AXI Reference Guide

UG761 (v14.3) November 15, 2012

XILINX®

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
04/24/2012	14.1	Reorganized How AXI Works in Chapter 1. Added AXI4-Stream IP Interoperability in Chapter 1. Reworded AXI4-Stream Adoption and Support in Chapter 3. Added upsizer/downsizer content to Real Scalar Data Example in Chapter 3. Modified: • All instances of "Video over AXI4-Stream" to "AXI4-Stream Video Protocol." • Figure 3-15 • Table 3-6 Removed redundant figure in Chapter 3. Added: • Chapter 6, "AXI4-Stream IP Interoperability: Tips and Hints."
	14.2	No Changes.
11/15/2012	14.3	Changes: • Updated document format. Additions: • Added reference to 7 series devices. • Added overview of Xilinx AXI4-Stream Interconnect Core IP in Chapter 2. • Added Packet Mode feature in Independently Configure Converter Banks in Chapter 5. • Added content to Cascading Interconnects in Chapter 5.

XILINX®

Table of Contents

XILINX®

Chapter 1

Introducing AXI for Xilinx System Development

Introduction

Xilinx® adopted the Advanced eXtensible Interface (AXI) protocol for Intellectual Property (IP) cores beginning with the Spartan®-6 and Virtex®-6 devices. Xilinx continues to use of the AXI protocol for IP targeting the 7 series, and the Zynq™-7000 All Programmable SoC devices.

This document is intended to:

- Introduce key concepts of the AXI protocol
- Give an overview of what Xilinx tools you can use to create AXI-based IP
- Explain what features of AXI Xilinx has adopted
- Provide guidance on how to migrate your existing design to AXI

Note: This document is not intended to replace the Advanced Microcontroller Bus Architecture (AMBA®) ARM® AXI4 specifications. Before beginning an AXI design, you need to download, read, and understand the *ARM AMBA AXI Protocol v2.0 Specification* [Ref 1], along with the *AMBA4 AXI4-Stream Protocol v1.0* [Ref 2].

These are the steps to download the specifications; you might need to fill out a brief registration before downloading the documents:

1. Go to www.amba.com
2. Click **Download Specifications**.
3. In the **Contents** pane on the left, click **AMBA > AMBA Specifications >AMBA4**.
4. Download both the *ABMA AXI4-Stream Protocol Specification* and *AMBA AXI Protocol Specification v2*.0.

What is AXI?

AXI is part of ARM AMBA, a family of micro controller buses first introduced in 1996. The first version of AXI was first included in AMBA 3.0, released in 2003. AMBA 4.0, released in 2010, includes the second version of AXI, AXI4.

There are three types of AXI4 interfaces:

- AXI4—For high-performance memory-mapped requirements.
- AXI4-Lite—For simple, low-throughput memory-mapped communication (for example, to and from control and status registers).
- AXI4-Stream—For high-speed streaming data.

Xilinx introduced these interfaces in the ISE® Design Suite, release 12.3.

Summary of AXI4 Benefits

AXI4 provides improvements and enhancements to the Xilinx product offering across the board, providing benefits to *Productivity, Flexibility,* and *Availability*:

- Productivity—By standardizing on the AXI interface, developers need to learn only a single protocol for IP.
- Flexibility—Providing the right protocol for the application:
  - AXI4 is for memory mapped interfaces and allows burst of up to 256 data transfer cycles with just a single address phase.
  - AXI4-Lite is a light-weight, single transaction memory mapped interface. It has a small logic footprint and is a simple interface to work with both in design and usage.
  - AXI4-Stream removes the requirement for an address phase altogether and allows unlimited data burst size. AXI4-Stream interfaces and transfers do not have address phases and are therefore not considered to be memory-mapped.
- Availability—By moving to an industry-standard, you have access not only to the Xilinx IP catalog, but also to a worldwide community of ARM Partners.
  - Many IP providers support the AXI protocol.
  - A robust collection of third-party AXI tool vendors is available that provide a variety of verification, system development, and performance characterization tools. As you begin developing higher performance AXI-based systems, the availability of these tools is essential.

How AXI Works

This section provides a brief overview of how the AXI interface works. The Introduction, page 3, provides the procedure for obtaining the ARM specification. Consult those specifications for the complete details on AXI operation.

The AXI specifications describe an interface between a single AXI master and a single AXI slave, representing IP cores that exchange information with each other. Memory mapped AXI masters and slaves can be connected together using a structure called an *Interconnect* block. The Xilinx AXI Interconnect IP contains AXI-compliant master and slave interfaces, and can be used to route transactions between one or more AXI masters and slaves. The AXI Interconnect IP is described in Xilinx AXI Interconnect Core IP, page 17.

Both AXI4 and AXI4-Lite interfaces consist of five different channels:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

Data can move in both directions between the master and slave simultaneously, and data transfer sizes can vary. The limit in AXI4 is a burst transaction of up to 256 data transfers. AXI4-Lite allows only 1 data transfer per transaction.

Figure 1-1 shows how an AXI4 read transaction uses the read address and read data channels:

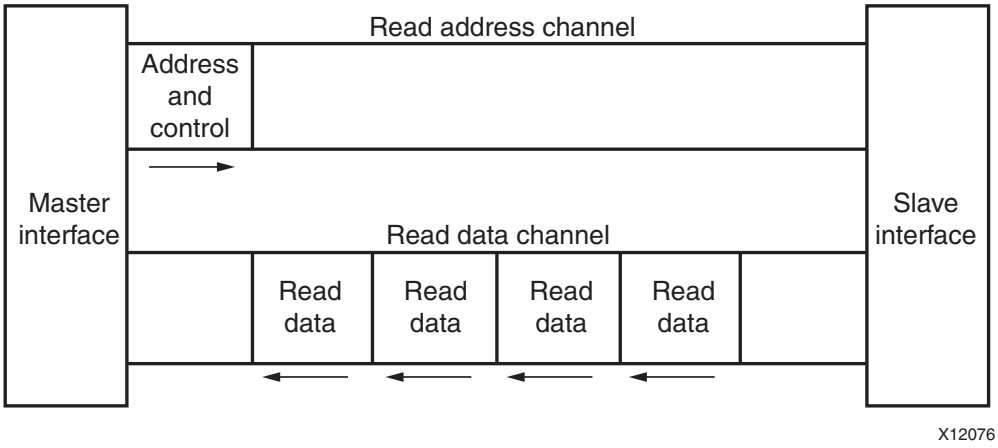


Figure 1-1: **Channel Architecture of Reads**

Figure 1-2 shows how a write transaction uses the write address, write data, and write response channels.

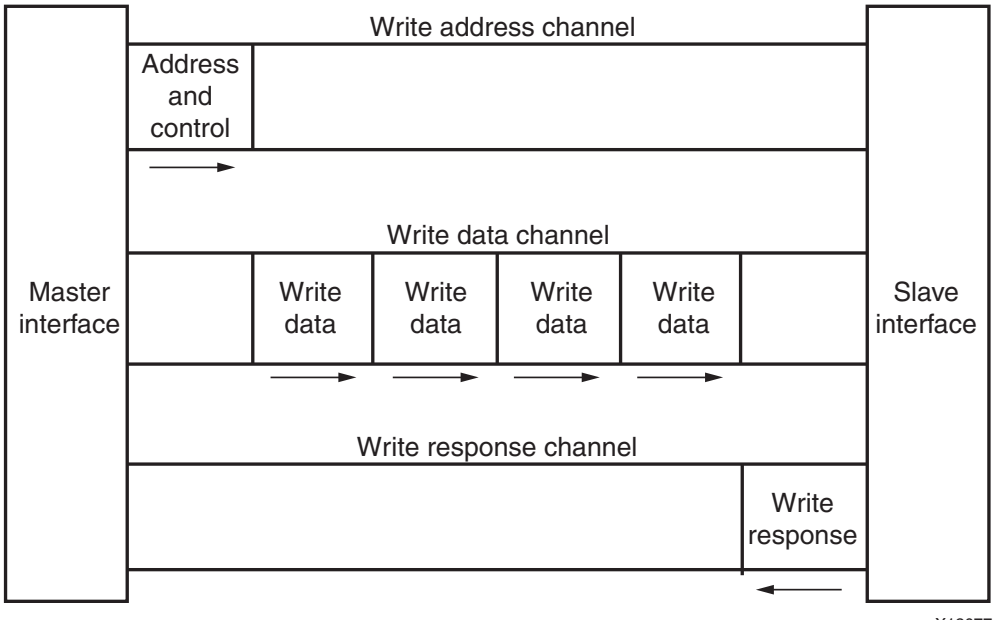


Figure 1-2: **Channel Architecture of writes**

As shown in the preceding figures, AXI4 provides separate data and address connections for reads and writes, which allows simultaneous, bidirectional data transfer. AXI4 requires a single address and then bursts up to 256 words of data. The AXI4 protocol describes a variety of options that allow AXI4-compliant systems to achieve very high data throughput. Some of these features, in addition to bursting, are: data upsizing and downsizing, multiple outstanding addresses, and out-of-order transaction processing.

At a hardware level, AXI4 allows a different clock for each AXI master-slave pair. In addition, the AXI protocol allows the insertion of register slices (often called pipeline stages) to aid in timing closure.

AXI4-Lite is similar to AXI4 with some exceptions, the most notable of which is that bursting, is not supported. The AXI4-Lite chapter of the *ARM AMBA AXI Protocol v2.0 Specification* describes the AXI4-Lite protocol in more detail.

The AXI4-Stream protocol defines a single channel for transmission of streaming data. The AXI4-Stream channel is modeled after the write data channel of the AXI4. Unlike AXI4, AXI4-Stream interfaces can burst an unlimited amount of data. There are additional, optional capabilities described in the *AMBA4 AXI4-Stream Protocol v1.0* specification [Ref 2]. The specification describes how AXI4-Stream-compliant interfaces can be split, merged, interleaved, upsized, and downsized. Unlike AXI4, AXI4-Stream transfers cannot be reordered.

Memory Mapped Protocols

In memory mapped AXI (AXI3, AXI4, and AXI4-Lite), all transactions involve the concept of a target address within a system memory space and data to be transferred.

Memory mapped systems often provide a more homogeneous way to view the system, because the IPs operate around a defined memory map.

AXI4-Stream Protocol

The AXI4-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI4-Stream acts as a single unidirectional channel for a handshake data flow.

At this lower level of operation (compared to the memory mapped AXI protocol types), the mechanism to move data between IP is defined and efficient, but there is no unifying address context between IP. The AXI4-Stream IP can be better optimized for performance in data flow applications, but also tends to be more specialized around a given application space.

Infrastructure IP

An infrastructure IP is another IP form used to build systems. Infrastructure IP tends to be a generic IP that moves or transforms data around the system using general-purpose AXI4 interfaces and does not interpret data.

Examples of infrastructure IP are:

- Register slices (for pipeling)
- AXI FIFOs (for buffering/clock conversion)
- AXI Interconnect IP (connects memory mapped IP together)
- AXI Direct Memory Access (DMA) engines (memory mapped to stream conversion)

These IP are useful for connecting a number of IP together into a system, but are not generally endpoints for data.

Combining AXI4-Stream and Memory Mapped Protocols

Another approach is to build systems that combine AXI4-Stream and AXI memory mapped IP together. Often a DMA engine can be used to move streams in and out of memory. For example, a processor can work with DMA engines to decode packets or implement a protocol stack on top of the streaming data to build more complex systems where data moves between different application spaces or different IP.

IP Interoperability

The AXI specification provides a framework that defines protocols for moving data between IP using a defined signaling standard. This standard ensures that IP can exchange data with each other and that data can be moved across a system.

AXI IP interoperability affects:

- The IP application space
- How the IP interprets data
- Which AXI interface protocol is used (AXI4, AXI4-Lite, or AXI4-Stream)

The AXI protocol defines how data is exchanged, transferred, and transformed. The AXI protocol also ensures an efficient, flexible, and predictable means for transferring data.

About Data Interpretation

The AXI protocol does not specify or enforce the interpretation of data; therefore, the data contents must be understood, and the different IP must have a compatible interpretation of the data.

For IP such as a general purpose processor with an AXI4 memory mapped interface, there is a great degree of flexibility in how to program a processor to format and interpret data as required by the Endpoint IP.

About IP Compatibility

For more application-specific IP, like an Ethernet MAC (EMAC) or a Video Display IP using AXI4-Stream, the compatibility of the IP is more limited to their respective application spaces. For example, directly connecting an Ethernet MAC to the Video Display IP would not be feasible.

Note: Even though two IP such as EMAC and Video Streaming can theoretically exchange data with each other, they would not function together because the two IP interpret bit fields and data packets in a completely different manner. See the AXI IP specifications at http://www.xilinx.com/support/documentation/axi_ip_documentation.htm, and AXI4-Stream Signals, page 53 for more information.

AXI4-Stream IP Interoperability

Chapter 6, "AXI4-Stream IP Interoperability: Tips and Hints," provides an overview of the main elements and steps for building an AXI4-Stream system with interoperable IP. These key elements include understanding the AXI protocol, learning domain specific usage guidelines, using AXI infrastructure IP as system building blocks, and validating the final result. You can be most effective if you follow these steps:

1. Review the AXI4 documents:
    - AXI4-Stream Protocol
    - AXI Reference Guide (this document): particularly, Chapter 6, "AXI4-Stream IP Interoperability: Tips and Hints."
    - *AXI4-Stream IP Interconnect Product Guide (PG035)* [Ref 14]
2. Understand IP Domains:
    - Review data types and layered protocols in Chapter 6, "AXI4-Stream IP Interoperability: Tips and Hints."
    - Review the list of AXI IP available at: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm
    - Understand the domain-level guidelines.
3. When creating your system:
    - Configure IP to share compatible data types and protocols.
    - Use Infrastructure IP or converters where necessary.
    - Validate the system.

What AXI Protocols Replace

Table 1-1 lists the high-level list of AXI4 features available and what protocols an AXI option replaces.

Table 1-1: **AXI4 Feature Availability and IP Replacement (1)**

Interface	Features	Replaces
AXI4	• Traditional memory mapped address/data interface. • Data burst support.	PLBv3.4/v4.6 OPB NPI XCL
AXI4-Lite	• Traditional memory mapped address/data interface. • Single data cycle only.	PLBv4.6 (singles only) DCR DRP
AXI4-Stream	• Data-only burst.	Local-Link DSP TRN (used in PCIe) FSL XSVI

1. See Chapter 4, "Migrating to Xilinx AXI Protocols," for more information.

Targeted Reference Designs

The other chapters of this document go into more detail about AXI support in Xilinx tools and IP. To assist in the AXI transition, the Spartan-6 and Virtex-6 Targeted Reference Designs, which form the basis of the Xilinx targeted domain platform solution, have been migrated to support AXI. These targeted reference designs provide the ability to investigate AXI usage in the various Xilinx design domains such as Embedded, DSP, and Connectivity. More information on the targeted reference designs is available at http://www.xilinx.com/products/targeted_design_platforms.htm.

AXI Support in Xilinx Tools and IP

AXI Development Support in Xilinx Design Tools

This section describes how Xilinx® tools can be used to build systems of interconnected Xilinx AXI IP (using Xilinx Platform Studio or System Generator for DSP), and deploy individual pieces of AXI IP (using the CORE Generator™ tool).

Using Embedded Development Kit: Embedded and System Edition

Xilinx ISE Design Suite: Embedded Edition and System Edition support the addition of AXI cores into your design through the tools described in the following subsections.

Creating an Initial AXI Embedded System

The following Embedded Development Kit (EDK) tools support the creation and addition of AXI-based IP Cores (pcores).

- **Base System Builder** (BSB) wizard—Creates either AXI or PLBv.46 working embedded designs using any features of a supported development board or using basic functionality common to most embedded systems. After creating a basic system, customization can occur in the main Xilinx Platform Studio (XPS) view and in the ISE® Design Suite. Xilinx recommends using the BSB to start new designs. See the XPS Help for more information.
- **Xilinx Platform Studio** (XPS)—Provides a block-based system assembly tool for connecting blocks of IPs together using many bus interfaces (including AXI) to create embedded systems, with or without processors. XPS provides a graphical interface for connection of processors, peripherals, and bus interfaces.
- **Software Development Toolkit** (SDK)— Is the development environment for application projects. SDK is built with the Eclipse open source standard. For AXI-based embedded systems, hardware platform specifications are exported in an XML format to SDK (XPS-based software development and debugging is not supported.) See the SDK Help for more information.
- Information on EDK is available at: http://www.xilinx.com/support/documentation/dt_edk.htm.

Creating and Importing AXI IP

XPS contains a Create and Import Peripheral (CIP) wizard that automates adding your IP to the IP repository in Platform Studio.

Debugging and Verifying Designs: Using ChipScope in XPS

The ChipScope™ Pro Analyzer AXI monitor core (`chipscope_axi_monitor`) aids in monitoring and debugging Xilinx AXI4 or AXI4-Lite protocol interfaces. This core lets you probe any AXI, memory mapped master or slave bus interface. It is available in XPS.

With this probe you can observe the AXI signals going from the peripheral to the AXI Interconnect core. For example, you can set a monitor on a MicroBlaze processor instruction or data interface to observe all memory transactions going in and out of the processor.

Each monitor core works independently, and allows chaining of trigger outputs to enable taking system level measurements. By using the auxiliary trigger input port and the trigger output of a monitor core you can create multi-level triggering environments to simplify complex system-level measurements.

For example, if you have a master operating at 100MHz and a slave operating at 50MHz, this multi-tiered triggering lets you analyze the transfer of data going from one time domain to the next. Also, with this system-level measurement, you can debug complex multi-time domain system-level issues, and analyze latency bottlenecks in your system.

You can add the `chipscope_axi_monitor` core to your system using the IP Catalog in XPS available under the `/debug` folder as follows:

1. Put the `chipscope_axi_monitor` into your bus interface System Assembly View (SAV).
2. Select the bus you want to probe from the **Bus Name** field.

   After you select the bus, an **M** for monitor displays between your peripheral and the AXI Interconnect core IP.

3. Add a ChipScope `ICON` core to your system, and connect the control bus to the AXI monitor.
4. In the SAV Ports tab, on the monitor core, set up the `MON_AXI_ACLK` port of the core to match the clock used by the AXI interface being probed.

Optionally, you can assign the `MON_AXI_TRIG_OUT` port and connect it to other `chipscope_axi_monitor` cores in the system.

Using Processor-less Embedded IP in Project Navigator

You might want to use portions of EDK IP outside of a processor system. For example, you can use an AXI Interconnect core block to create a multiported DDR3 controller. XPS can be used to manage, connect, and deliver EDK IP, even without a processor. See Answer Record37856 [Ref 33]for more information.

Using System Generator: DSP Edition

System Generator for DSP supports both AXI4 and AXI4-Stream interfaces, as follows:

- AXI4 interface in conjunction with the EDK Processor Block.
- AXI4-Stream interface in IP within the System Generator AXI4 block library.

AXI4 Support in System Generator

AXI4 (memory mapped) support in System Generator is available through the EDK Processor block found in the System Generator block set.

The EDK Processor block lets you connect hardware circuits created in System Generator to a Xilinx MicroBlaze™ processor; options to connect to the processor using either a PLBv4.6 or an AXI4 interface are available.

You do not need to be familiar with the AXI4 nomenclature when using the System Generator flow because the EDK Processor block provides an interface that is memory-centric and works with multiple bus types.

You can create hardware that uses shared registers, shared FIFOs, and shared memories, and the EDK Processor block manages the memory connection to the specified interface.

Figure 2-1 shows the EDK Processor Implementation tab with an AXI4 bus type selected.

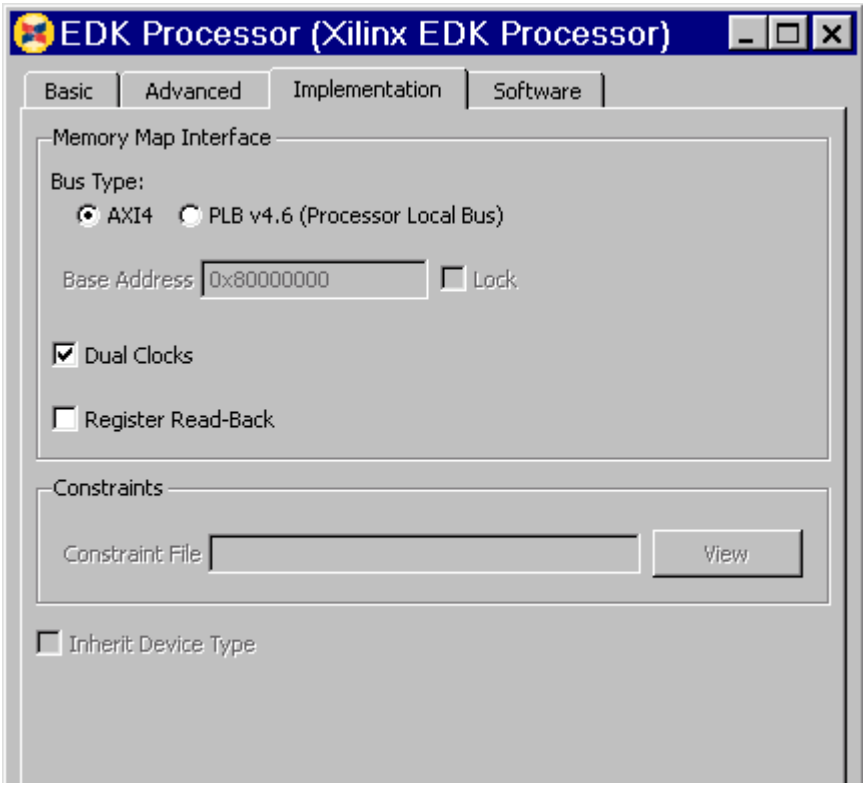


Figure 2-1: **EDK Processor Interface Implementation Tab**

Port Name Truncation

System Generator shortens the AXI4-Stream signal names to improve readability on the block; this is cosmetic and the complete AXI4-Stream name is used in the netlist. The name truncation is turned on by default; uncheck the **Display shortened port names** option in the block parameter dialog box to see the full name.

