exit pipeline logic (Mnn_Exit).

Initially, the table of advanced properties (2) shows the value of each property as assigned by SmartConnect's internal automation. You can override the automated value of a property by setting the **Override** checkbox, clicking on the **Value** cell and selecting a new value from the drop-down menu which lists alternate values for the property.

## *Functional View - Advanced Properties*

### Snn_Entry Advanced Properties

The following advanced properties can be specified for each S_AXI port enabled on the SmartConnect and connected to an endpoint master IP. Properties that apply specifically to the read or write path are only applicable if the connected endpoint master supports the corresponding access mode. For example, properties specific to the read path are not applicable or available if the connected endpoint master is WRITE_ONLY.

*Table 3-2:* **Snn_Entry Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| SUPPORTS_WRAP | **Supports Wrapped Bursts.**<br>Configures the SmartConnect entry logic to either accept or reject WRAP transactions from the connected endpoint AXI master. In most cases, AXI SmartConnect will automate the value of this parameter based on the interface metadata of the connected master. This parameter can be used to override the automation if, for example, the attached master has not been packaged with sufficient metadata to enable the automation of WRAP support. | **Type**: Integer<br>**Values**: 0,1<br>**Dependencies**:<br>Not applicable if the connected master's protocol is AXI4LITE. |
| SUPPORTS_DECERR | **Supports Decode Error Generation**<br>Allows some transaction checking features of SmartConnect to be disabled to save area and reduce latency. In most cases, SmartConnect automates this property based on the interface metadata of the connected endpoint master. This parameter can be used to override the automation if the packaging of the attached master does not lead to the desired automation outcome.<br>When the parameter is set to 0, address range checking and WRAP/FIXED burst detection will be disabled.<br>Address range decoding and checking will also be disabled and all transactions will be routed to the first mapped address segment. One cycle of internal pipelining will also be removed.<br>**IMPORTANT:** *Setting this parameter to 0 will prevent the SmartConnect from being able to access multiple MI interfaces and prevents it from generating decode errors if the master issues invalid transactions.* | **Type**: Integer<br>**Values**: 0, 1<br>**Default**:1, automated |

*Table 3-2:* **Snn_Entry Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| LIM_R_LEN<br>LIM_W_LEN | **Limit Read Length and Limit Write Length**<br>Enables splitting of read or write transactions that exceed the configured length into a series of transactions equal to or less than the configured length.<br><br>Splitting long write transactions from the master allows you to influence the arbitration behavior of the system or to match a slave's preferred transaction length to allow more efficient processing within the endpoint slave.<br>You should avoid reducing the burst lengths to the point that a later width conversion would not have sufficient data beats to fill at least one entire beat of a transaction. | **Type**: Integer<br>**Values**: 0, 1, 2, 4, 8, 16, 32, 64, 128, 256<br>**Default**: 0<br>**Dependencies**:<br>If the interface protocol is AXI3, the range is reduced to powers of 2 up to a maximum of 16.<br>Not applicable if the interface protocol is AXI4LITE.<br>A value of 0 disables the feature. |
| PKT_W_THR | **Packet Write Threshold**<br>Specifies the number of write data beats that should be accumulated in the write data FIFO before releasing the corresponding write command. The write command is otherwise released if the last data beat (wlast) is received prior to the specified threshold.<br>If a master cannot issue write data beats to the SmartConnect in a continuous manner, you can use this feature to accumulate a continuous sequence of data beats for release to the slave. This can lead to more efficient arbitration and data transfer in the system. | **Type:** Integer<br>**Values:** 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048.<br>**Default**: 0<br>**Dependencies**:<br>The value of the parameter cannot exceed the write payload buffer depth.<br>The range of the parameter is limited according to the maximum burst length of the interface's AXI protocol.<br>A value of 0 disables the feature. |
| PKT_R_THR | **Packet Read Threshold**<br>Specifies the maximum occupancy of the read data buffer. If a read command received from the master would exceed the specified value, the command will not be released to the slave until sufficient capacity becomes available in the data buffer for the command.<br><br>The value of the packet threshold cannot exceed the configured read buffer depth for the port.<br><br>If the master attached to the SmartConnect port does not self-limit the number of read commands it issues, and cannot guarantee all the read data is consumed at the rate it is returned, it is possible that the port's read data buffer will fill, leading to a data transfer stall affecting other masters.<br>In this scenario, you can enable the read packet threshold to prevent the master from over-filling its read data buffer and causing such stalls. If the feature is enabled, the read threshold is typically set to the same size as the read data buffer. | **Type**: Integer<br>**Values**: 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048<br>**Default**: 0<br>**Dependencies**:<br>The value of the parameter cannot exceed the read payload buffer depth.<br>A value of 0 disables the feature. |