Port Groupings

System Generator groups together and color-codes blocks of AXI4-Stream channel signals.

In the example illustrated in the following figure, the top-most input port, `data_tready`, and the top two output ports, `data_tvalid` and `data_tdata` belong in the same AXI4-Stream channel, as well as `phase_tready`, `phase_tvalid`, and `phase_tdata`.

System Generator gives signals that are not part of any AXI4-Stream channels the same background color as the block; the `rst` signal, shown in Figure 2-2, page 15, is an example.

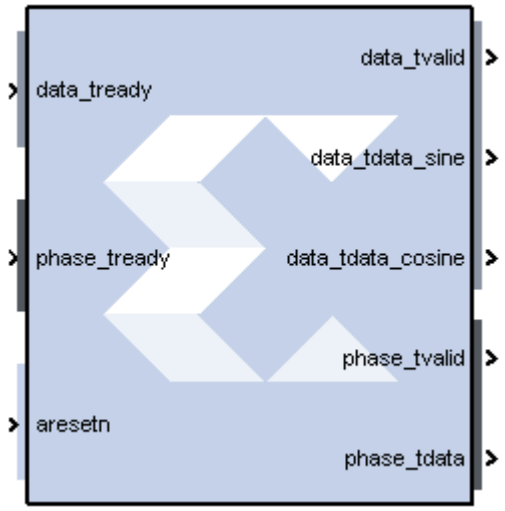


Figure 2-2: **Block Signal Groupings**

Breaking Out Multi-Channel TDATA

The `TDATA` signal in an AXI4-Stream can contain multiple channels of data. In System Generator, the individual channels for `TDATA` are broken out; for example, in the complex multiplier shown in Figure 2-3 the `TDATA` for the `dout` port contains both the imaginary and the real number components.

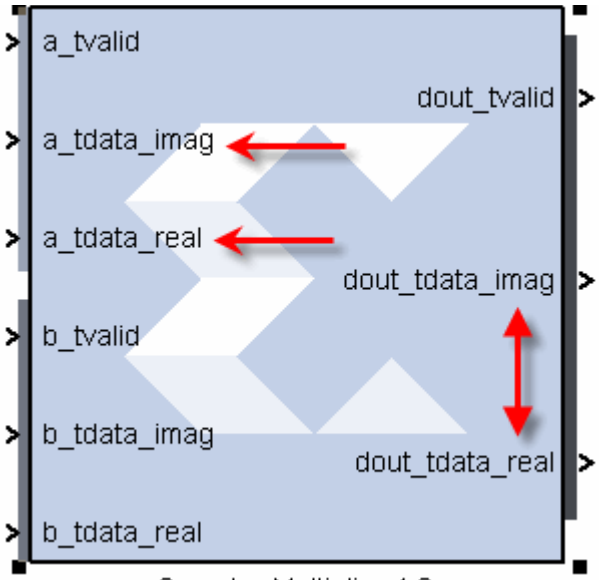


Figure 2-3: **Multi-Channel TDATA**

Note: Breaking out of multi-channel TDATA does not add additional logic to the design. The data is correctly byte-aligned also.

For more information about System Generator and AXI IP creation, see the following Xilinx website: http://www.xilinx.com/tools/sysgen.htm.

Using Xilinx AXI IP: Logic Edition

Xilinx IP with an AXI4 interface can be accessed directly from the IP catalog in CORE Generator, Project Navigator, and PlanAhead. An AXI4 column in the IP catalog shows IP with AXI4 support. The IP information panel displays the supported AXI4, AXI4-Stream, and AXI4-Lite interface.

Generally, for Virtex®-6 and Spartan®-6 device families, the AXI4 interface is supported by the latest version of an IP. Older, "Production," versions of IP continue to be supported by the legacy interface for the respective core on Virtex-6, Spartan-6, Virtex®-5, Virtex®-4 and Spartan®-3 device families. The IP catalog displays all "Production" versions of IP by default. Figure 2-4 shows the IP Catalog in CORE Generator.

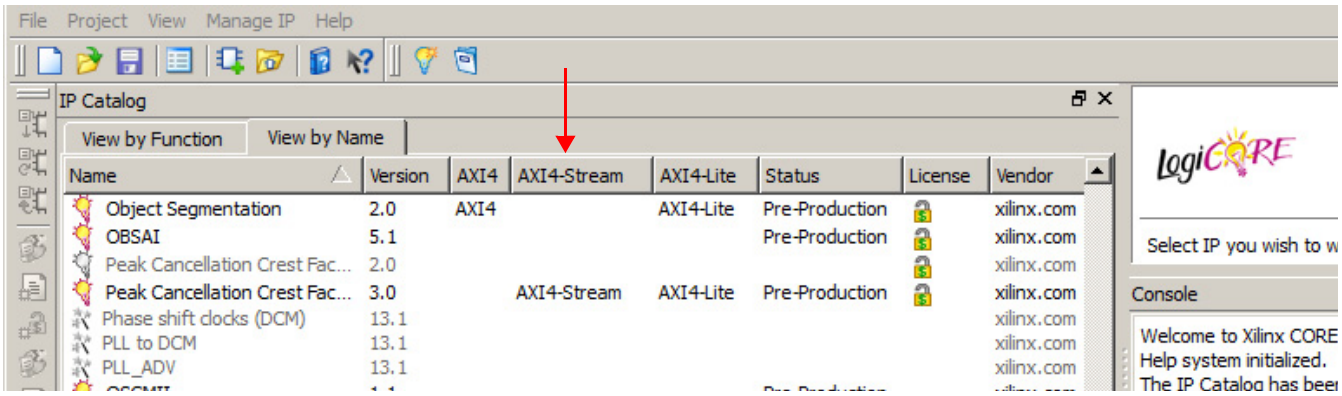


Figure 2-4: **IP Catalog in Xilinx Tools**

Figure 2-5 shows the IP catalog in the PlanAhead tool with the equivalent AXI4 column and the supported AXI4 interfaces in the IP details panel.

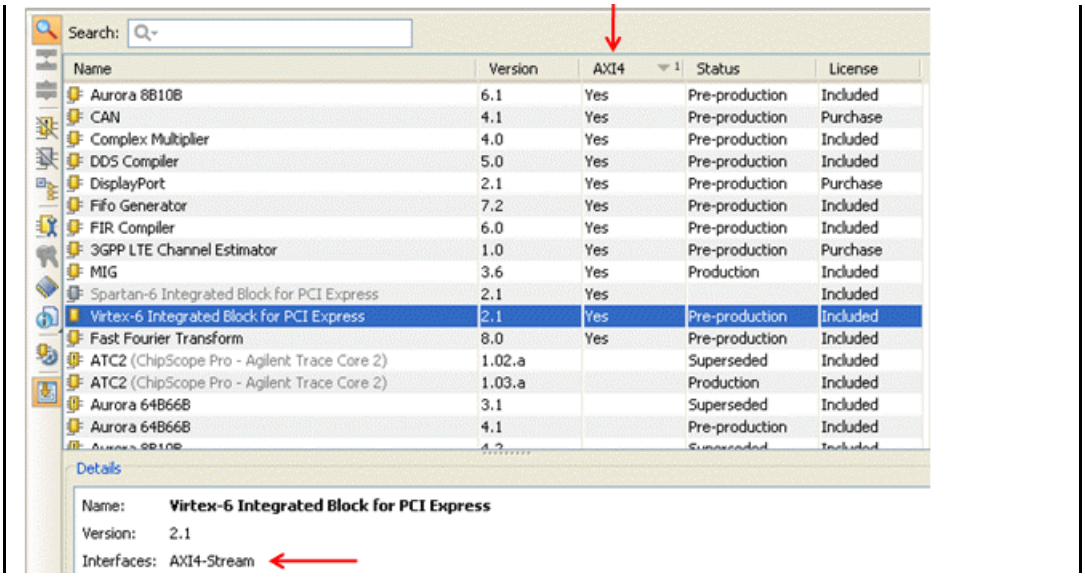


Figure 2-5: **IP Catalog in PlanAhead Tool**

New HDL designs for AXI4, AXI4-Lite, and AXI4-Stream masters and slaves can reference AXI IP HDL design templates provided in Xilinx Answer Record37856. Check this answer record periodically for updates or new templates.

Xilinx AXI Infrastructure IP

Xilinx has migrated a significant portion of the available IP to AXI protocol. This section provides an overview of the more complex IP that is used in many AXI-based systems.

The following common infrastructure Xilinx IP is available for Virtex®-6, Spartan®-6, and 7 series devices, and future device support:

- Xilinx AXI Interconnect Core IP
- Connecting AXI Interconnect Core Slaves and Masters
- External Masters and Slaves
- DataMover
- Xilinx AXI4-Stream Interconnect Core IP
- Centralized DMA
- Ethernet DMA
- Video DMA
- Memory Control IP and the Memory Interface Generator

Refer to Chapter 4, "Migrating to Xilinx AXI Protocols," for more detailed usage information. See the following for a list of all AXI IP: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm.

Xilinx AXI Interconnect Core IP

The AXI Interconnect core 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 core IP is intended for memory-mapped transfers only; AXI4-Stream transfers are not applicable, but instead can use the AXI4-Stream Interconnect core IP (`axis_interconnect`). IP with AXI4-Stream interfaces are generally connected to one another, to DMA IP, or to the AXI4-Stream Interconnect IP.

The AXI Interconnect core IP is provided as a non-encrypted, non-licensed (free) pcore in the Xilinx Embedded Development Toolkit (EDK) and in the Project Navigator for use in non-embedded designs using the CORE Generator® tool.

See the *AXI_Interconnect IP (DS768)* [Ref 6]for more information.

AXI Interconnect Core Features

The AXI Interconnect IP contains the following features:

- AXI protocol compliant (AXI3, AXI4, and AXI4-Lite), which includes:
  - Burst lengths up to 256 for incremental (`INCR`) bursts
  - Converts AXI4 bursts >16 beats when targeting AXI3 slave devices by splitting transactions.
  - Generates `REGION` outputs for use by slave devices 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 core (optional)
- Interface data widths:
  - AXI4: 32, 64, 128, 256, 512, or 1024 bits.
  - AXI4-Lite: 32 bits
- 32-bit address width
- The Slave Interface (SI) of the core can be configured to include 1-16 SI slots to accept transactions from up to 16 connected master devices. The Master Interface (MI) can be configured to comprise 1-16 MIT slots to issue transactions to up to 16 connected slave devices.
- Connects 1-16 masters to 1-16 slaves:
  - When connecting one master to one slave, the AXI Interconnect core 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, pathways through the AXI Interconnect core are implemented as wires, with no resources, no delay and no latency.

Note: When used in a non-embedded system such as CORE Generator, the AXI Interconnect core connects multiple masters to one slave, which is typically a memory controller.

- Built-in data-width conversion:
  - Each master and slave connection can independently use data widths of 32, 64, 128, 256, 512, or 1024 bits wide:
    - The internal crossbar can be configured to have a native data-width of 32, 64, 128, 256, 512, or 1024 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 (upsizing), data is packed (merged) optionally, when permitted by address channel control signals (CACHE modifiable bit is asserted).

    - When converting to a narrower interface (downsizing), 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 core exports reset signals resynchronized to the clock input associated with each SI and MI slot.
- Built-in AXI4-Lite protocol conversion:
    - The AXI Interconnect core can connect to any mixture of AXI4 and AXI4-Lite masters and slaves.
    - The AXI Interconnect core 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.
    - The AXI Interconnect core detects illegal AXI4-Lite transactions from AXI4 masters, such as any transaction that accesses more than one word. It 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 core 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 paths connecting to each master and each slave.
    - 32-deep LUT-RAM based.
    - 512-deep block RAM based.
    - Option to delay assertion of:
        - `AWVALID` until the complete burst is stored in the W-channel FIFO

    - ARVALID until the R-channel FIFO has enough vacancy to store the entire burst length
- Selectable Interconnect Architecture:
  - 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 can 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.
  - Shared Access Shared Data (SASD) mode (Area optimized):
    - Shared write data, shared read data, and single shared address pathways.
    - Issues one outstanding transaction at a time.
    - Minimizes resource utilization.
- Supports multiple outstanding transactions:
  - 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 SI slot that has reached its acceptance limit, or is targeting an MI slot that has reached its issuing limit, or is trying to access an MI slot in a manner that risks deadlock, is temporarily disqualified from arbitration, so that other SI slots can be granted arbitration.

- Supports TrustZone security for each connected slave as a whole:
  - If configured as a secure slave, only secure AXI accesses are permitted
  - Any non-secure accesses are blocked and the AXI Interconnect core returns a DECERR response to the master
- Support for read-only and write-only masters and slaves, resulting in reduced resource utilization.

AXI Interconnect Core Limitations

The AXI Interconnect core does not support the following AXI3 features:

- Atomic locked transactions; this feature was retracted by 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 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 SI to MI.
- The AXI Interconnect core does not convert multi-beat bursts into multiple single-beat transactions when connected to an AXI4-Lite slave.
- The AXI Interconnect core does not support low-power mode or propagate the AXI C-channel signals.
- The AXI Interconnect core 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 core provides no address remapping.
- The AXI Interconnect core provides no built-in conversion to non-AXI protocols, such as APB.
- The AXI Interconnect core 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.
- When used in the CORE Generator tool flow, the AXI Interconnect core can only be configured with one MI port (one connected slave device), and therefore performs no address decoding.

AXI Interconnect Core Diagrams

Figure 2-6 illustrates a top-level AXI Interconnect.

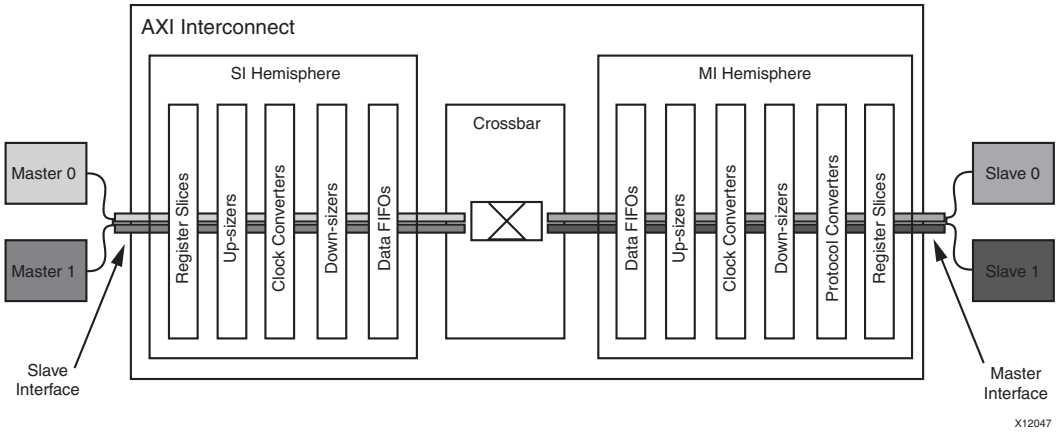


Figure 2-6: **Top-Level AXI Interconnect**

The AXI Interconnect core consists of the SI, the MI, and the functional units that comprise the AXI channel pathways between them. The SI accepts Write and read transaction requests from connected master devices. The MI issues transactions 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 core also includes 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 core down the middle between the SI-related functional units (SI hemisphere) and the MI-related units (MI hemisphere).

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

AXI Interconnect Core Use Models

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

- Pass-Through
- Conversion Only
- N-to-1 Interconnect
- 1-to-N Interconnect
- N-to-M Interconnect (Sparse Crossbar Mode)
- N-to-M Interconnect (Shared Access Mode)

Pass-Through

When there is only one master device and only one slave device connected to the AXI Interconnect core, and the AXI Interconnect core is not performing any optional conversion functions or pipelining, all pathways between the slave and master interfaces degenerate into direct wire connections with no latency and consuming no logic resources.

The AXI Interconnect core does, however, continue to resynchronize the `INTERCONNECT_ARESETN` input to each of the slave and master interface clock domains for any master or slave devices that connect to the `ARESET_OUT_N` outputs, which consumes a small number of flip-flops.

Figure 2-7 is a diagram of the Pass-Through use case.

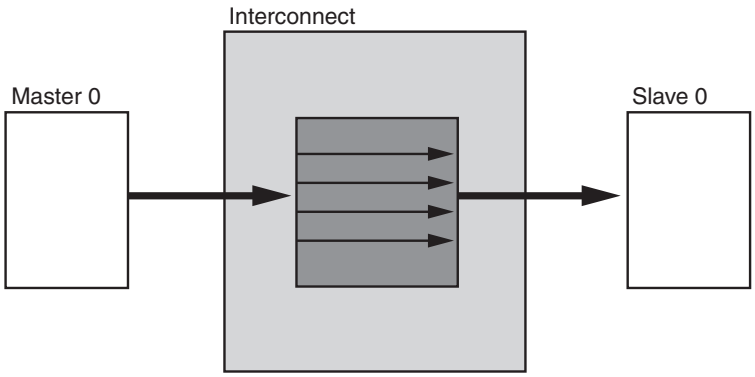


Figure 2-7: **Pass-through AXI Interconnect Use Case**

Conversion Only

The AXI Interconnect core can perform various conversion and pipelining functions when connecting one master device to one slave device. These are:

- Data width conversion
- Clock rate conversion
- AXI4-Lite slave adaptation
- AXI-3 slave adaptation
- Pipelining, such as a register slice or data channel FIFO

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

Figure 2-8 shows the one-to-one or conversion use case.

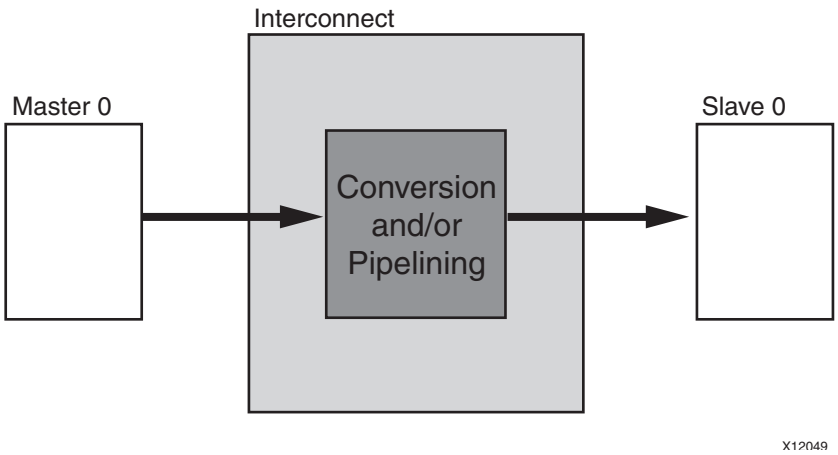


Figure 2-8: **1-to-1 Conversion AXI Interconnect Use Case**

N-to-1 Interconnect

A common degenerate configuration of AXI Interconnect core 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 core (unless address range validation is needed).

Conversion functions, such as data width and clock rate conversion, can also be performed in this configuration. Figure 2-9 shows the N to 1 AXI interconnection use case.

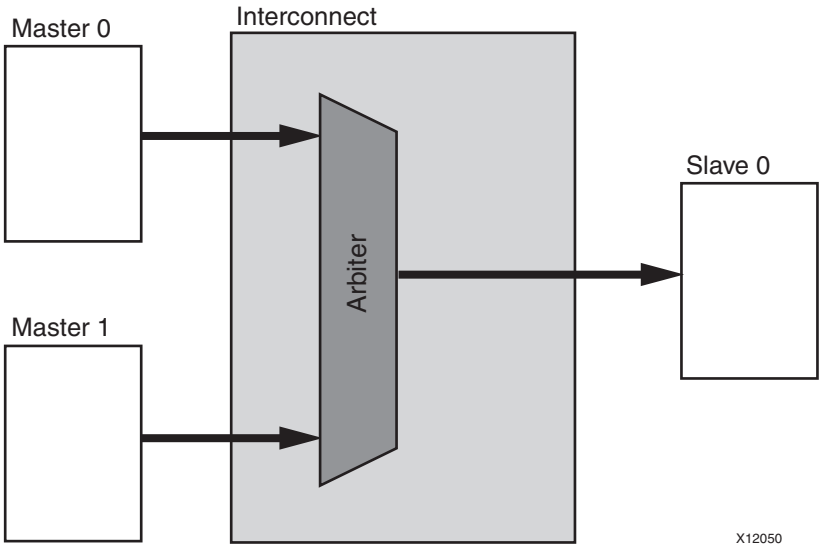


Figure 2-9: **N-to-1 AXI Interconnect**

1-to-N Interconnect

Another degenerative configuration of the AXI Interconnect core 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. Figure 2-10, shows the 1 to N Interconnect use case.

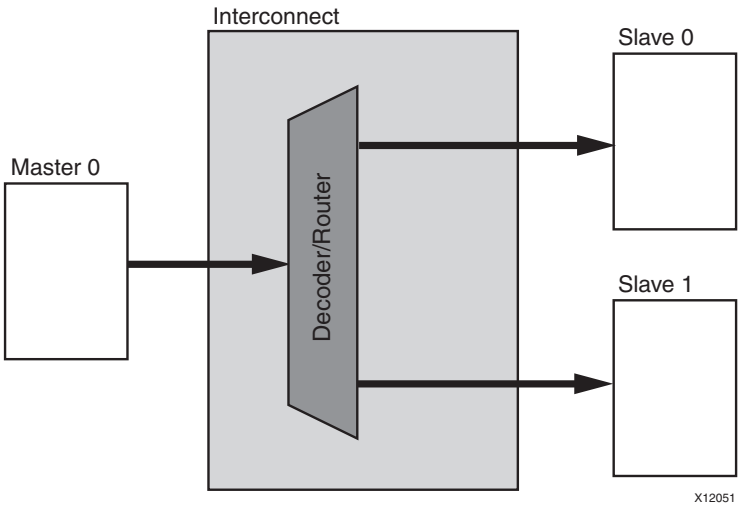


Figure 2-10: **1-to-N AXI Interconnect Use Case**

N-to-M Interconnect (Sparse Crossbar Mode)

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 2-11, page 26.