*Table 3-3:* **Snn_Buffer and Mnn_Buffer Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| AR_SIZE<br>AW_SIZE<br>B_SIZE | **Command and write response buffer size**<br>Specifies the depth of the buffer of the associated channel. | **Type**: Integer<br>**Values**: 0, 1, 2, 4, 8, 16, 32<br>**Default**: 32<br>**Dependencies**: N/A |
| W_SIZE<br>R_SIZE | **Data Channel Buffer Size**<br>Specifies the depth of the data channel buffers. | **Type**: Integer<br>**Values**: 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096<br>**Default**: 32 |

*Table 3-4:* **Switchboard Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| DATA_WIDTH | **Switchboard Data Width**<br>Specifies the data width in use by the switchboards in the center of the SmartConnect instance. | **Type**: Integer<br>**Values**: 32, 64, 128, 256, 512, 1024<br>**Default**: Automated |

## *Timing View - Advanced Properties*

*Table 3-5:* **Snn_Entry Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| MMU_REGSLICE | **MMU Register Slice**<br>Enables or disables a register slice at the outer boundary of the SmartConnect port.<br>The exterior register slice is enabled by default but you can disable it via this advanced property to reduce area and latency (if it is known that the IP attached to the SmartConnect port is suitably pipelined). | **Type:** Integer<br>**Values**: 0,1<br>**Default**: 1 |
| TR_REGSLICE | **Transaction Regulator Register Slice**<br>Enables or disables a register slice on the interior pathway of the port's entry pipeline.<br>The interior register slice is enabled by default but you can disable it to reduce area and latency (if it is known that no adverse effect on timing closure will occur). | **Type:** Integer<br>**Values**: 0,1<br>**Default**: 1 |

*Table 3-6:* **Snn_Buffer Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| AR_SLR_PIPE<br>AW_SLR_PIPE<br>R_SLR_PIPE<br>W_SLR_PIPE<br>B_SLR_PIPE | **Per-channel SLR Pipeline Control**<br>Specifies number of pipeline stages that improve SLR-crossing in the corresponding channel's data buffer. See Constraining the Core for information on associated placement constraints to ensure the SLR crossing pipeline flops are placed correctly. | **Type**: Integer<br>**Values**: 0..3<br>**Default**: 0 |
| AR_M_PIPE<br>AW_M_PIPE<br>W_M_PIPE | **Per-channel Payload Pipeline Control**<br>Enables or disables pipeline stages for channel payloads between the channel's Snn_Buffer and Switchboard. The interior register slice is enabled by default but you can disable it to reduce area and latency (if it is known that no adverse effect on timing closure will occur). | **Type**: Integer<br>**Values**: 0..3<br>**Default**: 0 |
| AR_M_SEND_PIPE<br>AW_M_SEND_PIPE<br>W_M_SEND_PIPE | **Per-channel Handshake Pipeline Control**<br>Enables or disables pipeline stages for channel handshakes between the Snn_Buffer and SmartConnect's channel switchboards. | **Type**: Integer:<br>**Values**: 0,1<br>**Default**: 1 |
| AR_SYNC_STAGES<br>AW_SYNC_STAGES<br>R_SYNC_STAGES<br>W_SYNC_STAGES<br>B_SYNC_STAGES | **Per-channel Asynchronous Clock Crossing Stages**<br>Specifies the number of synchronization stages used in asynchronous clock domain conversion. | **Type**: Integer:<br>**Values**: 2..8<br>**Default**: 3 |