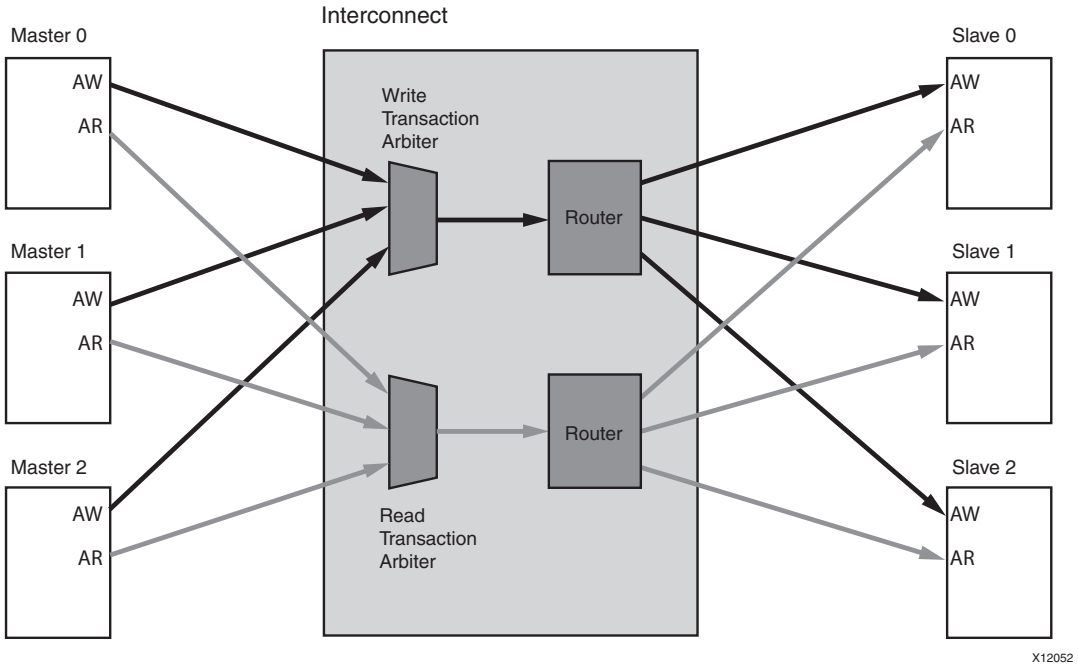


Figure 2-11: **Shared Write and Read Address Arbitration**

Figure 2-12 shows the sparse crossbar write and read data pathways.

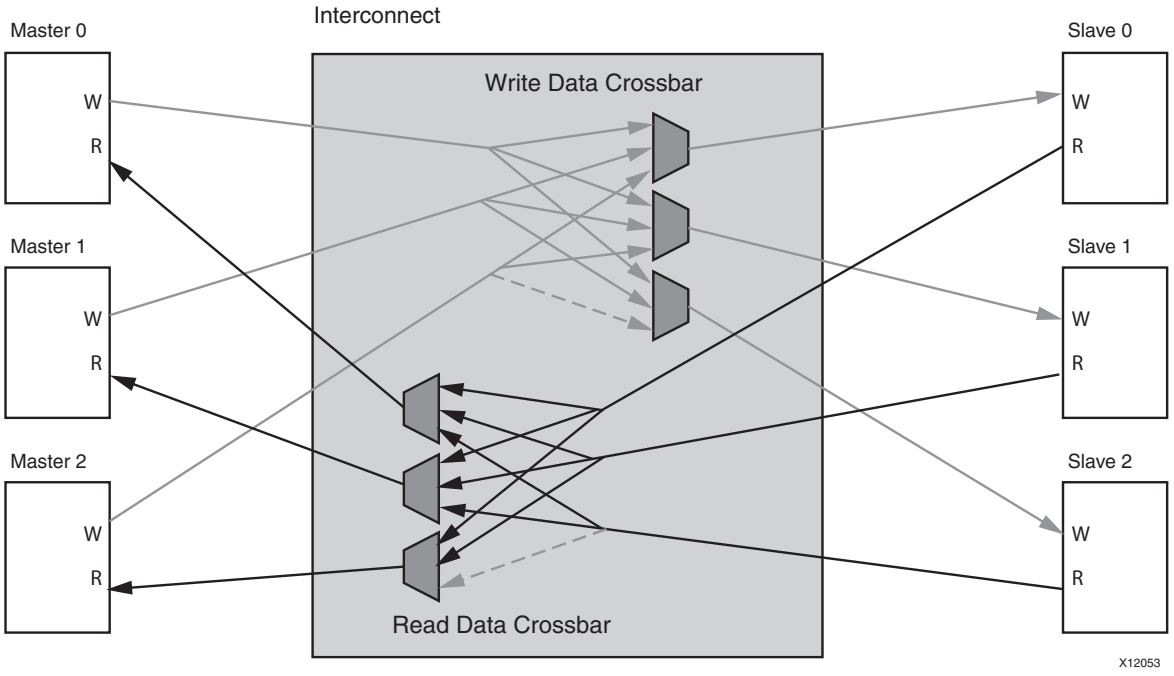


Figure 2-12: **Sparse Crossbar Write and Read Data Pathways**

Parallel write and read data pathways connect each SI slot (attached to AXI masters on the left) to all the MI slots (attached to AXI slaves on the right) 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.

N-to-M Interconnect (Shared Access Mode)

When in Shared Access mode, the N-to-M use case of the AXI Interconnect core provides for only one outstanding transaction at a time, as shown in Figure 2-13. For each connected master, read transactions requests always take priority over writes. The aribter then selects from among the requesting masters. A write or read data transfer is enabled to the targeted slave device. After the data transfer (including write response) completes, the next request is arbitrated. Shared Access mode minimizes the resources used to implement the crossbar module of the AXI Interconnect.

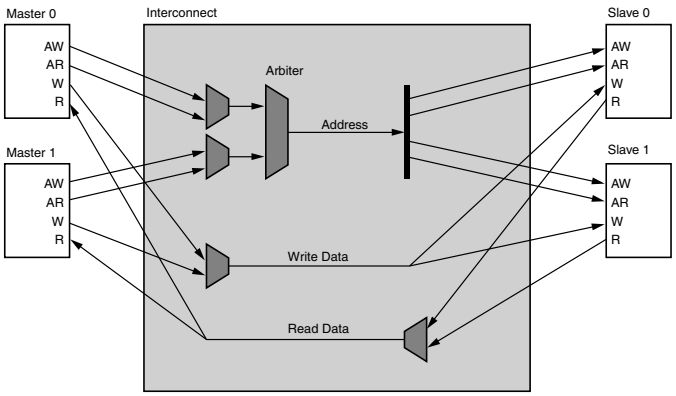


Figure 2-13: **Shared Access Mode**

Width Conversion

The AXI Interconnect core has a parametrically-defined, internal, native data width that supports 32, 64, 128, 256, 512, and 1024 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 core inserts width conversion units to adapt the slot width to the AXI Interconnect core native width before transiting the crossbar to the other hemisphere.

The width conversion functions differ, depending on whether the data path width gets wider ("upsizing") or more narrow ("downsizing"), when moving in the direction from the SI toward the MI.

The width conversion functions are the same in either the SI hemisphere (translating from the SI to the AXI Interconnect core native width) or the MI hemisphere (translating from the AXI Interconnect core native width to the MI).

MI and SI slots have an associated individual parametric data-width value. The AXI Interconnect core 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 downsizing conversion is performed along the pathways of the SI slot.
- When the internal native data width of the AXI Interconnect core is wider than that of an MI slot, a downsizing 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 upsizing conversion is performed along the pathways of the SI slot.
- When the internal native data width of the AXI Interconnect core is narrower than that of an MI slot, an upsizing conversion is performed along the pathways of the MI slot.

Typically, the data-width of the AXI Interconnect core is matched to that of the most throughput-critical peripheral, such as a memory controller, in the system design.

The following subsections describe the downsizing and upsizing behavior.

Downsizing

Downsizers 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 downsizing 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

The AXI Interconnect core sets the `RRESP` for each output data beat (on the SI) to the worst-case error condition encountered among the input data beats being merged, according to the following descending precedence order: DECERR, SLVERR, OKAY, EXOKAY.

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.

Upsizing

For upsizers 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

The AXI Interconnect core replicates the RRESP from each input data beat onto the RRESP of each output data beat (on the SI).

Clock Conversions

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 core 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 core 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 core 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 core 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, an optional, two-deep register slice (skid buffer) can be inserted 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

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

- The SI-side write data FIFO is located before crossbar module, after any SI-side width, or clock conversion.
- The MI-side write data FIFO is located after the crossbar module, before any MI slot width, clock, or protocol conversion.
- The MI-side read data FIFO is located before (on the MI side) of the crossbar module, after any MI-side width, or protocol conversion.
- The SI-side read data FIFO is located after (on the SI side) of the crossbar module, before any SI-side width, or clock conversion.

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

For more detail and the required signals and parameters of the AXI Interconnect core IP, refer to the *AXI Interconnect IP (DS768)* [Ref 6].

Connecting AXI Interconnect Core Slaves and Masters

You can connect the slave interface of one AXI Interconnect core module to the master interface of another AXI Interconnect core with no intervening logic using an AXI-to-AXI Connector (`axi2axi_connector`) IP. The `axi2axi_connector` IP provides the port connection points necessary to represent the connectivity in the system, plus a set of parameters used to configure the respective interfaces of the AXI Interconnect core modules being connected.),

AXI-To-AXI Connector Features

The features of the `axi2axi_connector` are:

- Connects the master interface of one AXI Interconnect core module to slave interface of another AXI Interconnect core module.
- Directly connects all master interface signals to all slave interface signals.
- Contains no logic or storage, and functions as a bus bridge in EDK.

Description

The AXI slave interface of the `axi2axi_connector`, *connector*, module always connects to one attachment point (slot) of the master interface of one AXI Interconnect core module (the *upstream interconnect*). The AXI master interface of the connector always connects to one slave interface slot of a different AXI Interconnect core module (the *downstream interconnect*) as shown in Figure 2-14.

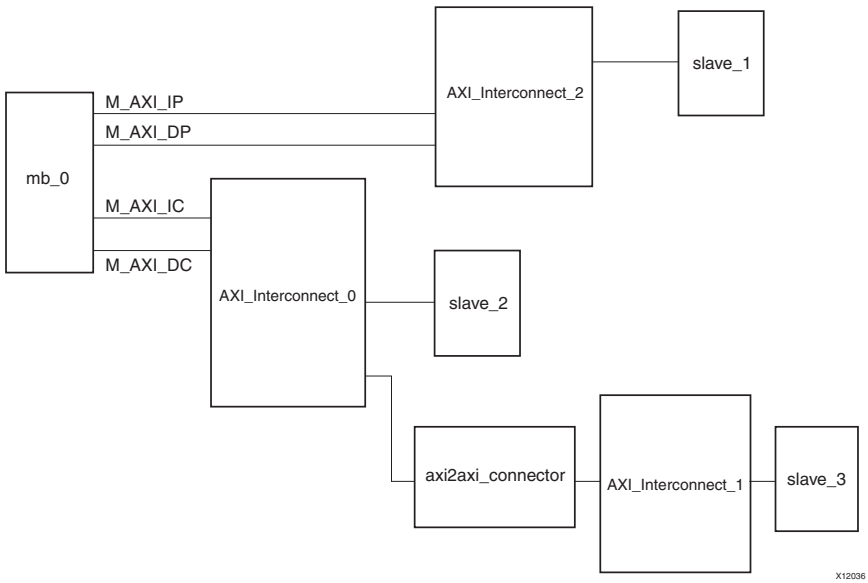


Figure 2-14: **Master and Slave Interface Modules Connecting Two AXI Interconnect cores**

Using the AXI To AXI Connector

When using the AXI To AXI Connector (`axi2axi_connector`) you can cascade two AXI Interconnect cores. The EDK tools set the data width and clock frequency parameters on the `axi2axi_connector` IP so that the characteristics of the master and slave interfaces match.

Also, the EDK tools auto-connect the clock port of the `axi2axi_connector` so that the interfaces of the connected interconnect modules are synchronized by the same clock source.

For more detail and the required signals and parameter of the AXI To AXI Connector, see the *AXI To AXI Connector IP Data Sheet (DS803)* [Ref 7].

External Masters and Slaves

When there is an AXI master or slave IP module that is not available as an EDK pcore (such as a pure HDL module) that needs to be connected to an AXI Interconnect core inside the EDK sub-module, you can use these external utility cores for that the purpose. The AXI master or slave module remains in the top-level of the design, and the AXI signals are connected to the EDK sub-system using this utility pcore.

Features

- Connects an AXI master or slave interface to the AXI Interconnect core IP.
- Master or slave AXI bus interface is on one side and AXI ports are on the other side.
- Models other ports as an I/O interface, which can be made external, thereby providing the necessary signals that can be connected to a top-level master or slave.

Figure 2-15 is a block diagram of the AXI external master connector.

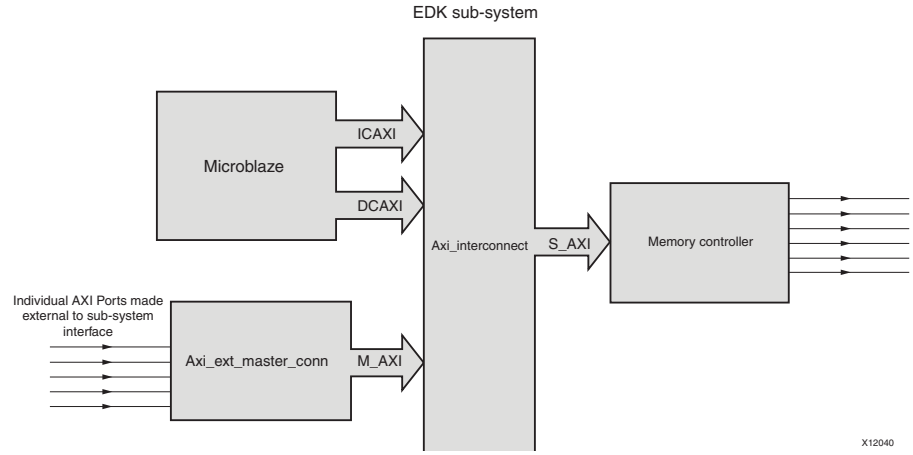


Figure 2-15: **EDK Sub-system using an External Master Connector**

Figure 2-16 shows a block diagram of the external slave connector.

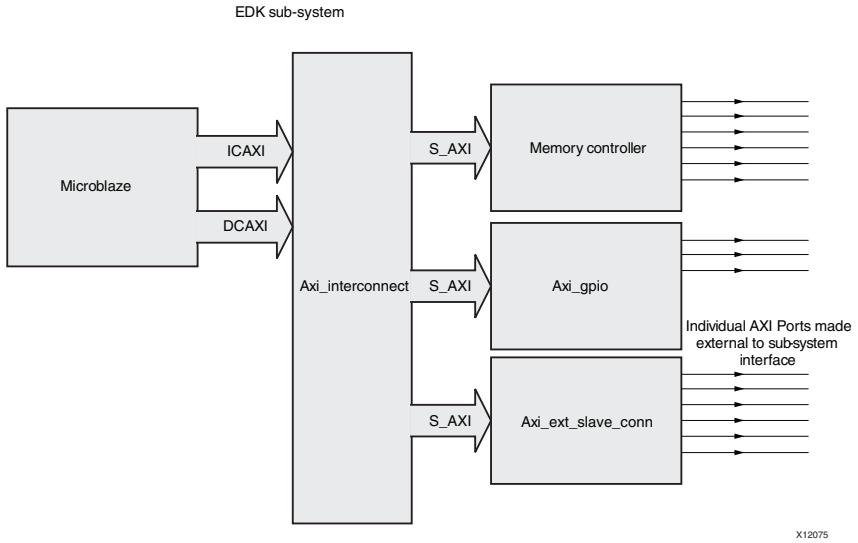


Figure 2-16: **EDK Subsystem using an External Slave Connector**

The Platform Studio IP Catalog contains the external master and external slave connectors. For more information, refer to the Xilinx website: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm.

DataMover

The AXI DataMover is an important interconnect infrastructure IP that enables high throughput transfer of data between the AXI4 memory-mapped domain to the AXI4-Stream domain. It provides Memory Map to Stream and Stream to Memory Map channels that operate independently in a full, duplex-like method. The DataMover IP has the following features:

- Enables 4k byte address boundary protection
- Provides automatic burst partitioning
- Provides the ability to queue multiple transfer requests.

It also provides byte-level data realignment allowing memory reads and writes to any byte offset location.
Xilinx recommends that you use AXI DataMover as a bridge between AXI4-Stream and AXI4 Memory Map interfaces for both write and read operations where the AXI4-Stream Master controls data flow through command and status bus.

The AXI DataMover is available in both CORE Generator and XPS. Figure 2-17, page 34 shows a block diagram of the DataMover functionality. See more information on the product page http://www.xilinx.com/products/intellectual-property/axi_datamover.htm.

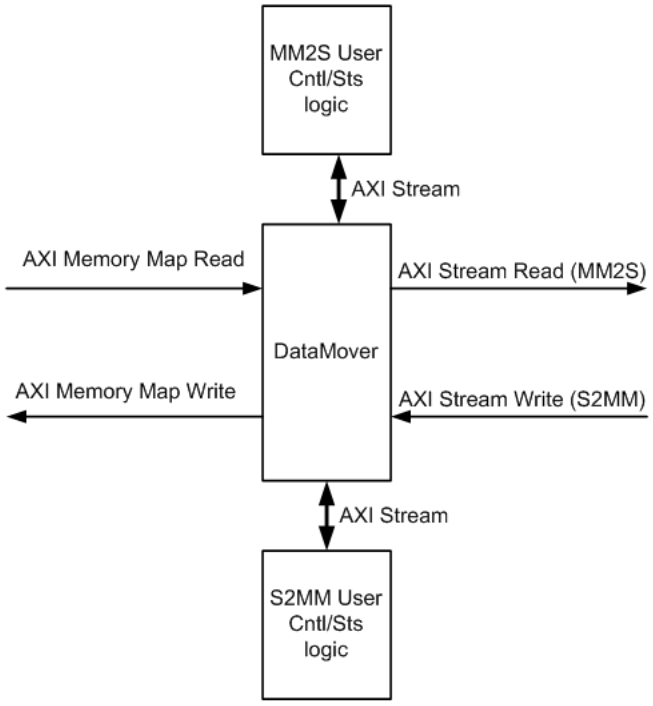


Figure 2-17: **Data Mover Block Diagram**

Xilinx AXI4-Stream Interconnect Core IP

The AXI4-Stream Interconnect core IP (`axis_interconnect`) connects one or more AXI4-Stream master devices to one or more stream slave devices. The AXI interfaces conform to the ARM®*AMBA*® *AXI4-Stream Protocol Specification* [Ref 2].

Note: The AXI4-Steam Interconnect core IP is intended for AXI4-Stream transfers only; AXI memory mapped transfers are not applicable.

The AXI4-Stream Interconnect core IP is:

- Provided as a non-encrypted, non-licensed (free) core in the Xilinx ISE® Design Suite Project Navigator for use in non-embedded designs using the CORE Generator tool.
- Available in the Xilinx Vivado™ Design Suite.

See the *AXI4-Stream Interconnect IP Product Guide (PG035)* [Ref 12] for more information.

AXI4-Stream Interconnect Core Features

The AXI4-Stream Interconnect IP contains the following features:

- AXI4-Stream compliant:
  - Supports all AXI4-Stream defined signals: TVALID, TREADY, TDATA, TSTRB, TKEEP, TLAST, TID, TDEST, and TUSER
    - TDATA, TSTRB, TKEEP, TLAST, TID, TDEST, and TUSER are optional
    - Programmable TDATA, TID, TDEST, and TUSER widths (TSTRB and TKEEP width is TDATA width/8)
  - Per port ACLK/ARESETn inputs (supports clock domain crossing)
  - Per port ACLKEN inputs (optional)
- Core switch:
  - 1-16 masters
  - 1-16 slaves
  - Full slave-side arbitrated crossbar switch
  - Slave input to master output routing based on TDEST value decoding and comparison against base and high value range settings
  - Round-Robin and Priority arbitration
    - Arbitration suppress capability to prevent head-of-line blocking
    - Native switch data width 8, 16, 24, 32, 48, ... 4096 bits (any byte width up to 512 bytes)
    - Arbitration tuning parameters to arbitrate on TLAST boundaries, after a set number of transfers, and/or after a certain number of idle clock cycles
  - Optional pipeline stages after internal TDEST decoder and arbiter functional blocks
  - Programmable connectivity map to specify full or sparse crossbar connectivity
- Built-in data width conversion:
  - Each master and slave connection can independently use data widths of 8, 16, 24, 32, 48, ... 4096 bits (any byte width up to 512 bytes)
- 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)

- Optional register-slice pipelining:
  - Available on each AXI4-Stream channel connecting to each master and slave device.
  - Facilitates timing closure by trading-off frequency versus latency.
  - One latency cycle per register-slice, with no loss in data throughput in the register slice under all AXI4-Stream handshake conditions.
- Optional data path FIFO buffering:
  - Available on data paths connecting to each master and each slave
  - 16, 32, 64, 128, through 32768 deep (16-deep and 32-deep are LUT-RAM based; otherwise are block RAM based)
  - Normal and Packet FIFO modes (Packet FIFO mode is also known as store-and-forward in which a packet is stored and only released downstream after a TLAST packet boundary is detected.)
  - FIFO data count outputs to report FIFO occupancy
- Additional error flags to detect conditions such as `TDEST` decode error, sparse `TKEEP` removal, and packer error

AXI4-Stream Interconnect Core Diagrams

Figure 2-18 illustrates a top-level AXI4-Stream Interconnect architecture.