*Table 3-7:* **Mnn_Buffer Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| AR_SLR_PIPE<br>AW_SLR_PIPE<br>R_SLR_PIPE<br>W_SLR_PIPE<br>B_SLR_PIPE | **Per-channel SLR Pipeline Control**<br>Specifies number of pipeline stages that improve SLR-crossing in the corresponding channel's data buffer. See Constraining the Core for information on associated placement constraints to ensure the SLR crossing pipeline flops are placed correctly. | **Type**:<br>Integer: 0..3<br>**Default**: 0 |
| R_M_PIPE<br>B_M_PIPE | **Per-channel Payload Pipeline Control**<br>Enables or disables pipeline stages for channel payloads between the channel's Snn_Buffer and Switchboard<br>The interior register slice is enabled by default but you can disable it to reduce area and latency (if it is known that no adverse effect on timing closure will occur). | **Type**:<br>Integer: 0..3<br>**Default**: 0 |
| R_M_SEND_PIPE<br>B_M_SEND_PIPE | **Per-channel Handshake Pipeline Control**<br>Enables or disables pipeline stages for channel handshakes between the Snn_Buffer and SmartConnect's channel switchboards. | **Type**:<br>Integer: 0, 1<br>**Default**: 1 |
| AR_SYNC_STAGES<br>AW_SYNC_STAGES<br>R_SYNC_STAGES<br>W_SYNC_STAGES<br>B_SYNC_STAGES | **Per-channel Asynchronous Clock Crossing Stages**<br>Specifies the number of synchronization stages used in asynchronous clock domain conversion. | **Type**: Integer:<br>**Values**: 2..8<br>**Default**: 3 |

*Table 3-8:* **Switchboard (SW0) Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| AR_M_PIPE<br>AW_M_PIPE<br>R_M_PIPE<br>W_M_PIPE<br>B_M_PIPE | **Per-channel Payload Pipeline Control**<br>Enables pipeline stages on the egress of the given channel switchboard. | **Type**:<br>Integer: 0..3<br>**Default**: 1 |
| AR_S_PIPE<br>AW_S_PIPE<br>R_S_PIPE<br>W_S_PIPE<br>B_S_PIPE | **Per-channel Payload Pipeline Control**<br>Enables pipeline stages on the ingress of the given channel switchboard. | **Type**:<br>Integer: 0..3<br>**Default**: 0 |

*Table 3-9:* **Mnn_Exit Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| REGSLICE | **Exit Register Slice**<br>Enables or disables a register slice at the outer boundary of the SmartConnect port.<br>The exterior register slice is enabled by default but you can disable it via this advanced property to reduce area and latency (if it is known that the IP attached to the SmartConnect port is suitably pipelined). | **Type**:<br>Integer: 0, 1<br>**Default**: 1 |

## *Clocking View - Advanced Properties*

*Table 3-10:* **Switchboard (SW0) Clocking Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| ASSOCIATED_CLK | **Switchboard Clock Selection**<br>Specifies which clock input of the SmartConnect is used as the clock input of internal switch IPs.<br>The clock source connected to the aclk pin is the default clock source of the switch IPs, but if multiple clock inputs are available, you can select a different clock source for the switches. | **Type**: String<br>**Value Range**: aclk, aclk[1..32] |

*Table 3-11:* **Mnn_Exit Clocking Advanced Properties**

| Name | Description | Format/Range/Dependencies |
|---|---|---|
| ASSOCIATED_CLK | **Cascading Master Interface Clock Selection**<br>Specifies which clock input of the SmartConnect is used as the clock source for the Mnn_AXI port.<br>This parameter is only available for Mnn_AXI ports that connect to another SmartConnect instance.<br>SmartConnect's internal automation will select a clock from the set of clock sources shared between the cascading SmartConnect instances. | **Type**: String<br>**Value Range**: aclk, aclk[1..32].<br>Only clock pins that connect to clock sources common to the two SmartConnect instances are valid. |

# Clocking

By default, all interfaces on the SmartConnect operate in the same clock domain, and the clock is received on the `aclk` input pin of the IP. The IP can be configured to support multiple clock domains by enabling a corresponding number of additional clock pins (`aclk1`... `aclkn`). The tools automatically determine which clock domain each interface operates in by tracing the interface connections in the system and identifying the clock source used to synchronize the AXI interface in the connected master or slave device. The SmartConnect is then automatically configured to apply each of its clock inputs to each of the SIs and MIs that are synchronized by the same clock source. An error will occur if any SI or MI interface connection is synchronized by a clock source not connected to one of the clock inputs on the SmartConnect.