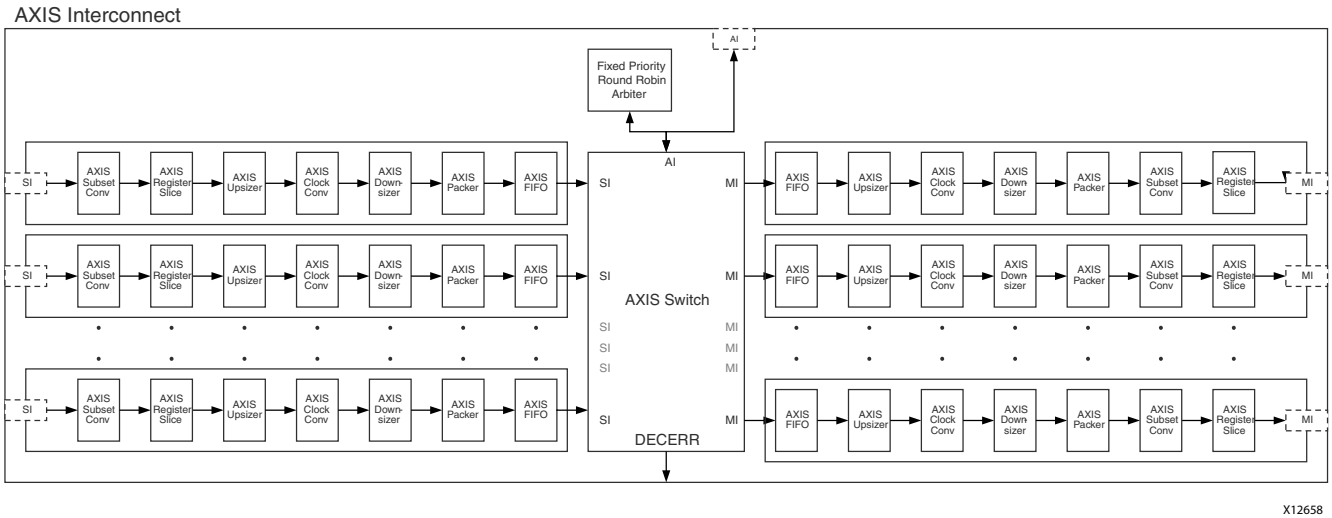


Figure 2-18: **Top-Level AXI4-Stream Interconnect Achitecture**

The AXI4-Stream Interconnect core consists of the SI, the MI, and the functional units that include the AXI channel pathways between them.

- The SI accepts transaction requests from connected master devices.
- The MI issues transactions to slave devices.
- At the center is the switch that arbitrates and routes traffic between the various devices connected to the SI and MI.

The AXI4-Stream Interconnect core also includes other functional units located between the switch and each of the SI and MI interfaces that optionally perform various conversion and storage functions. The switch effectively splits the AXI4-Stream Interconnect core down the middle between the SI-related functional units (*SI hemisphere*) and the MI-related units (*MI hemisphere*). This architecture is similar to that of the AXI Interconnect.

AXI4-Stream Interconnect Core Use Models

The AXI4-Stream Interconnect IP core connects one or more AXI4-Stream master devices to one or more AXI4-Stream slave devices. The following subsections describe the possible use cases:

- Streaming data routing and switching
- Stream multiplexing and demultiplexing

Streaming Data Routing and Switching (Crossbar Mode)

The AXI4-Stream Interconnect can implement a full N x M crossbar switch as shown in Figure 2-19. It supports slave side arbitration capable of parallel data traffic between N masters and M slaves. Decoders and arbiters serve to route transactions between masters and slaves.

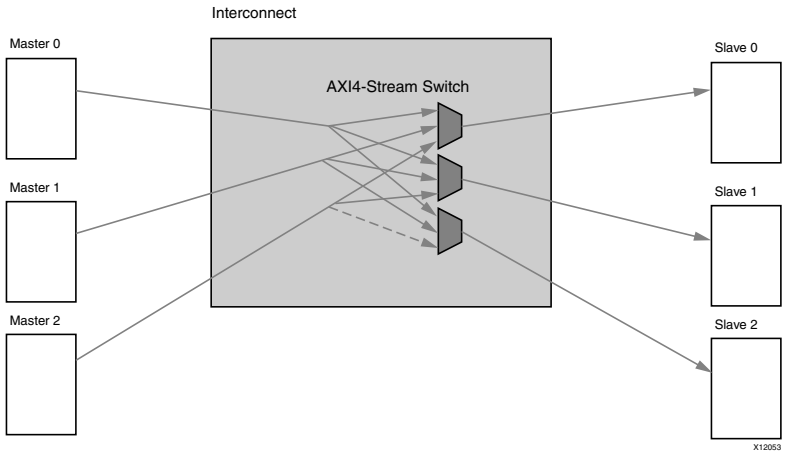


Figure 2-19: N x M Crossbar Switch (Crossbar Mode)

Stream Multiplexing and Demultiplexing

You can configure the AXI4-Stream Interconnect in an Nx1 configuration to multiplex streams together and then configured as 1xM to demultiplex streams.

Use multiplexing and demultiplexing to create multi-channel streams where a smaller number of wires can carry shared traffic from multiple masters or slaves.

For example, in Figure 2-20, AXI4-Stream interconnects are used with the AXI virtual FIFO controller to multiplex and demultiplex multiple streams from multiple endpoint masters and slaves together. For more information, see the *AXI Virtual FIFO Controller (PG038)* [Ref 15].

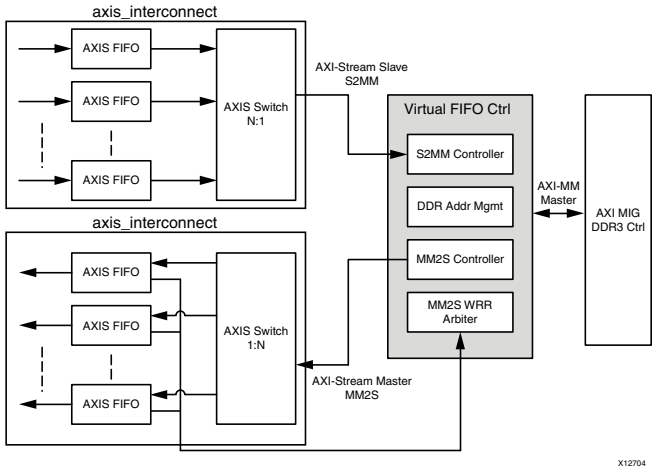


Figure 2-20: **IAXI4-Stream Interconnects with AXI Virtual FIFO Controller**

AXI Virtual FIFO Controller

The Xilinx LogiCORE™ IP AXI Virtual FIFO Controller core (VFIFO) is a high performance core that implements multiple AXI4-Stream FIFOs. The memory storage for data contained in the FIFOs comes from an attached AXI4 slave memory controller. The VFIFO core manages multiple sets of read and write address pointers to emulate the behavior of multiple independent FIFOs.

An AXI4 slave memory controller with external memory can provide large depths of external SRAM or DDR memory. VFIFO is useful in applications using PCIe, DSP, video, or Ethernet that require FIFOs that are deeper than can be otherwise constructed from an on-chip block RAM memory, distributed RAM memory, or external chips.

The virtual FIFO controller can send and receive multiplexed streams to implement up to 8 logical AXI4-Stream FIFOs. It can be used in conjunction with the Xilinx AXI4-Stream Interconnect Core IP, page 34 to route the data to each endpoint IP as shown in Figure 2-22, page 39. The AXI4-Stream interconnect can also perform local FIFO buffering, clock conversion, and width conversion to adapt the interface of the stream endpoints to the data path of the virtual FIFO controller and the AXI memory controller.

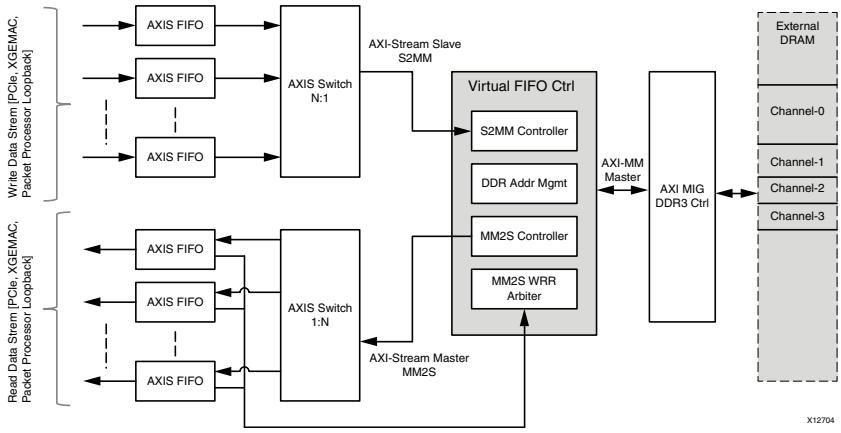


Figure 2-22: **AXI4-Stream with Virtual FIFO Controller**

Centralized DMA

Xilinx provides a Centralized DMA core for AXI. This core replaces legacy PLBv4.6 Centralized DMA with an AXI4 version that contains enhanced functionality and higher performance. Figure 2-23 shows a typical embedded system architecture incorporating the AXI (AXI4 and AXI4-Lite) Centralized DMA.

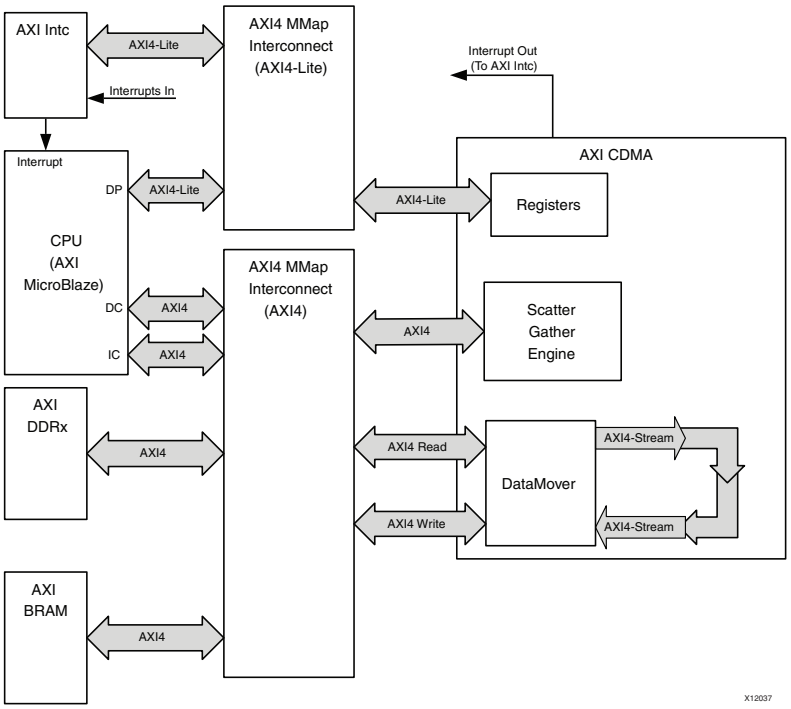


Figure 2-23: **Typical Use Case for AXI Centralized DMA**

The AXI4 Centralized DMA performs data transfers from one memory mapped space to another memory mapped space using high speed, AXI4, bursting protocol under the control of the system microprocessor.

AXI Centralized DMA Summary

The AXI Centralized DMA provides the same simple transfer mode operation as the legacy PLBv4.6 Centralized DMA. A simple mode transfer is defined as that which the CPU programs the Centralized DMA register set for a single transfer and then initiates the transfer. The Centralized DMA:

- Performs the transfer
- Generates an interrupt when the transfer is complete
- Waits for the microprocessor to program and start the next transfer

Also, the AXI Centralized DMA includes an optional data realignment function for 32- and 64-bit bus widths. This feature allows addressing independence between the transfer source and destination addresses.

AXI Centralized DMA Scatter Gather Feature

In addition to supporting the legacy PLBv4.6 Centralized DMA operations, the AXI Centralized DMA has an optional Scatter Gather (SG) feature.

SG enables the system CPU to off-load transfer control to high-speed hardware automation that is part of the Scatter Gather engine of the Centralized DMA. The SG function fetches and executes pre-formatted transfer commands (buffer descriptors) from system memory as fast as the system allows with minimal required CPU interaction. The architecture of the Centralized DMA separates the SG AXI4 bus interface from the AXI4 data transfer interface so that buffer descriptor fetching and updating can occur in parallel with ongoing data transfers, which provides a significant performance enhancement.

DMA transfer progress is coordinated with the system CPU using a programmable and flexible interrupt generation approach built into the Centralized DMA. Also, the AXI Centralized DMA allows the system programmer to switch between using Simple Mode transfers and SG-assisted transfers using the programmable register set.

The AXI Centralized DMA is built around the new high performance AXI DataMover helper core which is the fundamental bridging element between AXI4-Stream and AXI4 memory mapped buses. In the case of AXI Centralized DMA, the output stream of the DataMover is internally looped back to the input stream. The SG feature is based on the Xilinx SG helper core used for all Scatter Gather enhanced AXI DMA products.

Centralized DMA Configurable Features

The AXI4 Centralized DMA lets you trade-off the feature set implemented with the FPGA resource utilization budget. The following features are parameterizable at FPGA implementation time:

- Use DataMover Lite for the main data transport (Data Realignment Engine (DRE) and SG mode are not supported with this data transport mechanism)
- Include or omit the Scatter Gather function
- Include or omit the DRE function (available for 32- and 64-bit data transfer bus widths only)
- Specify the main data transfer bus width (32, 64, 128, 256, 512, and 1024 bits)
- Specify the maximum allowed AXI4 burst length the DataMover will use during data transfers

Centralized DMA AXI4 Interfaces

The following table summarizes the four external AXI4 Centralized DMA interfaces in addition to the internally-bridged DataMover stream interface within the AXI Centralized DMA function.

Table 2-1: **AXI Centralized DMA AXI4 Interfaces**

Interface	AXI Type	Data Width	Description
Control	AXI4-Lite slave	32	Used to access the AXI Centralized DMA internal registers. This is generally used by the system processor to control and monitor the AXI Centralized DMA operations.
Scatter Gather	AXI4 master	32	An AXI4 memory mapped master that is used by the AXI Centralized DMA to read DMA transfer descriptors from System Memory and then to write updated descriptor information back to System Memory when the associated transfer operation has completed.
Data MMap Read	AXI4 Read master	32, 64, 128, 256, 512, 1024	Reads the transfer payload data from the memory mapped source address. The data width is parameterizable to be 32, 64, 128, 256, 512, and 1024 bits wide.
Data MMap Write	AXI4 Write master	32, 64, 128, 256, 512, 1024	Writes the transfer payload data to the memory mapped destination address. The data width is parameterizable to be 32, 64, 128, 256, 512, and 1024 bits wide, and is the same width as the Data Read interface.

Ethernet DMA

The AXI4 protocol adoption in Xilinx embedded processing systems contains an Ethernet solution with Direct Memory Access (DMA). This approach blends the performance advantages of AXI4 with the effective operation of previous Xilinx Ethernet IP solutions.

Figure 2-24 provides high-level block diagram of the AXI DMA.

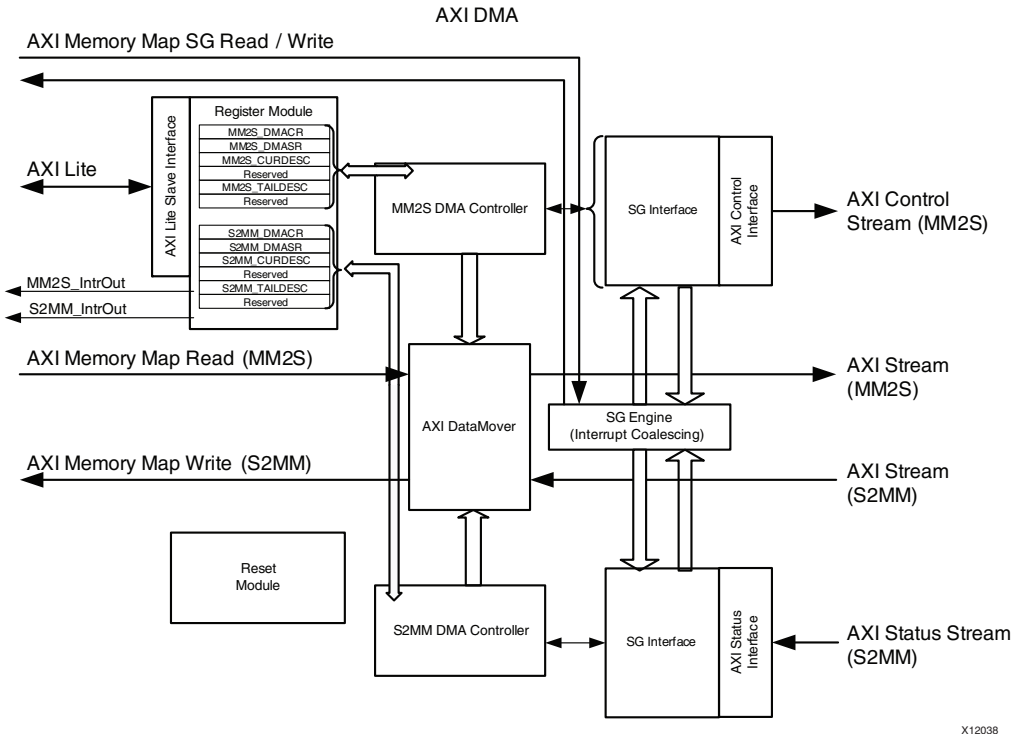


Figure 2-24: **AXI DMA High Level Block Diagram**

Figure 2-25 shows a typical system architecture for the AXI Ethernet.

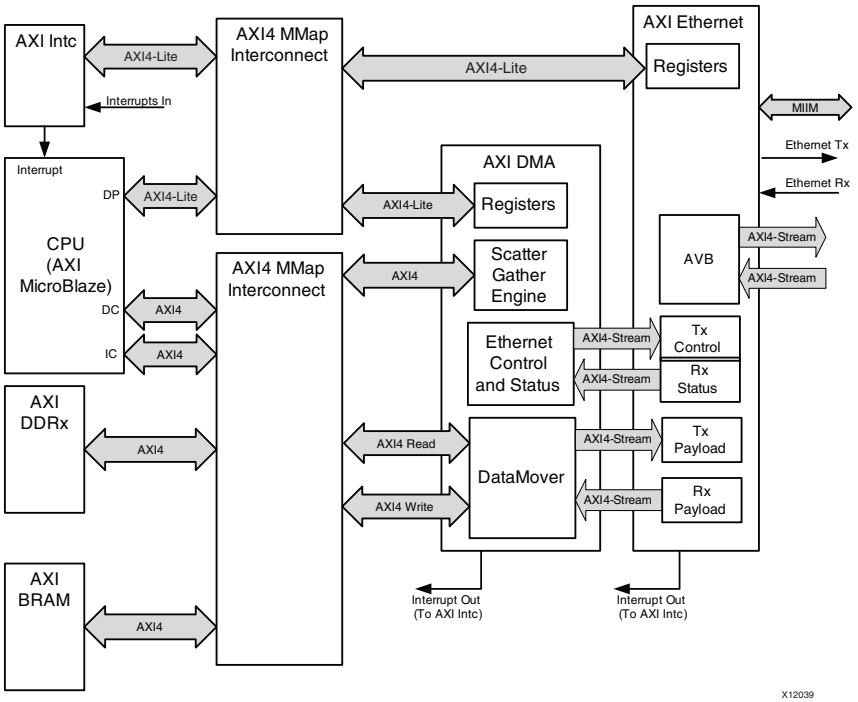


Figure 2-25: **Typical Use Case for AXI DMA and AXI4 Ethernet**

As shown in Figure 2-25, the AXI Ethernet is now paired with a new AXI DMA IP. The AXI DMA replaces the legacy PLBv4.6 SDMA function that was part of the PLBv4.6 Multi-Port Memory Controller (MPMC).

The AXI DMA is used to bridge between the native AXI4-Stream protocol on the AXI Ethernet to AXI4 memory mapped protocol needed by the embedded processing system.

The AXI DMA core can also be connected to a user system other than an Ethernet-based AXI IP. In this case, the parameter `C_SG_INCLUDE_STSCNTRL_STRM` must be set to 0 to exclude status and control information and use it for payload only.

AXI4 DMA Summary

The AXI DMA engine provides high performance direct memory access between system memory and AXI4-Stream type target peripherals. The AXI DMA provides Scatter Gather (SG) capabilities, allowing the CPU to offload transfer control and execution to hardware automation.

The AXI DMA as well as the SG engines are built around the AXI DataMover helper core (shared sub-block) that is the fundamental bridging element between AXI4-Stream and AXI4 memory mapped buses.

AXI DMA provides independent operation between the Transmit channel Memory Map to Slave (MM2S) and the Receive channel Slave to Memory Map (S2MM), and provides optional independent AXI4-Stream interfaces for offloading packet metadata.

An AXI control stream for MM2S provides user application data from the SG descriptors to be transmitted from AXI DMA.

Similarly, an AXI status stream for S2MM provides user application data from a source IP like AXI4 Ethernet to be received and stored in a SG descriptors associated with the Receive packet.

In an AXI Ethernet application, the AXI4 control stream and AXI4 status stream provide the necessary functionality for performing checksum offloading.

Optional SG descriptor queuing is also provided, allowing fetching and queuing of up to four descriptors internally in AXI DMA. This allows for very high bandwidth data transfer on the primary data buses.

DMA AXI4 Interfaces

The Xilinx implementation for DMA makes extensive use of the AXI4 capabilities. Table 2-2 summarizes the eight AXI4 interfaces used in the AXI DMA function.