It is not necessary to connect any one clock source to more than one clock input pin of SmartConnect.

When information is exchanged along any AXI channel between interfaces belonging to different clock domains, clock conversion logic is automatically inserted along the pathway.

When the tools determine that the relationship between the clock domains is an integer ratio (faster or slower) within the range 1:16 to 16:1, and that the clocks are derived from the same clock source, the tools automatically configure the clock converter to perform synchronous conversion; otherwise, the clock converter is configured in asynchronous mode.

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

All designs (especially those containing clock conversions) should have all their clocks defined in a system-level XDC file using the `create_clock` command. These clock constraints are sometimes automatically generated when targeting a development board or when including a Clock Wizard IP in the system. Each SmartConnect IP instance that implements an asynchronous clock-domain-crossing (CDC) attempts to generate IP-level timing constraints based on the clocks defined for the design. The purpose of the IP-generated constraints is to prevent timing violations during static timing analysis for CDCs which get resynchronized by the IP core. If you implement a design containing asynchronous clock conversions without defining the clocks at the system level, you may receive warnings informing you that clocks cannot be found, and that the generated `set_max_delay` constraints cannot be applied. These warnings have no impact on the correct functional implementation of the design, and they will resolve when the required system-level clock definitions are provided.

Clock converters always introduce latency. Asynchronous conversion incurs more latency and uses more logic resources than synchronous conversion.

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

# Resets

The SmartConnect IP has one active-low reset input (`aresetn`). The reset input is internally resynchronized to each of the clock domains connected to the IP.

If no *soft* reset is required beyond power-on, the `aresetn` pin may be disabled (HAS_ARESETN=0). All internal state logic will automatically be initialized during power-up.

The SmartConnect core deasserts all `valid` and `ready` outputs during the power-on reset cycle and shortly after `aresetn` is sampled active, and for the duration of the `aresetn` pulse.

AXI protocol requires that all connected masters also de-assert all `valid` outputs during reset (until after `aresetn` is sampled inactive). Slaves must not assert response-channel `valid` outputs until after they receive a command from a master. It is also strongly recommended that slave IP de-assert their `ready` outputs until after reset. This avoids inadvertently signaling a transfer completion in case a connected IP recovers from reset during an earlier cycle and asserts its `valid`.

There is no requirement that the assertion or de-assertion of `aresetn` be observed during the same cycle or in any relative order among SmartConnect and its connected masters and slaves. It is, however, required that the cycles during which reset is applied to SmartConnect and all its connected masters and slaves overlap.

SmartConnect does not support independent reset domains. If any master or slave device connected to SmartConnect is reset, then all connected devices must be reset concurrently.

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

# Design Flow Steps

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

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

## Customizing and Generating the Core

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

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

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

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

## Output Generation

The AXI4 SmartConnect deliverables are organized in the directory:

<project name>/<project name>.srcs/sources_1/ip/<component name> and is designated as the <ip source dir>.

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

# Constraining the Core

In general, the only constraints required when deploying the SmartConnect core is the specification of a clock period constraint for each of the clock signals connected to all `aclk` inputs of the core. Placement constraints may also be needed when the SmartConnect is required to span across Super Logic Regions (SLRs).

## Required Constraints

When the SmartConnect is configured to use the SLR crossing pipelines, and the given SI or MI interface will cross an SLR boundary, placement constraints can improve timing and placement of these pipelining FFs.

**XDC Constraints for SLR crossing at the SI side of SmartConnect.**

The example below is for interface `s00`, but can be applied to any `s<nn>` interface that crosses SLRs.

```
# SLR of Endpoint Master IP
create_pblock pblock_axi_master_on_si
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_ar_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_aw_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_b_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_r_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_w_node/inst/inst_si_handler]] -quiet
resize_pblock [get_pblocks pblock_smartconnect_switchboard] -add
{CLOCKREGION_X0Y5:CLOCKREGION_X5Y9} # clock region defining SLR depends on device part

# SLR of SmartConnect core Switchboard
create_pblock pblock_smartconnect_switchboard
add_cells_to_pblock [get_pblocks pblock_axi_master_on_si] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_ar_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_axi_master_on_si] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_aw_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_axi_master_on_si] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_b_node/inst/inst_si_handler]] -quiet
```

```
add_cells_to_pblock [get_pblocks pblock_axi_master_on_si] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_r_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_axi_master_on_si] [get_cells [list
<netlist_path_SC_component>/inst/s00_nodes/s00_w_node/inst/inst_mi_handler]] -quiet
resize_pblock [get_pblocks pblock_axi_master_on_si] -add
{CLOCKREGION_X0Y0:CLOCKREGION_X5Y4} # clock region defining SLR depends on device part
```

**XDC Constraints for SLR crossing at the MI side of SmartConnect.**

Example below is for interface `m00` but can be applied to any `m<nn>` interface that crosses SLRs.

```
# SLR of SmartConnect core Switchboard
create_pblock pblock_smartconnect_switchboard
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_ar_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_aw_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_b_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_r_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_smartconnect_switchboard] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_w_node/inst/inst_si_handler]] -quiet
resize_pblock [get_pblocks pblock_smartconnect_switchboard] -add
{CLOCKREGION_X0Y5:CLOCKREGION_X5Y9} # clock region defining SLR depends on device part

# SLR of Endpoint Slave IP
create_pblock pblock_axi_slave_on_mi
add_cells_to_pblock [get_pblocks pblock_axi_slave_on_mi] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_ar_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_axi_slave_on_mi] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_aw_node/inst/inst_mi_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_ axi_slave_on_mi] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_b_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_ axi_slave_on_mi] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_r_node/inst/inst_si_handler]] -quiet
add_cells_to_pblock [get_pblocks pblock_ axi_slave_on_mi] [get_cells [list
<netlist_path_SC_component>/inst/m00_nodes/m00_w_node/inst/inst_mi_handler]] -quiet
resize_pblock [get_pblocks pblock_ axi_slave_on_mi] -add
{CLOCKREGION_X0Y0:CLOCKREGION_X5Y4} # clock region defining SLR depends on device part
```

### Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

### Clock Frequencies

This section is not applicable for this IP core.

### Clock Management

This section is not applicable for this IP core.

### Clock Placement

This section is not applicable for this IP core.

### Banking

This section is not applicable for this IP core.

### Transceiver Placement

This section is not applicable for this IP core.

### I/O Standard and Placement

This section is not applicable for this IP core.

## Simulation

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

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

# Example Design

The AXI SmartConnect core does not provide an example design.

# Upgrading

This appendix provides a feature comparison of the AXI SmartConnect v1.0 core with the AXI Interconnect v2.1 core (as used in the Vivado® Design Suite). It also contains migration guidance for upgrading from the AXI Interconnect v2.1 core.

The AXI SmartConnect v1.0 core is intended as a drop-in replacement for the AXI Interconnect v2.1 core in all applications.

There is no migration automation available to convert a design using the AXI Interconnect v2.1 core.

## Upgrading Guidelines

This section provides information on upgrading from the AXI Interconnect v2.1 core to the AXI SmartConnect v1.0 core.

### Upgrading Exceptions

At this time, if you have a design containing AXI Interconnect v2 or any of its stand-alone converter cores, Xilinx recommends that you continue using them under the following situations. For all other cases, you may upgrade to SmartConnect v1.0, but it is not required:

- If your design uses an AXI Data Width Converter, AXI Clock Converter, AXI Protocol Converter, AXI Data FIFO or AXI Register Slice core, and it is not directly connected to an interface of an AXI Interconnect v2 core, then continue to use it. Xilinx does not recommend using SmartConnect in a 1:1 configuration to perform those in-line functions at this time.
- If your design uses the AXI Interconnect v2 core in the *Area Strategy* mode, and any of your connected endpoint masters or slaves are not AXI4-Lite, then continue using it at this time. SmartConnect provides an area-optimized mode only when all SI and MI are connected to AXI4-Lite endpoints.
- If your design uses the AXI Interconnect v2 core in the *Area Strategy* mode and it is performing width conversion or clock conversion, then continue using it at this time. SmartConnect provides an area-optimized mode only when there are no clock or width conversions among any of the endpoints.