Table 2-2: **AXI DMA Interfaces**

Interface	AXI Type	Data Width	Description
Control	AXI4-Lite slave	32	Used to access the AXI DMA internal registers. This is generally used by the System Processor to control and monitor the AXI DMA operations.
Scatter Gather	AXI4 master	32	An AXI4 memory mapped master used by the AXI4 DMA to Read DMA transfer descriptors from system memory and write updated descriptor information back to system memory when the associated transfer operation is complete.
Data MM Read	AXI4 Read master	32, 64, 128, 256, 512, 1024	Transfers payload data for operations moving data from the memory mapped side of the DMA to the Main Stream output side.
Data MM Write	AXI4 Write master	32, 64, 128, 256, 512, 1024	Transfers payload data for operations moving data from the Data Stream In interface of the DMA to the memory mapped side of the DMA.
Data Stream Out	AXI4-Stream master	32, 64, 128, 256, 512, 1024	Transfers data read by the Data MM Read interface to the target receiver IP using the AXI4-Stream protocol.

Table 2-2: **AXI DMA Interfaces *(Cont'd)***

Interface	AXI Type	Data Width	Description
Data Stream In	AXI4-Stream slave	32, 64, 128, 256, 512, 1024	Received data from the source IP using the AXI4-Stream protocol. Transferred the received data to the Memory Map system using the Data MM Write Interface.
Control Stream Out	AXI4-Stream master	32	The Control stream Out is used to transfer control information imbedded in the Tx transfer descriptors to the target IP.
Status Stream In	AXI4-Stream slave	32	The Status Stream In receives Rx transfer information from the source IP and updates the data in the associated transfer descriptor and written back to the System Memory using the Scatter Gather interface during a descriptor update.

Video DMA

The AXI4 protocol Video DMA (VDMA) provides a high bandwidth solution for Video applications. It is a similar implementation to the Ethernet DMA solution.

Figure 2-26 shows a top-level AXI4 VDMA block diagram.

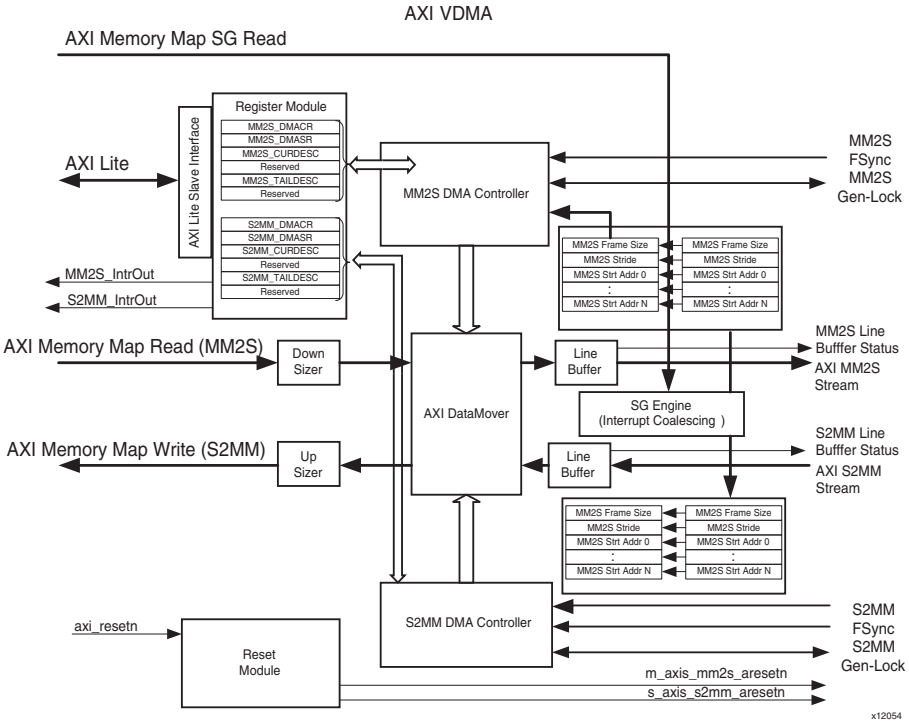


Figure 2-26: **AXI VDMA High-Level Block Diagram**

Figure 2-27 illustrates a typical system architecture for the AXI VDMA.

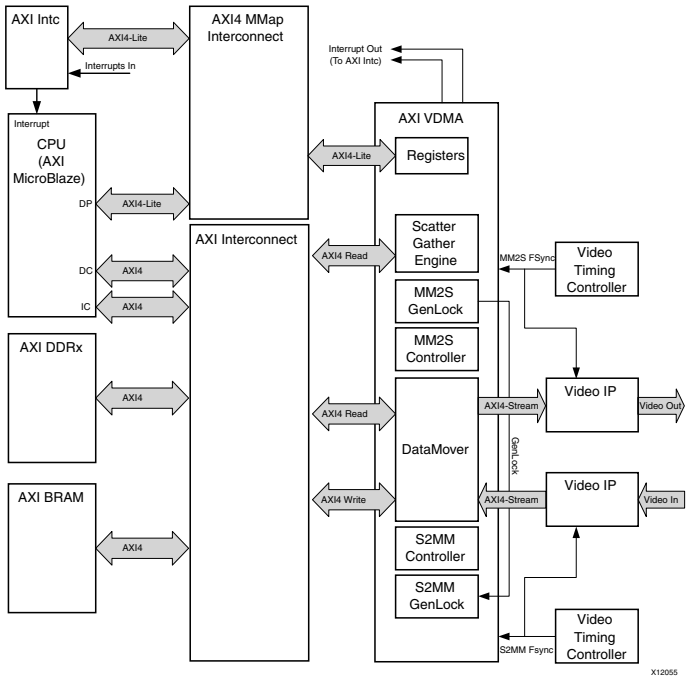


Figure 2-27: **Typical Use Case for AXI VDMA and Video IP**

AXI VDMA Summary

The AXI VDMA engine provides high performance direct memory access between system memory and AXI4-Stream type target peripherals. The AXI VDMA provides Scatter Gather (SG) capabilities also, which allows the CPU to offload transfer control and execution to hardware automation. The AXI VDMA and the SG engines are built around the AXI DataMover helper core which is the fundamental bridging element between AXI4-Stream and AXI4 memory mapped buses.

AXI VDMA provides:

- Circular frame buffer access for up to 32 frame buffers and provides the tools to transfer portions of video frames or full video frames.
- The ability to "park" on a frame, allowing the same video frame data to be transferred repeatedly.
- Independent frame synchronization and an independent AXI clock, allowing each channel to operate on a different frame rate and different pixel rate. To maintain synchronization between two independently functioning AXI VDMA channels, there is an optional *Gen-Lock* synchronization feature. Gen-Lock provides a method of synchronizing AXI VDMA slaves automatically to one or more AXI VDMA masters so the slave does not operate in the same video frame buffer space as the master.

In this mode, the slave channel skips or repeats a frame automatically. Either channel can be configured to be a Gen-Lock slave or a Gen-Lock master.

For video data transfer, the AXI4-Stream ports can be configured from 8 bits up to 1024 bits wide in multiples of 8. For configurations where the AXI4-Stream port is narrower than the associated AXI4 memory map port, the AXI VDMA upsizes or downsizes the data providing full bus width burst on the memory map side. It also supports an asynchronous mode of operation where all clocks are treated asynchronously.

VDMA AXI4 Interfaces

Table 2-3: **AXI VDMA Interfaces**

Interface	AXI Type	Data Width	Description
Control	AXI4-Lite slave	32	Accesses the AXI VDMA internal registers. This is generally used by the System Processor to control and monitor the AXI VDMA operations.
Scatter Gather	AXI4 master	32	An AXI4 memory mapped master that is used by the AXI VDMA to read DMA transfer descriptors from System Memory. Fetched Scatter Gather descriptors set up internal video transfer parameters for video transfers.
Data MM Read	AXI4 Read master	32, 64, 128, 256, 512, 1024	Transfers payload data for operations moving data from the memory mapped side of the DMA to the Main Stream output side.
Data MM Write	AXI4 Write master	32, 64, 128, 256, 512, 1024	Transfers payload data for operations moving data from the Data Stream In interface of the DMA to the memory mapped side of the DMA.
Data Stream Out	AXI4-Stream master	8,16, 32, 64, 128, 256, 512, 1024	Transfers data read by the Data MM Read interface to the target receiver IP using the AXI4-Stream protocol.
Data Stream In	AXI4-Stream slave	8, 16, 32, 64, 128, 256, 512, 1024	Receives data from the source IP using the AXI4-Stream protocol. The data received is then transferred to the Memory Map system using the Data MM Write Interface.

Memory Control IP and the Memory Interface Generator

There are two DDRx (SDRAM) AXI memory controllers available in the IP catalog.

Because the Virtex-6 and Spartan-6 devices have natively different memory control mechanisms (Virtex-6 uses a fabric-based controller and Spartan-6 has an on-chip Memory Control Block (MCB)), the description of memory control is necessarily device-specific. The following subsections describe AXI memory control by Virtex-6 and Spartan-6 devices.

The Virtex-6 and Spartan-6 memory controllers are available in two different tool packages:

- In EDK, as the `axi_v6_ddrx` or the `axi_s6_ddrx` memory controller core.
- In the CORE™ Generator interface, through the Memory Interface Generator (MIG) tool.

The underlying HDL code between the two packages is the same with different wrappers.

The flexibility of the AXI4 interface allows easy adaptation to both controller types.

Virtex-6

The Virtex-6 memory controller solution is provided by the Memory Interface Generator (MIG) tool and is updated with an optional AXI4 interface.

This solution is available through EDK also, with an AXI4-only interface as the `axi_v6_ddrx` memory controller.

The `axi_v6_ddrx` memory controller uses the same Hardware Design Language (HDL) logic and uses the same GUI, but is packaged for EDK processor support through XPS. The Virtex-6 memory controller is adapted with an AXI4 Slave Interface (SI) through an AXI4 to User Interface (UI) bridge. The AXI4-to-UI bridge converts the AXI4 slave transactions to the MIG Virtex-6 UI protocol. This supports the same options that were previously available in the Virtex-6 memory solution.

The optimal AXI4 data width is the same as the UI data width, which is four times the memory data width. The AXI4 memory interface data width can be smaller than the UI interface but is not recommended because it would result in a higher area, lower timing/performance core to support the width conversion.

The AXI4 interface maps transactions over to the UI by breaking each of the AXI4 transactions into smaller stride, memory-sized transactions. The Virtex-6 memory controller then handles the bank/row management for higher memory utilization.

Figure 2-28 shows a block diagram of the Virtex-6 memory solution with the AXI4 interface.

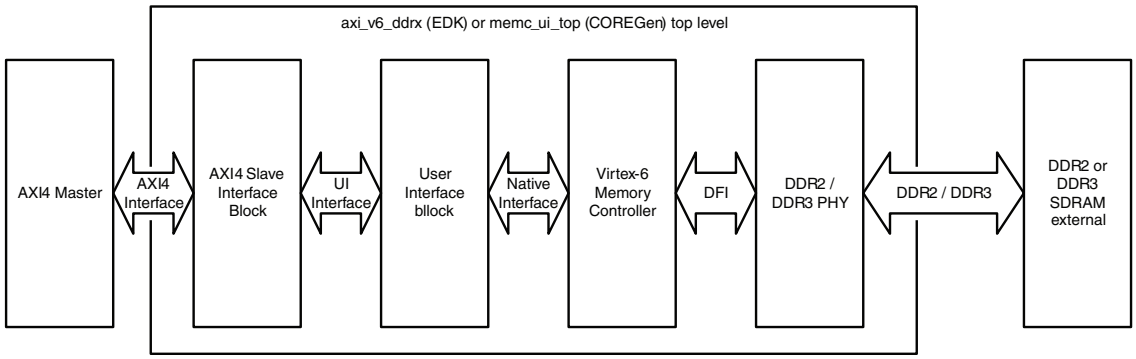


Figure 2-28: **Virtex-6 Memory Control Block Diagram**

Spartan-6 Memory Control Block

The Spartan-6 device uses the hard Memory Control Block (MCB) primitive native to that device. The Spartan-6 MCB solution was adapted with an AXI4 memory mapped Slave Interface (SI).

To handle AXI4 transactions to external memory on Spartan-6 architectures requires a bridge to convert the AXI4 transactions to the MCB user interface.

Because of the similarities between the two interfaces, the AXI4 SI can be configured to be "lightweight," by connecting a master that does not issue narrow bursts and has the same native data width as the configured MCB interface. The AXI4 bridge:

- Converts AXI4 incremental (`INCR`) commands to MCB commands in a 1:1 fashion for transfers that are 16 beats or less.
- Breaks down AXI4 transfers greater than 16 beats into 16-beat maximum transactions sent over the MCB protocol.

This allows a balance between performance and latency in multi-ported systems. AXI4 `WRAP` commands can be broken into two MCB transactions to handle the wraps on the MCB interface, which does not natively support `WRAP` commands natively.

The `axi_s6_ddrx` core and Spartan-6 AXI MIG core from CORE Generator support all native port configurations of the MCB including 32, 64, and 128 bit wide interfaces with up to 6 ports (depending on MCB port configuration).

Figure 2-29 shows a block diagram of the AXI Spartan-6 memory solution.

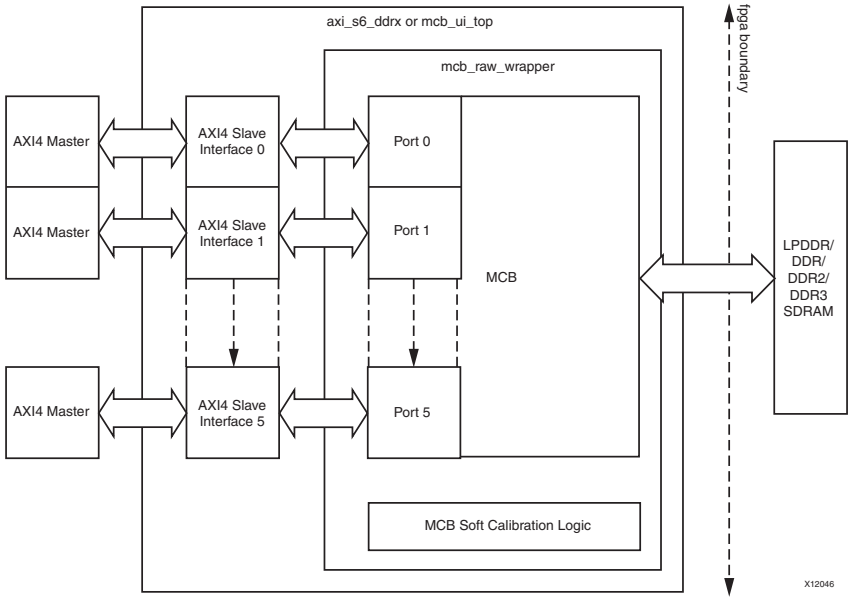


Figure 2-29: **Spartan-6 Memory Solution Block Diagram**

For more detail on memory control, refer to the memory website documents at http://www.xilinx.com/products/technology/memory-solutions/index.htm.

AXI Feature Adoption in Xilinx FPGAs

Introduction

This chapter describes how Xilinx® utilizes specific features from the AXI standard in Xilinx IP to familiarize the IP designer with various AXI-related design and integration choices.

Memory Mapped IP Feature Adoption and Support

Xilinx has implemented and is supporting a rich feature set from AXI4 and AXI4-Lite to facilitate interoperability among memory mapped IP from Xilinx developers, individual users, and third-party partners.

Table 3-1 lists the key aspects of Xilinx AXI4 and AXI4-Lite adoption, and the level to which Xilinx IP has support for, and implemented features of, the AXI4 specification.

Table 3-1: **Xilinx AXI4 and AXI4-Lite Feature Adoption and Support**

AXI Feature	Xilinx IP Support
`READY/VALIDY` Handshake	Full forward and reverse direction flow control of AXI protocol-defined `READY/VALID` handshake.
Transfer Length	AXI4 memory mapped burst lengths of: · 1 to 256 beats for incrementing bursts · 1 to 16 beats for wrap bursts Fixed bursts should not be used with Xilinx IP. Fixed bursts do not cause handshake-level protocol violations, but this behavior is undefined or can be aliased into an incrementing burst.
Transfer Size / Data Width	IP can be defined with native data widths of 32, 64, 128, 256, 512, and 1024 bits wide. For AXI4-Lite, the supported data width is 32 bits only. The use of AXI4 narrow bursts is supported but is not recommended. Use of narrow bursts can decrease system performance and increase system size. Where Xilinx IP of different widths need to communicate with each other, the AXI Interconnect provides data width conversion features.
Read/Write only	The use of read/write, read-only, or write-only interfaces. Many IP, including the AXI Interconnect, perform logic optimizations when an interface is configured to be Read-only or Write-only.

Table 3-1: **Xilinx AXI4 and AXI4-Lite Feature Adoption and Support *(Cont'd)***

AXI Feature	Xilinx IP Support
AXI3 vs. AXI4	Designed to support AXI4 natively. Where AXI3 interoperability is required, the AXI Interconnect contains the necessary conversion logic to allow AXI3 and AXI4 devices to connect. AXI3 write interleaving is *not* supported and should not be used with Xilinx IP. ***Note:*** The AXI3 write Interleaving feature was removed from the AXI4 specification.
Lock / Exclusive Access	No support for locked transfers. Xilinx infrastructure IP can pass exclusive access transactions across a system, but Xilinx IP does not support the exclusive access feature. All exclusive access requests result in "OK" responses.
Protection/Cache Bits	Infrastructure IP passes protection and cache bits across a system, but Endpoint IP generally do not contain support for dynamic protection or cache bits. · Protections bits should be constant at 000 signifying a constantly secure transaction type. · Cache bits should generally be constant at 0011 signifying a bufferable and modifiable transaction. This provides greater flexibility in the infrastructure IP to transport and modify transactions passing through the system for greater performance.
Quality of Service (QoS) Bits	Infrastructure IP passes QoS bits across a system. Endpoint IP generally ignores the QoS bits.
REGION Bits	The Xilinx AXI interconnect generates REGION bits based upon the Base/High address decoder ranges defined in the address map for the AXI interconnect. Xilinx infrastructure IP, such as register slices, pass region bits across a system. Some Endpoint slave IP supporting multiple address ranges might use region bits to avoid redundant address decoders. AXI Master Endpoint IP do not generate REGION bits.
User Bits	Infrastructure IP passes user bits across a system, but Endpoint IP generally ignores user bits. The use of user bits is discouraged in general purpose IP due to interoperability concerns. However the facility to transfer user bits around a system allows special purpose custom systems to be built that require additional transaction-based sideband signaling. An example use of USER bits would be for transferring parity or debug information.
Reset	Xilinx IP generally deasserts all VALID and READY outputs within eight cycles of reset, and have a reset pulse width requirement of 16 cycles or greater. Holding AXI ARESET*N* asserted for 16 cycles of the slowest AXI clock is generally a sufficient reset pulse width for Xilinx IP. DSP IP has a requirement of 2 cycles for ARESET*N* on the AXI4-Stream interface.
Low Power Interface	***Not Supported***. The optional AXI low power interfaces, CSYSREQ, CSYSACK, and CACTIVE are not present on IP interfaces.

AXI4-Stream Adoption and Support

To facilitate interoperability, Xilinx IP adopts a consistent set of AXI4-Stream protocol usage guidelines. This section provides an overview of the adoption of AXI4-Stream signals, numerical data type representation, and AXI4-Stream protocol usage across DSP, Wireless, and Video IP. For more information on how to construct systems by configuring your AXI4-Stream-based IP designs for interoperability, see Chapter 6, "AXI4-Stream IP Interoperability: Tips and Hints."

AXI4-Stream Signals

Table 3-2 lists the AXI4-Stream signals, status, and notes on usage.

Table 3-2: **AXI4-Stream Signals**

Signal	Status	Notes
TVALID	Required	
TREADY	Optional	`TREADY` is optional, but highly recommended.
TDATA	Optional	
TSTRB	Optional	Not typically used by endpoint IP; available for sparse stream signalling. ***Note:*** For marking packet remainders, `TKEEP` use rather than `TSTRB`.
TKEEP	Absent	Null bytes are only used for signaling packet remainders. Leading or intermediate Null bytes are generally not supported.
TLAST	Optional	
TID	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TDEST	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TUSER	Optional	

Numerical Data in an AXI4-Stream

An AXI4-Stream channel is a method of transferring data from a master to a slave. To enable interoperability, both the master and slave must agree on the correct interpretation of those bits.

In Xilinx IP, streaming interfaces are frequently used to transfer numerical data representing sampled physical quantities (for example: video pixel data, audio data, and signal processing data). Interoperability support requires a consistent interpretation of numerical data.

Numerical data streams within Xilinx IP are defined in terms of logical and physical views. This is especially important to understand in DSP applications where information can be transferred essentially as data structures.

- The logical view describes the application-specific organization of the data.
- The physical view describes how the logical view is mapped to bits and the underlying AXI4-Stream signals.

Simple vectors of values represent numerical data at the logical level. Individual values can be real or complex quantities depending on the application. Similarly the number of elements in the vector will be application-specific.

At the physical level, the logical view must be mapped to physical wires of the interface. Logical values are represented physically by a fundamental base unit of bit width *N*, where *N* is application-specific. In general:

- *N* bits are interpreted as a fixed point quantity, but floating point quantities are also permitted.
- Real values are represented using a single base unit.
- Complex values are represented as a pair of base units signifying the real component followed by the imaginary component.

To aid interoperability, all logical values within a stream are represented using base units of identical bit width.

Before mapping to the AXI4-Stream signal, TDATA, the *N* bits of each base unit are rounded up to a whole number of bytes. As examples:

- A base unit with *N*=12 is packed into 16 bits of TDATA.
- A base unit with *N*=20 is packed into 24 bits of TDATA.

The AXI4-Stream protocol requires that TDATA ports of the IP have a width in multiples of 8. It is a specification violation to define an AXI4-Stream IP with a TDATA port width that is not a multiple of 8, therefore, it is a requirement to round up TDATA widths to byte multiples. This simplifies interfacing with memory-orientated systems, and also allows the use of AXI infrastructure IP, such as the AXI Interconnect, to perform upsizing and downsizing.