- If your AXI Interconnect depends on fixed priority arbitration to strictly lock out lower-priority masters during higher-priority requests, then continue using AXI Interconnect, at least among the masters that must adhere to fixed priority. If your AXI Interconnect uses fixed priority to lessen the impact of lower-priority masters on high-performance traffic, then you may still upgrade to SmartConnect by following the applicable guidelines in the Upgrading Instructions section.

## Upgrading Instructions

Use the following steps to upgrade from AXI Interconnect v2 to SmartConnect v1.0:

1. Configure the number of SIs and MIs to match AXI Interconnect being replaced.
2. Set the number of clock inputs to the number of unique clock signals that were connected to AXI Interconnect. Do not include additional clock input pins to which the same clock signal(s) were connected.
3. If your AXI Interconnect connected to endpoints that were all AXI4-Lite, then SmartConnect will automatically use its area-optimized mode, provided your AXI Interconnect did not perform any width or clock conversions. There is no user setting required for this feature.
4. You cannot specify register pipelining and/or FIFO buffering at this time. You can set them using the SmartConnect Advance Properties only after connecting the SmartConnect instance to all endpoint logic and running Validation. Otherwise, wait until after you review your implementation results using the default settings. The deployment of pipelines and FIFO buffers in SmartConnect is fundamentally different from that in AXI Interconnect.
5. After customizing, reconnect all SI and MI ports to their AXI endpoints.
6. Reconnect the SmartConnect `aclk` and `aresetn` inputs to the same signals as in AXI Interconnect.
7. Connect all remaining clock signals (if any) to any of the additional clock inputs. (SmartConnect automatically detects the clock domain associated with each SI and MI interface connection.)
8. Do not connect any other reset signals that were previously connected to AXI Interconnect. SmartConnect will internally re-synchronize to each SI/MI clock domain.
9. If your AXI Interconnect used fixed priority to lessen the impact of lower-priority masters on high-performance traffic, then you could likely achieve similar results by specifying a low positive value for Limit Read/Write Length (Advanced Properties) for lower-priority SI interfaces. This forces any bursts longer than the specified value to get split into a series of shorter bursts, effectively lowering the transaction's service rate during SmartConnect round-robin arbitration.
10. If any of the AXI slave devices in your application relies on reading the ID value issued by AXI Interconnect to positively identify the master issuing the transaction, this capability is no longer supported using the ID signal when using SmartConnect. Instead, Xilinx

recommends using the `awuser/awuser` signals to transmit source identification constants to endpoint slaves. SmartConnect automatically compresses or eliminates AXI ID signals when not needed to control response reordering among multi-threaded traffic.

11. If your AXI Interconnect used packet-mode FIFOs to prevent data channel stalling due to buffer overrun/underrun, refer to the Packet Write/Read Threshold (Advanced Properties) for the corresponding SmartConnect functionality.

# Feature Comparison

This section highlights the feature differences between AXI SmartConnect v1.0 and AXI Interconnect v2.1.

- The individual conversion and storage modules used in SmartConnect are not available as stand-alone IP. Visibility of internal logic blocks inside SmartConnect is not supported.
- SmartConnect provides parallel destination-side arbitration on all AXI channels (Multiple-Address Multiple-Data [MAMD] topology instead of Single-Address Multiple-Data [SAMD].)
- SmartConnect supports back-to-back arbitration and propagation on AW and AR channels (supporting high-bandwidth for single-beat transactions).
- Fixed priority arbitration is no longer supported by SmartConnect, only round-robin.
- SmartConnect propagates ID signals (multi-threaded traffic, if enabled) through all data-width conversions, protocol conversions (except AXI4-Lite) and transaction splitting.
- SmartConnect propagates User-defined signals on all channels through all data-width conversions, protocol conversions (except AXI4-Lite) and transaction splitting.
- SmartConnect remaps Wide ID signals at the SI to shorter IDs at the MI.
- Automatic single-ordering mode eliminates ID tracking overhead when multi-threaded traffic is not required.
- One clock input per clock domain instead of one clock input per SI/MI interface.
- One reset input for the whole IP instead of one reset per SI/MI interface.
- SmartConnect performs address decode and error detection only at the first SI interface, not repeated by each cascaded SmartConnect instance.
- SmartConnect does not support propagation of WRAP type bursts; WRAP bursts are automatically converted to INCR type bursts.