By convention, the additional packing bits are ignored at the input to a slave; they therefore use no additional resources and are removed by the back-end tools. To simplify diagnostics, masters drive the unused bits in a representative manner, as follows:

- Unsigned quantities are zero-extended (the unused bits are zero).
- Signed quantities are sign-extended (the unused bits are copies of the sign bit).

The width of TDATA can allow multiple base units to be transferred in parallel in the same cycle; for example, if the base unit is packed into 16 bits and TDATA signal width was 64 bits, four base units could be transferred in parallel, corresponding to four scalar values or two complex values. Base units forming the logical vector are mapped first spatially (across TDATA) and then temporally (across consecutive transfers of TDATA).

Deciding whether multiple sub-fields of data (that are not byte multiples) should be concatenated together before or after alignment to byte boundaries is generally determined by considering how *atomic* is the information. Atomic information is data that can be interpreted on its own whereas non-atomic information is incomplete for the purpose of interpreting the data.

For example, atomic data can consist of all the bits of information in a floating point number. However, the exponent bits in the floating point number alone would not be atomic. When packing information into TDATA, generally non-atomic bits of data are concatenated together (regardless of bit width) until they form atomic units. The atomic units are then aligned to byte boundaries using pad bits where necessary.

Real Scalar Data Example

A stream of scalar values can use two, equally valid, uses of the optional TLAST signal. This is illustrated in the following example.

Consider a numerical stream with characteristics of the following values:

Logical type	Unsigned Real
Logical vector length	1 (for example, scalar value)
Physical base unit	12-bit fixed point
Physical base unit packed width	16 bits
Physical TDATA width	16 bits

This would be represented as an AXI4-Stream, as shown in Figure 3-1.

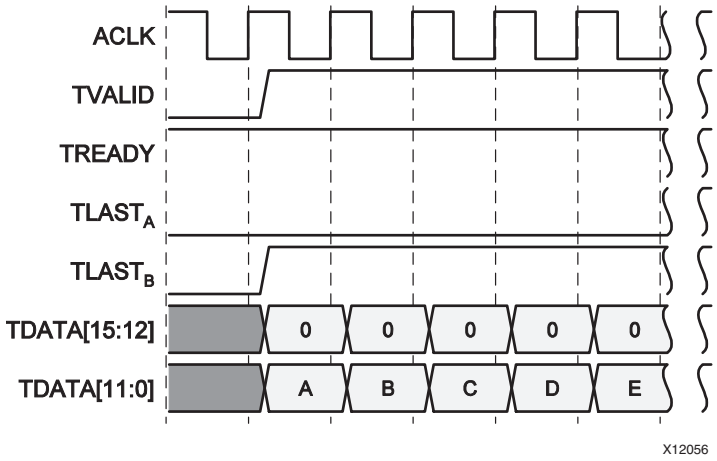


Figure 3-1: **Real Scalar (Unsigned) Data Example in an AXI4-Stream**

Scalar values can be considered as not packetized at all, in which case TLAST can legitimately be driven active-Low ($TLAST_A$); and, because TLAST is optional, it could be removed entirely from the channel also.

Alternatively, scalar values can also be considered as vectors of unity length, in which case TLAST should be driven active-High ($TLAST_B$). As the value type is unsigned, the unused packing bits are driven 0 (zero extended).

Similarly, for signed data the unused packing bits are driven with the sign bits (sign-extended), as shown in Figure 3-2.

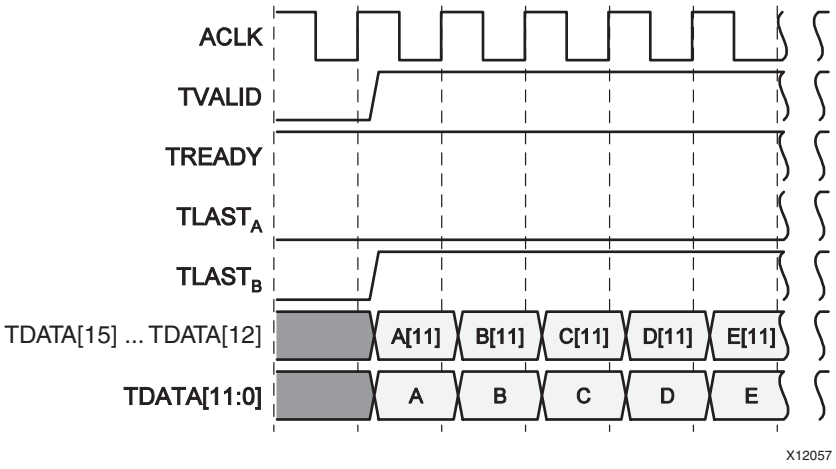


Figure 3-2: **Alternative (Sign-Extended) Scalar Value Example**

Complex Scalar Data Example

Consider a numerical stream with the following characteristics:

Logical type	Signed Complex
Logical vector length	1 (for example: scalar)
Physical base unit	12 bit fixed point
Physical base unit packed width	16 bits
Physical TDATA width	16 bits

This would be represented as an AXI4-Stream, as shown in Figure 3-3:

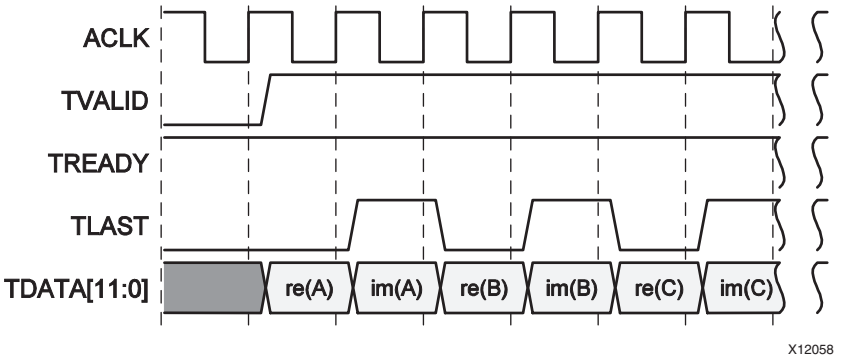


Figure 3-3: **Complex Scalar Data Example in AXI4-Stream**

Where re(X) and im(X) represent the real and imaginary components of X respectively.

Note: For simplicity, sign extension into `TDATA`[15:12] is not illustrated here. A complex value is transferred every two clock cycles.

The same data can be similarly represented on a channel with a TDATA signal width of 32 bits; the wider bus allows a complex value to be transferred every clock cycle, as shown in Figure 3-4.

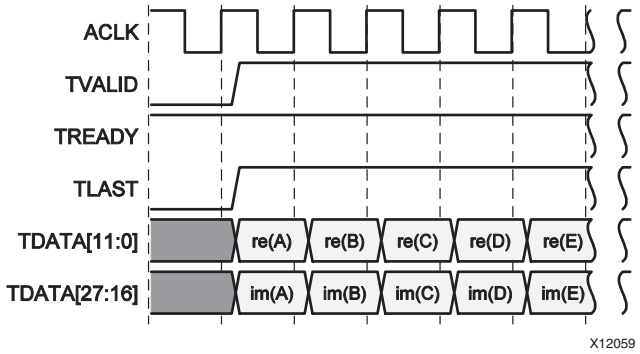


Figure 3-4: **Complex Scalar Example with 32-Bit TDATA Signal**

The two representations in the preceding figures of the same data (serial and parallel) show that data representation can be tailored to specific system requirements.

For example, a:

- High throughput processing engine such as a Fast Fourier Transform (FFT) might favor the parallel form
- MAC-based Finite Impulse Response (FIR) might favor the serial form, thus enabling Time Division Multiplexing (TDM) data path sharing

To enable interoperability of sub-systems with differing representation, you need a conversion mechanism. This representation was chosen to enable simple conversions using a standard AXI infrastructure IP:

- Use an AXI4-Stream-based upsizer to convert the serial form to the parallel form.
- Use an AXI4-Stream-based downsizer to convert the parallel form to the serial form.

You can implement an AXI4-Stream-based upsizer or downsizer by using the width conversion feature of the AXI4-Stream Interconnect. For more information see: http://www.xilinx.com/products/intellectual-property/axi4-stream_interconnect.htm.

Vector Data Example

Consider the example of a numerical stream with the following characteristics:

Logical type	Signed Complex
Logical vector length	4
Physical base unit	12 bit fixed point
Physical base unit packed width	16 bits
Physical TDATA width	16 bits

Figure 3-5 shows the AXI4-Stream representation of that numerical stream:

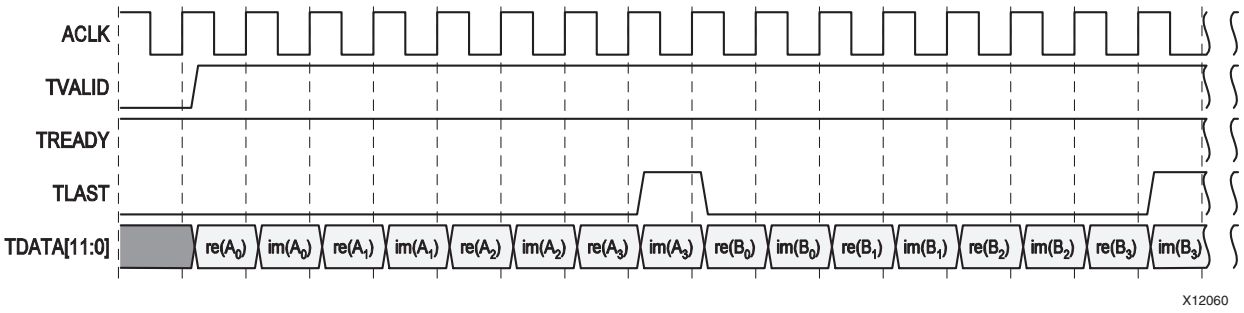


Figure 3-5: **Numerical Stream Example**

As for the scalar case, the same data can be represented on a channel with TDATA width of 32 bits, as shown in Figure 3-6.

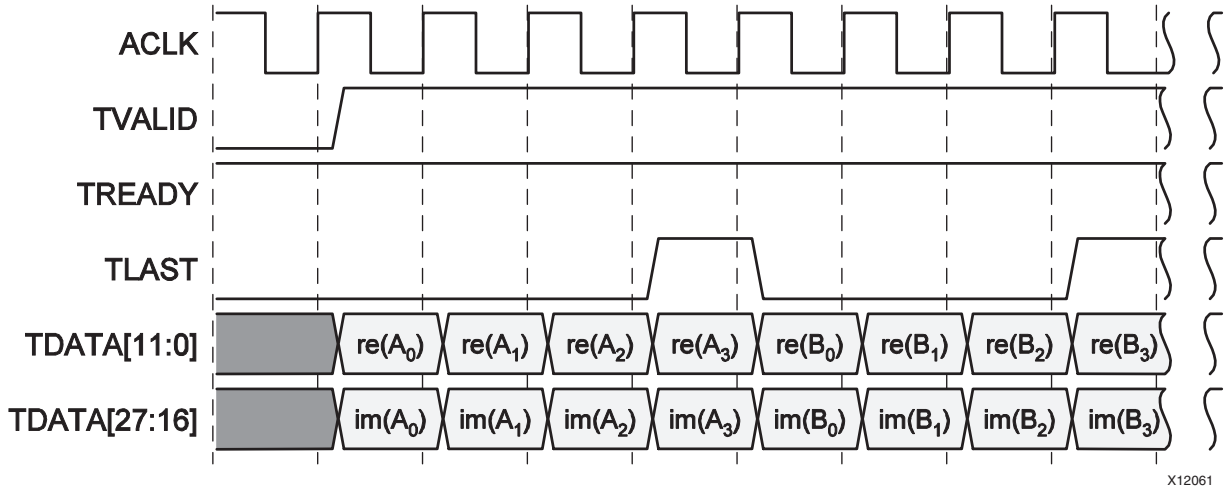


Figure 3-6: **AXI4-Stream Scalar Example**

The degree of parallelism can be increased further for a channel with TDATA width of 64 bits, as shown in Figure 3-7.

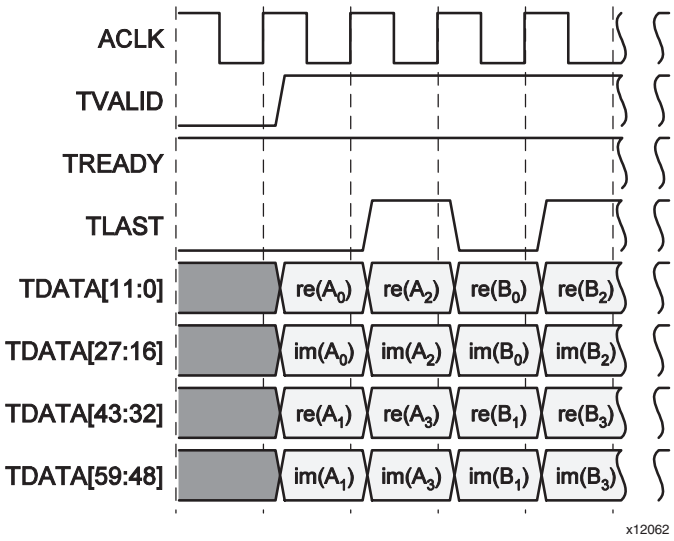


Figure 3-7: **TDATA Example with 64-Bits**

Full parallelism can be achieved with TDATA width of 128 bits, as shown in Figure 3-8.

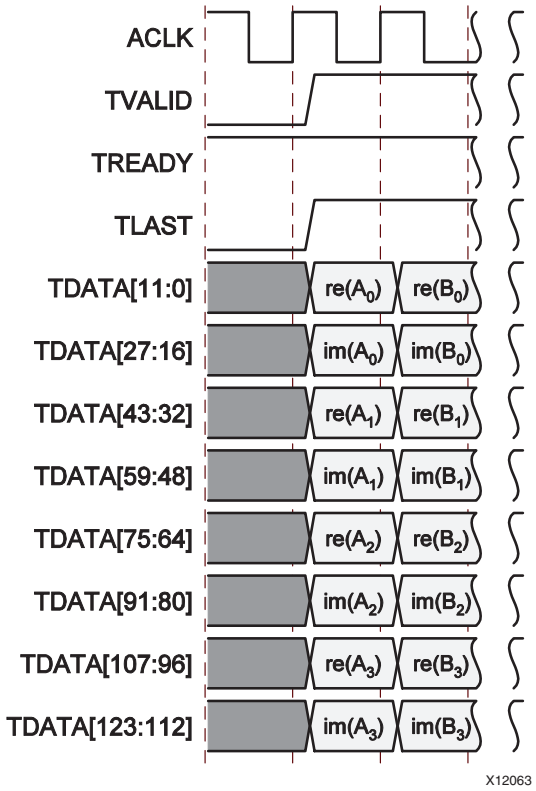


Figure 3-8: **128 Bit TDATA Example**

As shown in the preceding figures, for the scalar data there are multiple representations that you can customize to the application.

Similarly, AXI4-Stream upsizers and downsizers can be used for conversion.

Packets and NULL Bytes

The AXI4-Stream protocol lets you specify packet boundaries using the optional TLAST signal.

In many situations this is sufficient; however, by definition, the TLAST signal indicates the size of the data at the end of the packet, while many IP require packet size at the beginning. In such situations, where packet size must be specified at the beginning, the IP typically requires an alternative mechanism to provide that packet-size information. Streaming slave channels can therefore be divided into three categories:

- Slaves that do not require the interpretation of packet boundaries.

  There are slave channels that do not have any concept of packet boundaries, or when the size of packets do not affect the processing operation performed by the core.

Frequently, IP of this type provides a pass-through mechanism to allow TLAST to propagate from input to output with equal latency to the data.

- Slaves that require the TLAST signal to identify packet boundaries.

  These slaves channels are inherently packet-orientated, and can use TLAST as a packet size indicator. For example, a Cyclic Redundancy Check (CRC) core can calculate the CRC while data is being transferred, and upon detecting the TLAST signal, can verify that the CRC is correct.

- Slaves that do not require TLAST to identify packet boundaries.

  Some slave channels have an internal expectation of what size packets are required. For example, an FFT input packet size is always the same as the transform size. In these cases, TLAST would represent redundant information and also potentially introduce ambiguity into packet sizing (for example: what should an *N*-point FFT do when presented with an *N*-1 sample input packet.)

  To prevent this ambiguity, many Xilinx IP cores are designed to ignore TLAST on slave channels, and to use the explicit packet sizing information available to them. In these situations the core uses the required number of AXI transfers it is expecting regardless of TLAST. This typically greatly aides interoperability as the master and slave are not required to agree on when TLAST must be asserted.

  For example, consider an FIR followed by an *N*-point FFT. The FIR is a stream-based core and cannot natively generate a stream with TLAST asserted every *N* transfers. If the FFT is designed to ignore the incoming TLAST this is not an issue, and the system functions as expected. However, if the FFT did require TLAST, an intermediate "re-framing" core would be required to introduce the required signalling.

- For Packetized Data, TKEEP might be necessary to signal packet remainders. When the TDATA width is greater than the atomic size (minimum data granularity) of the stream, a remainder is possible because there might not be enough data bytes to fill an entire data beat. The only supported use of TKEEP for Xilinx endpoint IP is for packet remainder signaling and deasserted TKEEP bits (which is called "Null Bytes" in the *AXI4-Stream Protocol v1.0*) are only present in a data beat with TLAST asserted.
  For non-packetized continuous streams or packetized streams where the data width is the same size or smaller than the atomic size of data, there is no need for TKEEP. This generally follows the "Continuous Aligned Stream" model described in the AXI4-Stream protocol.

  The AXI4-Stream protocol describes the usage for TKEEP to encode trailing null bytes to preserve packet lengths after size conversion, especially after upsizing an odd length packet. This usage of TKEEP essentially encodes the remainder bytes after the end of a packet which is an artifact of upsizing a packet beyond the atomic size of the data.

  Xilinx AXI master IP do not to generate any packets that have trailing transfers with all TKEEP bits deasserted.

This guideline maximizes compatibility and throughput because Xilinx IP does not originate packets containing trailing transfers with all TKEEP bits deasserted. Any deasserted TKEEP bits must be associated with TLAST = 1 in the same data beat to signal the byte location of the last data byte in the packet.

Xilinx AXI slave IP are generally not designed to be tolerant of receiving packets that have trailing transfers with all TKEEP bits deasserted. Slave IP that have TKEEP inputs only sample TKEEP with TLAST is asserted to determine the packet remainder bytes. In general if Xilinx IP are used in the system with other IP designed for "Continuous Aligned Streams" as described in the AXI4-Stream specification, trailing transfers with all TKEEP bits deasserted will not occur.

All streams entering into a system of Xilinx IP must be fully packed upon entry in the system (no leading or intermediate null bytes) in which case arbitrary size conversion will only introduce TKEEP for packet remainder encoding and will not result in data beats where all TKEEP bits are deasserted.

Sideband Signals

The AXI4-Stream interface protocol allows passing sideband signals using the TUSER bus.

From an interoperability perspective, use of TUSER on an AXI4-Stream channel is an issue as both master and Slave must now not only have the same interpretation of TDATA, but also of TUSER.

Generally, Xilinx IP uses the TUSER field only to augment the TDATA field with information that could prove useful, but ultimately can be ignored. Ignoring TUSER could result in some loss of information, but the TDATA field still has some meaning.

For example, an FFT core implementation could use a TUSER output to indicate block exponents to apply to the TDATA bus; if TUSER was ignored, the exponent scaling factor would be lost, but TDATA would still contain unscaled transform data.

Events

An event signal is a single wire interface used by a core to indicate that some specific condition exists (for example: an input parameter is invalid, a buffer is empty or nearly full, or the core is waiting on some additional information). Events are asserted while the condition is present, and are deasserted once the condition passes, and exhibit no latching behavior. Depending on the core and how it is used in a system, an asserted event might indicate an error, a warning, or information. Event signals can be viewed as AXI4-Stream channels with an VALID signal only, without any optional signals. Event signals can also be considered out-of-band information and treated like generic flags, interrupts, or status signals.

Events can be used in many different ways:

- **Ignored**:

  Unless explicitly stated otherwise, a system can ignore all event conditions. In general, a core continues to operate while an event is asserted, although potentially in some degraded manner.

- **As Interrupts or GPIOs**:

  An event signal might be connected to a processor using a suitable interrupt controller or general purpose I/O controller. System software is then free to respond to events as necessary.

- **As Simulation Diagnostic**:

  Events can be useful during hardware simulation. They can indicate interoperability issues between masters and slaves, or indicate misinterpretation of how subsystems interact.

- **As Hardware Diagnostic**:

  Similarly, events can be useful during hardware diagnostic. You can route events signals to diagnostic LED or test points, or connect them to the ChipScope™ Pro Analyzer.

As a system moves from development through integration to release, confidence in its operation is gained; as confidence increases the need for events could diminish.

For example, during development simulations, events can be actively monitored to ensure a system is operating as expected. During hardware integration, events might be monitored only if unexpected behavior occurs, while in a fully-tested system, it might be reasonable to ignore events.

Note: Events signals are asserted when the core detects the condition described by the event; depending on internal core latency and buffering, this could be an indeterminate time after the inputs that caused the event were presented to the core.

TLAST Events

Some slave channels do not require a `TLAST` signal to indicate packet boundaries. In such cases, the core has a pair of events to indicate any discrepancy between the presented `TLAST` and the internal concept of packet boundaries:

- **Missing TLAST:** `TLAST` is not asserted when expected by the core.
- **Unexpected TLAST:** `TLAST` is asserted when not expected by the core.

Depending on the system design these events might or might not indicate potential problems.

For example, consider an FFT core used as a coprocessor to a CPU where data is streamed to the core using a packet-orientated DMA engine.

The DMA engine can be configured to send a contiguous region of memory of a given length to the FFT core, and to correctly assert `TLAST` at the end of the packet. The system software can elect to use this coprocessor in a number of ways:

- **Single Transforms**:

  The simplest mode of operation is for the FFT core and the DMA engine to operate in a lockstep manner. If the FFT core is configured to perform an N point transform, then the DMA engine should be configured to provide packets of N complex samples.

  If a software or hardware bug is introduced that breaks this relationship, the FFT core will detect `TLAST` mismatches and assert the appropriate event; in this case indicating error conditions.

- **Grouped Transforms**:

  Typically, for each packet transferred by the DMA engine, a descriptor is required containing start address, length, and flags; generating descriptors and sending them to the engine requires effort from the host CPU. If the size of transform is short and the number of transforms is high, the overhead of descriptor management might begin to overcome the advantage of offloading processing to the FFT core.

  One solution is for the CPU to group transforms into a single DMA operation. For example, if the FFT core is configured for 32-point transforms, the CPU could group 64 individual transforms into a single DMA operation. The DMA engine generates a single 2048 sample packet containing data for the 64 transforms; however, as the DMA engine is only sending a single packet, only the data for the last transform has a correctly placed `TLAST`. The FFT core would report 63 individual '`missing TLAST`' events for the grouped operation. In this case the events are entirely expected and do not indicate an error condition.

  In the example case, the '`unexpected TLAST`' event should not assert during normal operation. At no point should a DMA transfer occur where `TLAST` does not align with the end of an FFT transform. However, as for the described single transform example case, a software or hardware error could result in this event being asserted. For example, if the transform size is incorrectly changed in the middle of the grouped packet, an error would occur.

- **Streaming Transforms**:

  For large transforms it might be difficult to arrange to hold the entire input packet in a single contiguous region of memory.

  In such cases it might be necessary to send data to the FFT core using multiple smaller DMA transfers, each completing with a `TLAST` signal.

Depending on how the CPU manages DMA transfers, it is possible that the `TLAST` signal never aligns correctly with the internal concept of packet boundaries within the FFT core.

The FFT core would therefore assert both '`missing TLAST`' and '`unexpected TLAST`' events as appropriate while the data is transferring. In this example case, both events are entirely expected, and do not indicate an error condition.

DSP and Wireless IP: AXI Feature Adoption

An individual AXI4-Stream slave channel can be categorized as either a blocking or a non-blocking channel.

A slave channel is blocking when some operation of the core is inhibited until a transaction occurs on that channel.

For example, consider an FFT core that features two slave channels; a data channel and a control channel. The data channel transfers input data, and the control channel transfers control packets indicating how the input data should be processed (like transform size).

The control channel can be designed to be either blocking or non-blocking:

- A blocking case performs a transform only when both a control packet and a data packet are presented to the core.
- A non-blocking case performs a transform with just a data packet, with the core reusing previous control information.

There are numerous tradeoffs related to the use of blocking versus non-blocking interfaces:

Feature	Blocking	Non-blocking
Synchronization	Automatic Core operates on transactions; for example, one control packet and one input data packet produces one output data packet.	Not automatic System designer must ensure that control information arrives at the core before the data to which it applies.
Signaling Overhead	Small Control information must be transferred even if it does not change.	Minimized Control information need be transferred only if it changes.
Connectivity	Simple Data flows through a system of blocking cores as and when required; automatic synchronization ensures that control and data remain in step.	Complex System designer must manage the flow of data and control flow though a system of non-blocking cores.
Resource Overhead	Small Cores typically require small additional buffers and state machines to provide blocking behavior.	None Cores typically require no additional resources to provide non-blocking behavior.

Generally, the simplicity of using blocking channels outweighs the penalties.

Note: The distinction between blocking and non-blocking behavior is a characteristic of how a core uses data and control presented to it; it does not necessarily have a direct influence on how a core drives `TREADY` on its slave channels. For example, a core might feature internal buffers on slave channels, in which case `TREADY` is asserted while the buffer has free space.

Note: In many cases, DSP and Wireless IP have base units that do not usually fall on the 8 bit (Byte) boundaries. Refer to the Numerical Data in an AXI4-Stream, page 53 for information on how to handle data that does not fall on byte boundaries.

Video IP: AXI Feature Adoption

In the video domain, there are established signals that are used in many standards to transmit data across video communication channels. These signals include video data and synchronization for proper communication and flow control.

Typically, video signals are propagated using several processing cores and frame buffers along which video resolution, frame rate, and formatting (such as interlaced to de-interlaced) can change.

Processing cores can:

- Process a single stream (the Xilinx Image Statistics core)
- Process and generate a single stream (most Xilinx Image Processing cores)
- Process multiple streams to generate a single stream (the Xilinx On Screen Display (OSD) core)
- Process multiple streams to generate multiple streams (the Xilinx Motion Adaptive Noice Reduction (MANR) core)

Video data enters the system through the input interface and exits the system through a similar interface, which is, in many cases, connected to a monitor for display of the processed video sequence. In a complex video system, the IP cores provide a register interface that is used for setup and control by a central managing microprocessor. This type of system is supported by Xilinx design tools such as the Embedded Development Kit (EDK) using the Xilinx MicroBlaze™ processor.

Xilinx Video IP cores are used in a variety of video and imaging applications and a wide range of markets, from Industrial, Scientific, and Medical (ISM), automotive and customer electronics to professional broadcast markets. The interface and protocol addresses the needs of multiple application domains.

IP using the AXI4-Stream Video Protocol provides a simple, versatile, high-performance point-to-point communication interface between video IP cores that is easy for video designers to use.

Using the industry standard AXI interface lets video cores connect to embedded processors and infrastructure IP. Based upon a well-defined, standard interface and protocol, video and system designers can leverage advanced Xilinx tools to connect video IP and to build video systems.

The following subsections provide the requirements, standards, recommendations, and guidelines for Xilinx Video IP design to adapt AXI4-Stream interfaces, and harmonize AXI4-Stream based Video IP development with AXI4-Stream based DSP IP, infrastructure IP, and tools development.

The subsections also provide the details for defining AXI4-Stream based Video IP interfaces, and describes the signals and protocols for transmitting video using the AXI4-Stream interface, its applicability to a wide range of video systems, and usage guidelines for interoperability.

This subsection also defines the:

- Set of AXI4-Stream signals used for video data exchange between IP cores
- Protocol of transmitting video frames using the AXI4-Stream interface
- List of supported data, such as RGB, 420 YCC, and the mapping of data to the `TDATA` bus (see Table 3-6, page 81).

Video systems follow the general pipelined processing chain, shown in Figure 3-9.

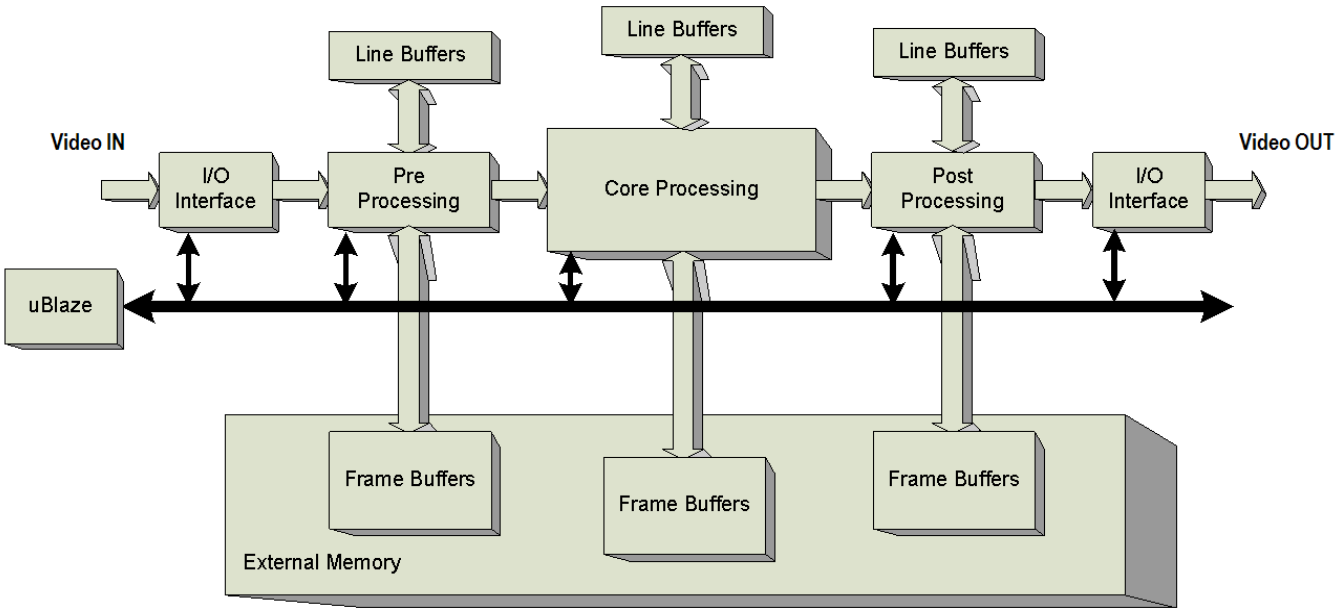


Figure 3-9: **Typical Video Processing System**

IP Using AXI4-Stream Video Protocol

AXI4-Stream only carries active video data, throttled by both the master and slave interfaces.

Note: Blank periods, audio, and ancillary data packets are not transferred using the AXI4-Stream Video Protocol.

Signaling Interface

Table 3-3 **lists the mandatory interface signal names and functions for the input (slave) side connectors.** The AXI4-Stream Signal Name column lists the mandatory, top-level IP port names.

Table 3-3: AXI4-Stream Video Protocol Input (Slave) Interface Signals

Function	Width	Direction	AXI4-Stream Signal Name	Video Specific Name
Video Data	8, 16, 24, 32, 40, 48, 56, 64	IN	s_axis_video_tdata	DATA
Valid	1	IN	s_axis_video_tvalid	VALID
Ready	1	OUT	s_axis_video_tready	READY
Start Of Frame	1	IN	s_axis_video_tuser	SOF
End Of Line	1	IN	s_axis_video_tlast	EOL

For IP with multiple AXI4-Stream input interfaces, append the `s_axis_video` signal with the index of the respective input AXI4-Stream, as shown in Figure 3-10.

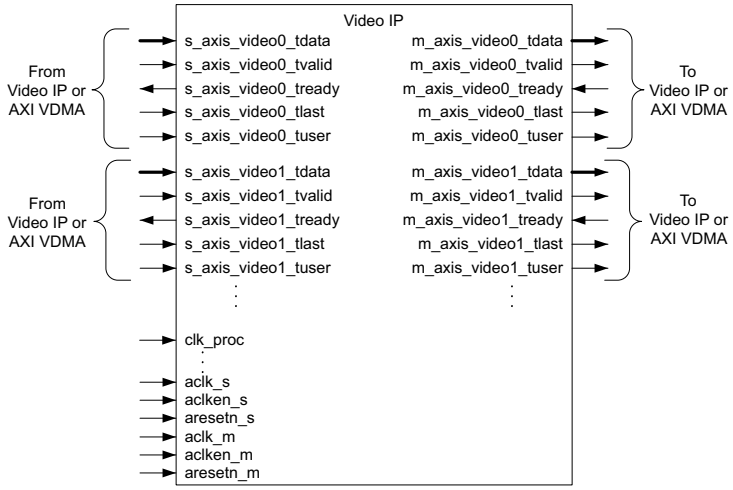


Figure 3-10: **Video IP with Multiple AXI4-Stream Slave (Input) and Master (Output) Interfaces**

Table 3-4 **lists the mandatory interface signal names and functions for the output (master) side signals**.

Similarly, for IP with multiple AXI4-Stream output interfaces, append the `m_axis_video_signal` with the index of the respective output AXI4-Stream, shown in Figure 3-10, page 69.

Table 3-4: **AXI4-Stream Video Protocol Output (Master) Interface Signals**

Function	Width	Direction	AXI4-Stream Signal Name	Video Specific Name
Video Data	8, 16, 24, 32, 40, 48, 56, 64	OUT	m_axis_video_tdata	DATA
Valid	1	OUT	m_axis_video_tvalid	VALID
Ready	1	IN	m_axis_video_tready	READY
Start Of Frame	1	OUT	m_axis_video_tuser	SOF
End Of Line	1	OUT	m_axis_video_tlast	EOL

The Video Specific Name column recommends short, descriptive signal names referring to AXI4-Stream ports, to be used in HDL code, timing diagrams, and test benches.

Clocking and ACLK

Each IP using the AXI4-Stream Video Protocol must reference a clock source. Directly-connected master and slave interfaces must be clocked by the same clock source. Any clock available to the IP can be used as the referenced clock source for an AXI interface.

The AXI protocol requires each component interface to use a single clock input signal, `ACLK`. **The `ACLK` signal is a mandatory pin on the IP core interface.**

The `ACLK` pin name either be appended or prefixed to designate clock functionality, such as `m0_axis_aclk`, or `aclk_out` for IP with multiple AXI4 interfaces using different clocks.

Interface input signals are sampled on the rising edge of `ACLK`. Output signal changes must occur after the rising edge of `ACLK`.

On Video IP interfaces, the `ACLK` pin is not part of the AXI4-Stream component interface; `ACLK` signals associated with AXI4-Stream component interfaces are provided to Video IP using one or multiple core clock signals. The clock signals can be shared by multiple AXI4-Stream interfaces and signal processing circuitry within the IP.

Signals in each component interface must be synchronous to one of the core clock signals, which are inputs to Video IP cores, but not directly part of the AXI4-Stream Video Protocol interface.

For example, if a core uses a single processing ACLK signal, to which all operations within the core are synchronous, the master and slave AXI4-Stream video interfaces should use this clock signal as their clock reference, as shown in Figure 3-11.

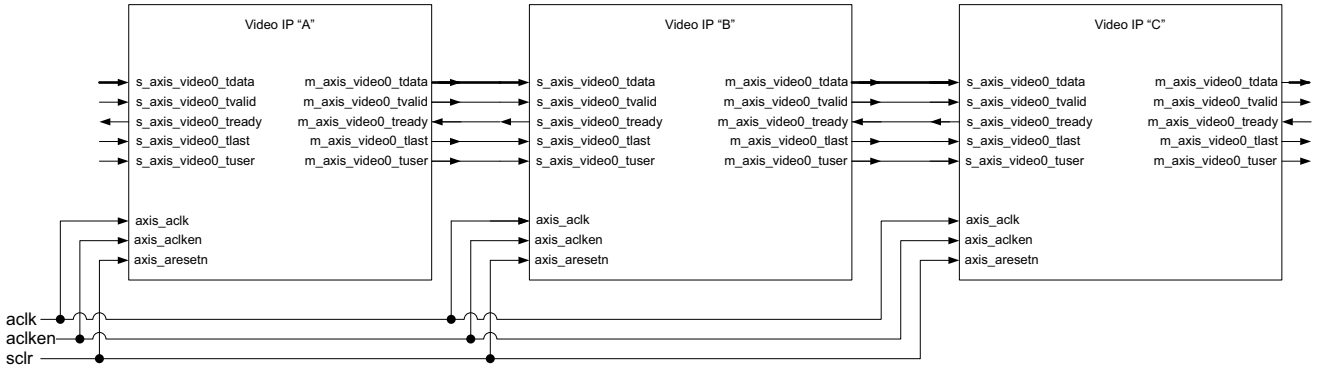


Figure 3-11: **Single Clock Domain Processing Pipeline - IP with One Processing Clock**

In the Figure 3-11 example, all AXI4-Stream interfaces operate synchronously to the ACLK processing clock. If the core designer intends to receive AXI4-Stream Video Protocol from a core operating in a different clock domain, and cross the clock domains within the IP, then the core must receive both clocks on the IP interface.

A Video IP core can contain multiple AXI4-Stream interfaces and multiple clocks, shown in Figure 3-12. Also, for system integration tools, the IP must contain metadata tags identifying clock domain associations.

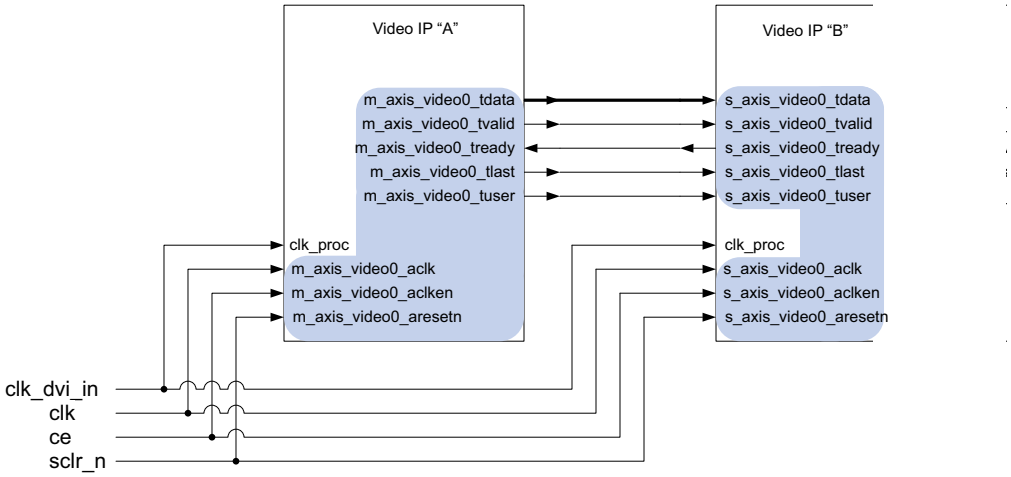


Figure 3-12: **IP Connection using Clock Domain other than the Processing Clock**

TDATA Structure

DATA bits are represented using the (*N*-1 downto 0) or [*N*-1:0] bit numbering convention. The components of implicit subfields of DATA shall be packed together tight; for example, a DW=10 bit RGB data packed together to 30 bits. If necessary, the packed data word can be zero padded with Most Significant Bit (MSBs) so the width of the resulting word is an integer multiple of 8.

Clock Enable, ACLKEN

The ACLKEN signal, associated with ACLK, is an optional, recommended pin on the IP core interface. For IP with multiple AXI4-Stream interfaces using different clocks, the name of the ACLKEN pin can be appended to designate clock association, such as ACLKEN_m0, or ACLKEN_in.

Note: When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal to prevent transaction errors.

Note: When two cores connect using AXI4-Stream interfaces, where only the master or the slave interface has an ACLKEN port, which is not permanently tied high, the two interfaces must be connected using the AXI4 FIFO core to avoid data corruption. See the *LogiCORE IP FIFO Generator (DS317)* [Ref 13] for information about the core.

Reset Requirements, ARESET*n*

Video IP cores must have two reset source types:

- Reset pins provided in conjunction with the corresponding clocks (hardware reset)
- The software reset option provided by the processor interface

An Active-low reset pin, ARESET*n*, associated with ACLK, is required on the IP core interface. For IP with multiple AXI4-Stream interfaces using different clocks, each clock domain can have corresponding reset signals. The name of the ARESET*n* pin can be appended to designate clock association, such as ARESET*n*_m0.

The ARESET*n* signal takes precedence over ACLKEN, cores with optional ACLKEN inputs that must reset when ARESET*n* is deasserted irrespective of the state of the associated ACLKEN input.

Note: When a system with multiple-clocks and corresponding reset signals are being reset, the reset generator has to ensure all reset signals are asserted/deasserted long enough that all interfaces and clock-domains in all IP cores are correctly reinitialized.

TKEEP and TSTRB

TKEEP and TSROBE are not used in IP using AXI4-Stream Video Protocol. When connecting to IP requiring TKEEP or TSTRB assignments, use the default values of TKEEP=1 and TSTRB=1.

The AXI4-Stream definition permits the insertion of placeholder (position) bytes and null bytes into the data stream; **however, AXI4-Stream compliant Video IP should only use the "Continuous Aligned Stream" mode of AXI4-Stream, and use packed data format and TDATA padded to integer (*N*) multiples of 8 bits** (see "Data Format" on page 78). For most video formats, all data bytes are always valid, when DATA is qualified by VALID.

For 420 encoded YCbCr / YUV data, only every second video line contains valid Chroma data. For the remaining lines, Luma is zero-padded.

TID

Video IP must use designated AXI4-Stream interfaces to transfer video and data streams; **therefore, TID is not used in IP using AXI4-Stream Video Protocol.** Video IP is not expected to forward a slave TID, or generate a TID, instead, the unconnected TID signal is expected to default to 0.

TDEST

The TDEST signal is not used in IP using AXI4-Stream Video Protocol. Video IP is not expected to forward a slave TDEST, or generate a TDEST, instead, the unconnected TDEST signal is expected to default to 0.

TUSER

TUSER bit 0, labeled Start of Frame (see Start of Frame Signal - SOF, page 76) **is the only AXI4-Stream signal used for video**. Other TUSER signal bits are not propagated by video cores.

Signaling Protocol

This section describes how you can use the interface signals and basic protocols of the AXI4-Stream specification to construct streaming interfaces to meet the needs of various video system applications. Generic AXI protocol signals are referenced using signal names reflecting their video specific function.

Channel Structure

The interface contains a set of handshake signals, VALID and READY, and a set of information-carrying signals, DATA, EOL, and SOF, that are conditioned by the handshake signals.

AXI4-Stream interface signals must operate in the same clock domain; however, the master and slave side can operate in different clock domains. In this case, proper clock-domain crossing logic must be employed when connecting the interfaces.

Figure 3-13 is a simplified diagram showing the use of an asynchronous FIFO for clock-domain crossing, omitting the enable and reset logic from the FIFO. A similar design can be used to connect to third party IP with no ACLKEN, or to a master with no TREADY input.