- FIXED burst type transactions are no longer supported.
- SmartConnect does not support propagation of narrow bursts (awsize/arsize less than the full interface data width); all bursts are fully packed, regardless of the "modifiable" bit in `awcache/arcache`.
- The `awregion/arregion` signals are no longer generated on the MI.
- SmartConnect performs deadlock avoidance checking only at the first SI interface, not repeated by each cascaded SmartConnect instance.
- At this time, SmartConnect provides an area-optimized mode only when all SIs and MIs are connected to AXI4-Lite endpoints and there are no clock or width conversions among any of the endpoints. For any other configurations where an area-optimized mode is preferred, continue to use axi_interconnect_v2_1 in area strategy mode.

# Debugging

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

## AXI Protocol Violations

When designing with *custom* or *non-production* IP, it is common to encounter system malfunctions caused by AXI protocol violations. Xilinx AXI IP cores, including SmartConnect, do not contain any logic to guard against AXI protocol violations incurred by IP cores to which they are connected.

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

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

# Finding Help on Xilinx.com

To help in the design and debug process when using the AXI SmartConnect Core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

## Documentation

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

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

## Answer Records

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

You can locate Answer Records for this core by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

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

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

**Master Answer Record for the AXI SmartConnect Core**

AR: 66780

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

## Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

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

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

# Debug Tools

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

## Vivado Design Suite Debug Feature

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

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

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

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 8].

## Interface Debug

The core has no accessible registers.

# Definitions, Acronyms, and Abbreviations

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

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

| Item | Description |
|---|---|
| AXI | The generic term for all implemented AXI protocol interfaces. |
| master device or connected master | An IP core or device (or one of multiple interfaces on an IP core) that generates AXI transactions out from the IP core onto the wires connecting to a slave device. |
| slave device or connected slave | An IP core or device (or one of multiple interfaces on an IP core) that receives and responds to AXI transactions coming in to the IP core from the wires connecting to a master device. |
| master interface (generic) | An interface of an IP core or module that generates out-bound AXI transactions and thus is the initiator (source) of an AXI transfer. On AXI master interfaces, AWVALID, ARVALID, and WVALID are outputs, and RVALID and BVALID are inputs. |
| slave interface (generic) | An interface of an IP core or module that receives in-bound AXI transactions and becomes the target (destination) of an AXI transfer. On AXI slave interfaces, AWVALID, ARVALID, and WVALID are inputs, and RVALID and BVALID are outputs. |
| SI-side | A module interface closer to the SI side of the AXI SmartConnect core. |
| MI-side | A module interface closer to the MI side of the AXI SmartConnect core. |
| upsizer | Data width conversion function in which the datapath width gets wider when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction). |
| downsizer | Data width conversion function in which the datapath width gets narrower when moving in the direction from the SI-side toward the MI-side (regardless of write/read direction). |

# Additional Resources and Legal Notices

## Xilinx Resources

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

## References

To search for Xilinx documentation, go to Xilinx Support.


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

# Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 12/20/2017 | 1.0 | • Added Migration instructions.<br>• Added Pipelining/Constraints content. |
| 10/04/2017 | 1.0 | • Tables 3-6 and 3-7: Added Per-channel Asynchronous Clock Crossing Stages.<br>• Updated description for downsizing to an AXI4-Lite slave. |
| 04/05/2017 | 1.0 | • ADVANCED_PROPERTIES added to Design Parameters.<br>• Protocol Conversion section: Additional information provided on downsizing to AXI4-Lite slave. |
| 10/05/2016 | 1.0 | • Support for FIXED Burst type transactions removed.<br>• HAS_ARESETN added to Design Parameters. |
| 05/10/2016 | 1.0 | Initial Xilinx release as product guide release. |

# Please Read: Important Legal Notices