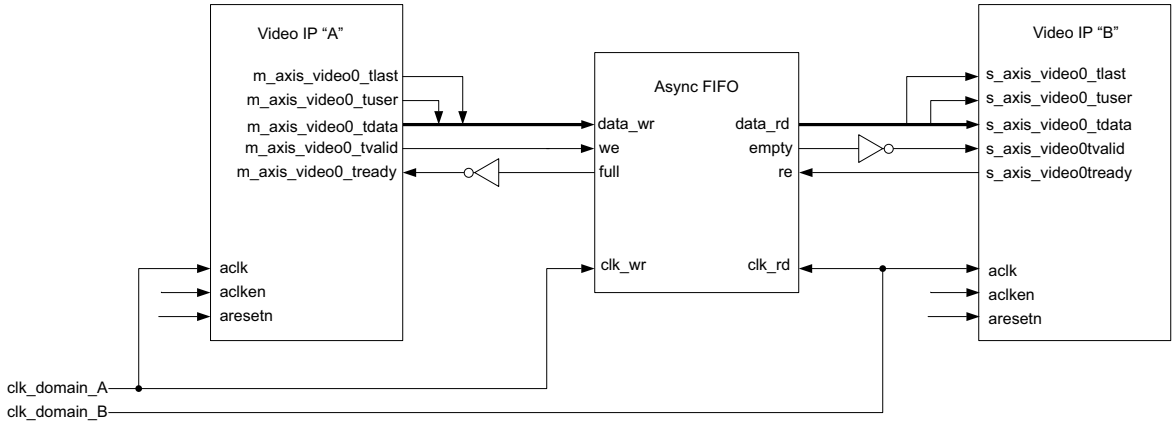


Figure 3-13: **Use of Asynchronous FIFO for Clock-Domain Crossing- No FIFO Enable or Reset Logic**

In EDK, the AXI4-Stream Interconnect IP can be used to simplify connecting AXI4-Stream interfaces in different clock domains, shown in Figure 3-13.

Note: In this protocol specification, for the sake of simplicity, both master and slave AXI4-Stream interfaces are assumed to operate in the same clock domain, synchronous to ACLK, with ACLKEN=1, and ARESET*n*=1.

For any given channel, signals propagate from the source (master) to the destination (slave) with the exception of the READY signal.

Any other information-carrying or control signals that need to propagate in the opposite direction must be part of a separate interface, READY is not used as a mechanism to transfer opposite direction information from a slave to a master.

READY/VALID Handshake

A valid transfer occurs when READY, VALID, ACLKEN, and ARESETn signals are high at the rising edge of ACLK, as shown in Figure 3-14, page 75. During valid transfers, DATA only carries active video data.

Note: Blank periods, audio, and ancillary data packets are not transferred in IP using the AXI4-Stream Video Protocol.

Guidelines on Driving VALID

After VALID is asserted, no other signals in the same channel (except READY) can change value until the transfer completes (the cycle after READY is asserted). After it is asserted, VALID can only be de-asserted after a transfer has completed (READY is sampled high).

Transfers cannot be retracted or aborted. In any cycle following a transfer (handshake completion), VALID can either be de-asserted or remain asserted to initiate a new transfer.

Figure 3-14 shows an example of a READY/VALID handshake at the start of a new frame.

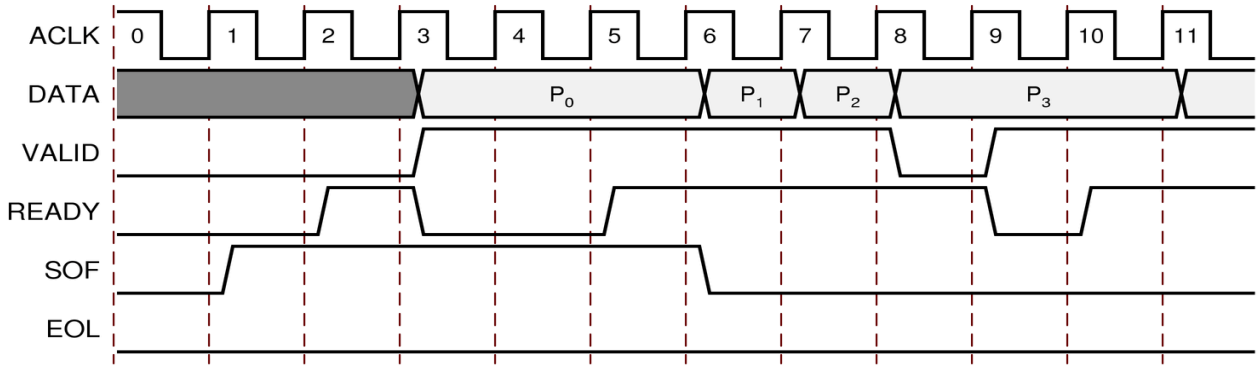


Figure 3-14: **Example of Ready/Valid Handshake at Start of New Frame**

Guidelines on Driving READY

The READY signal can be asserted before, during or after the cycle in which VALID is asserted. The assertion of READY might be dependent upon the value of VALID. The READY slave output cannot be generated combinatorially from the VALID slave input. A slave that can immediately accept data qualified by VALID must pre-assert its READY signal until data is received. Alternatively, READY can be registered and driven in the cycle following the VALID assertion. The default design convention is as follows:

- A slave must drive READY independently.

  or

- Pre-assert READY to minimize latency.

Interfacing to AXI4-Stream With No TREADY Signal

Although READY is a required signal for IP using the AXI4-Stream Video Protocol, the AXI4-Stream allows READY to be omitted.

In the case that the downstream IP is always ready to receive data, the AXI4-Stream slave interface READY signal has a defaults value of 1. However, the upstream IP AXI4-Stream master interface not having a READY could limit interoperability with Video IP that generate READY. It is possible to connect an AXI4-Stream master with only forward flow control (VALID only) to an AXI4-Stream slave with full flow control, such as Video IP (READY and VALID). This generally requires knowledge of the data rates and the use of an AXI4-Stream FIFO block, as shown in Figure 3-13, page 74, to provide elasticity to handle backpressure from READY deassertion.

An example is a Video Input (master) connected to a Video Frame Buffer (slave) that writes to memory. The video camera produces video data that comes in from a unidirectional link such as a DVI cable. The data produced by the camera cannot be back-throttled which is analogous to having VALID handshake only.

The Frame Buffer might have to arbitrate with other devices, such as a processor, for memory access. This could require the memory controller to temporarily become unavailable by deasserting TREADY while waiting for memory access.

After the controller grants access to the Frame Buffer write interface, it asserts READY and takes data. In this example, having an AXI FIFO between the Video Input IP and the Frame Buffer IP would allow the two to connect to each other. If the FIFO depth is selected correctly by analyzing the memory arbitration process, no data is lost to FIFO overflow.

Start of Frame Signal - SOF

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream TUSER0 signal, marks the first pixel of a video frame. The SOF pulse is 1 **valid** transfer wide, and must coincide with the first pixel of the frame shown in Figure 3-14, page 75.

SOF (TUSER0) is defined on a data beat, and is associated technically with the least significant byte of the beat, if between AXI4-Stream infrastructure codes TDATA and TUSER are byte packed, or go through width conversions.

The SOF does the following:

- Serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame
- Can be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on DATA, as long as a VALID is not asserted

Parameterization and/or configuration registers define the dimensions of the video frames that video IP can process. Starting from a known state, based on these configuration settings the IP can predict when the beginning of the next frame is expected.

The SOF that is detected before expected (early), or the SOF that is not present when it is expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration. It is recommended that Video IP flag both error conditions with dedicated flags in the core ERROR register.

Recommended flag names are sk_SOF_EARLY and sk_SOF_LATE, where *k* is the index of the AXI4-Stream slave interface. Also, it is recommended that these flags can trigger interrupts, so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system.

Also, to minimize the impact of sustained error conditions it is also ***recommended***, but not mandated, that:

- When the SOF_EARLY condition is detected, if possible, the IP immediately start processing the new frame. All pixels pertaining to the previous frame should be processed, the last sample of the previous frame should be qualified with the EOL signal, and processing of the new frame should commence.
- When the SOF_LATE condition is detected, the IP should drop (accept on the input, but not propagate to the output) subsequent pixels until the SOF signal arrives.

End Of Line Signal - EOL

The End-Of-Line (EOL) signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is 1 **valid** transfer wide, and must coincide with the last pixel of a scan-line, shown in Figure 3-15.

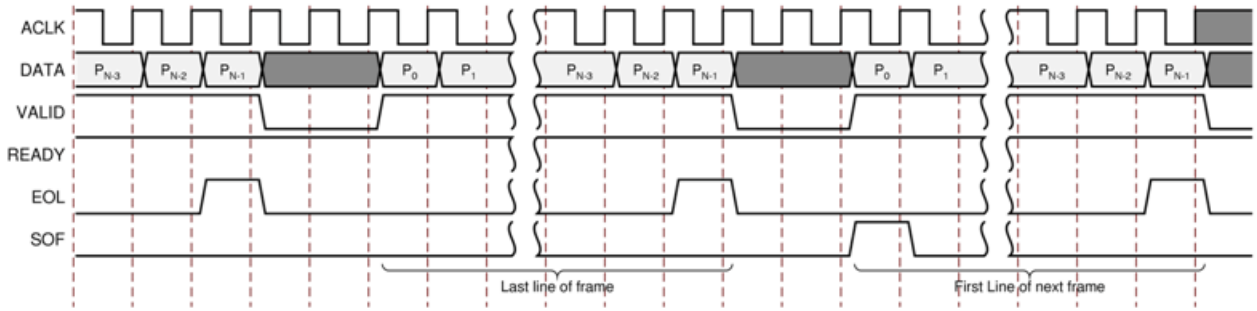


Figure 3-15: **Use of EOL and SOF Signals**

Parameterization and/or configuration registers define the dimensions of video frames video IP should process. Starting from a known state, based on these configuration settings the IP can predict when the last pixel of each scanline is expected. The EOL detected before expected (early), or EOL not present when expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration.

It is recommended that video IP flags both error conditions with dedicated flags in the core ERROR register.

Recommended flag names are sk_EOL_EARLY and sk_EOL_LATE, where *k* is the index of the AXI4-Stream slave interface.

It is recommended that these flags can trigger interrupts, so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system.

Also, to minimize the impact of sustained error conditions it is recommended, but not mandated, that:

- When the EOL_EARLY condition is detected, if possible, the IP should immediately start processing the new line. All pixels pertaining to the previous frame should be flushed out, the line should be qualified with the EOL signal, and processing of the new line should commence.
- When the EOL_LATE condition is detected, the IP must generate its output EOL signal according to the programmed/parameterized line-length, and drop (accept on the input, but not propagate to the output) subsequent pixels until the EOL signal arrives.

Real Time Requirements

The AXI4-Stream interface protocol does not impose any rules on real-time requirements. Video IP should not impose real-time requirements on data transfers, other than to meet and not to break, the fundamental AXI handshake rules at the AXI4-Stream interface. The IP must meet the constraint on the clock signal to which the interface is synchronous.

Data Format

To transport video data, the DATA vector encodes logical channel subsets of physical DATA signals. Various AXI4-Stream interfaces between the modules can facilitate transferring video using different precision (for example; 8, 10, 12, or 16 bits per color channel), and/or different formats (for example RGB or YUV 420).

A specific example of a typical image pre-processing system is illustrated in Figure 3-16, page 79, which consists of a number of Xilinx IP cores connected using AXI4-Stream to implement an imaging sensor processing pipeline.

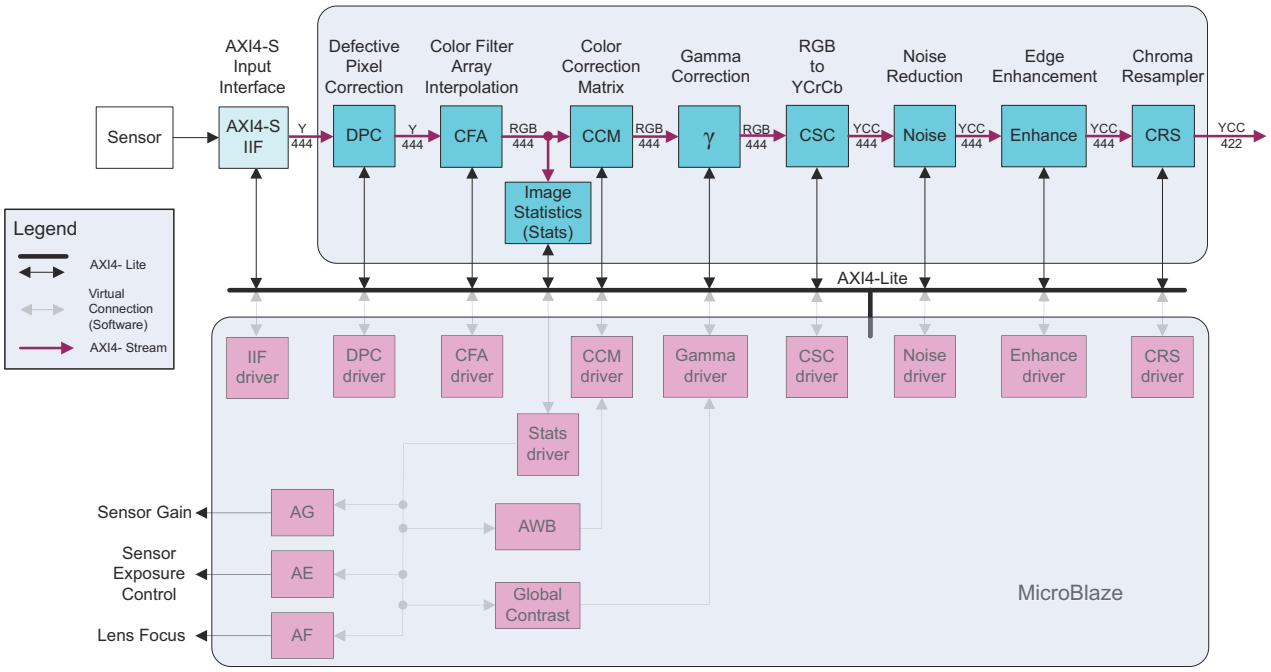


Figure 3-16: **Image Processing Pipeline**

AXI4-Stream channels in Figure 3-16 are annotated to reflect the transferred video format. The `DATA` signal must not have any explicit subfields defined, either as separate ports or with special signal suffixes.

For example, cores with `DATA_Y` and `DATA_C` signals are not permitted. Format information is embedded in the IP-XACT representation of IP, as metadata tags attached to AXI4-Stream ports.

AXI4-Stream Specific Parameterization

Table 3-5 lists the parameters specific to IP using the AXI4-Stream Video Protocol. The paragraphs immediately following the table provide further description.

Table 3-5: **IP using AXI4-Stream Video Protocol Parameters**

Parameter Name	Parameter Function
C_*t*_AXIS_DATA_WIDTH	Width of color/component data.
C_*t*_VIDEO_FORMAT	Video format code (see following description).
C_*t*_AXIS_TDATA_WIDTH	Width of the DATA interface signal.

The C_*t*_AXIS_TDATA_WIDTH parameter determines the width of the variable-width DATA interface signal on AXI4-Stream interface *t*, where interface type *t* can have the values [m, s] designating a master or slave interface

Typically C_*t*_AXIS_TDATA_WIDTH is a function of the component data width and the number of components the actual video format is using.

The recommended parameter names for component data width is C_*t*_DATA_WIDTH. Supported component widths are 8, 10, 12, and 16 bits.

The optional format parameter C_*t*_VIDEO_FORMAT can assist the IP in determining the number of color or components present on DATA using a HDL function. Video IP is typically very specific on the formats expected on the input interfaces, and could already have the number of color or component channels hard coded in the IP. However, when the C_*t*_VIDEO_FORMAT parameter, set by a default value on the master interface, is propagated in HDL designs to slave interfaces, the IP source code can perform DRC by means of assertions to ensure that AXI4-Stream video interfaces are driven by video encoded in the expected format.

Encoding

Table 3-6 lists the detailed representation of video data formats, with DW = C_DATA_WIDTH and VF = C_VIDEO_FORMAT. Video data format codes follow the examples of from the following industry standards:

- *International Telecommunications Union (ITU)*: *ITU-R BT.1614:* http://engineers.ihs.com/document/abstract/SUCFEBAAAAAAAAAA,
- *HDTV Standards and Practices for Digital Broadcasting: SMTPE 352M-2009* (available on the web in video).

Table 3-6: **Data Format Representations**

VF code	Video Format	[4DW-1: 3DW]	[3DW-1: 2DW]	[2DW-1: DW]	[DW-1:0]
0	YUV 4:2:2			V/U, Cr/Cb	Y
1	YUV 4:4:4		V, Cr	U, Cb	Y
2	RGB		R	B	G
3	YUV 4:2:0			V/U, Cr/Cb	Y
4	YUVA 4:2:2		α	V/U, Cr/Cb	Y
5	YUVA 4:4:4	α	V, Cr	U, Cb	Y
6	RGBA	α	R	B	G
7*	YUVA 4:2:0			α V/U, Cr/Cb	Y
8	YUVD 4:2:2		D	V/U, Cr/Cb	Y
9	YUVD 4:4:4	D	V, Cr	U, Cb	Y
10	RGBD	D	R	B	G
11*	YUV 4:2:0		D	V/U, Cr/Cb	Y
12*	Mono				Y Sensor
13*	Reserved			No DRC - 3 user-defined components	
14*	Reserved		No DRC - 3 user-defined components		
15*	Reserved	No DRC - 3 user-defined components			

* Reserved values in SMTPE 352M-2009 Specification Standard

Migrating to Xilinx AXI Protocols

Introduction

Migrating an existing core is a process of mapping your core's existing I/O signals to corresponding AXI protocol signals. In some cases, additional logic might be needed.

The MicroBlaze™ embedded processor changed its *endian orientation* to go from Big-endian orientation, (which aligned with the PLB interfaces of the PowerPC® embedded processors), to Little-endian orientation (which aligns with ARM® processor requirements and the AXI protocol). Migration considerations are covered in Using CORE Generator to Upgrade IP, page 96.

Migrating to AXI for IP Cores

Xilinx provides the following guidance for migrating IP.

Embedded PLBv4.6 IP: There are three embedded IP migration scenarios that need to be considered. Those are IP that:

- Was created from scratch.
- Was created using the Create and Import IP (CIO) wizard in a previous version of Xilinx tools.
- Cannot be altered, and needs to be used as-is with its existing PLBv4.6 interface.

IP created from scratch is not discussed in this section, except for using the AXI IP templates provided in the Xilinx Answer Record37856. Refer to Memory Mapped IP Feature Adoption and Support, page 51 as well as the *ARM AMBA AXI Protocol v2.0 Specification* specification available from the ARM website. New IP should be designed to the AXI protocol.

IP that was created using the Create and Import Peripheral (CIP) Wizard in a previous version of Xilinx tools (before AXI was supported) can be migrated by rerunning the CIP Wizard to create AXI-based template designs. Alternatively, the IP can be migrated using AXI IP templates provided in the Xilinx Answer Record37856. These templates provide example RTL level designs for AXI4, AXI4-Lite, and AXI4-Stream masters and slaves. Check this answer record periodically for updates or new templates.

IP that needs to remain unchanged can be used in the Xilinx tools using the AXI to PLB bridge. See the following section, The AXI To PLBv.46 Bridge, page 83 for more information.

Larger pieces of Xilinx IP (often called *Connectivity* or *Foundation* IP): This class of IP has migration instructions in the respective documentation. This class of IP includes: PCIe, Ethernet, Memory Core, and Serial Rapid I/O.

Local-Link Interface: Local-Link is a generic streaming, FIFO-like interface that has been in service at Xilinx for a number of generations. See Migrating Local-Link to AXI4-Stream, page 86 for more information.

DSP IP: General guidelines on converting this broad class of IP is covered in Migrating HDL Designs to use DSP IP with AXI4-Stream, page 94.

The AXI To PLBv.46 Bridge

For Xilinx embedded processor customers who have developed their own PLBv4.6-based IP, the AXI to PLBv.4.6 Bridge provides a mechanism to incorporate existing IP in AXI-based systems. The AXI4 to Processor Local Bus (PLBv4.6) Bridge translates AXI4 transactions into PLBv4.6 transactions. It functions as 32- or 64-bit slave on AXI4 and a 32- or 64-bit master on the PLBv4.6.

Features

The Xilinx AXI (AXI4 and AXI4-Lite) to PLBv4.6 Bridge is a soft IP core with the following supported features:

- AXI4, AXI4-Lite, and PLB v.46 (with Xilinx simplification)
- 1:1 (AXI:PLB) synchronous clock ratio
- 32-bit address on AXI and PLB interface
- 32- or 64-bit data buses on AXI and PLB interfaces (1:1 ratio)
- Write and read data buffering

AXI4 Slave Interface

The following are the supported AXI4 Slave Interface (SI) features:

- Configurable AXI4 interface categories
- Control (AXI4-Lite) interface
- Read/write interface
- Read-only interface

- Write-only interface
- Additional control interface to access internal registers of the design
- INCR bursts of 1 to 256
- Bursts of 1-16 for FIXED-type transfers
- Bursts of 2, 4, 8 and 16 for WRAP-type transfers
- Configurable support for narrow transfers
- Unaligned transactions
- Early response for bufferable write transfer
- Debug register for error/timeout condition for bufferable write transfer
- Configurable (maximum of two) number of pipelined read/write addresses
- Interrupt generation for write null data strobes
- Interrupt generation for partial data strobes except first and last data beat
- Supports simultaneous read and write operations

PLBv4.6 Master Interface

The following are the supported PLBv4.6 Master Interface (MI) features:

- Configurable (maximum of two) number of pipelined read/write address
- Xilinx-simplified PLBv4.6 protocol
- Single transfers of 1 to 4 through 8 bytes
- Fixed length of 2 to 16 data beats
- Cacheline transactions of line size 4 and 8
- Address pipelining for one read and one write
- Supports simultaneous read and write operations
- Supports 32-, 64-, and 128-bit PLBv4.6 data bus widths with required data mirroring

AXI to PLBv4.6 Bridge Functional Description

Figure 4-1 shows a block diagram of the AXI PLBv4.6 bridge.

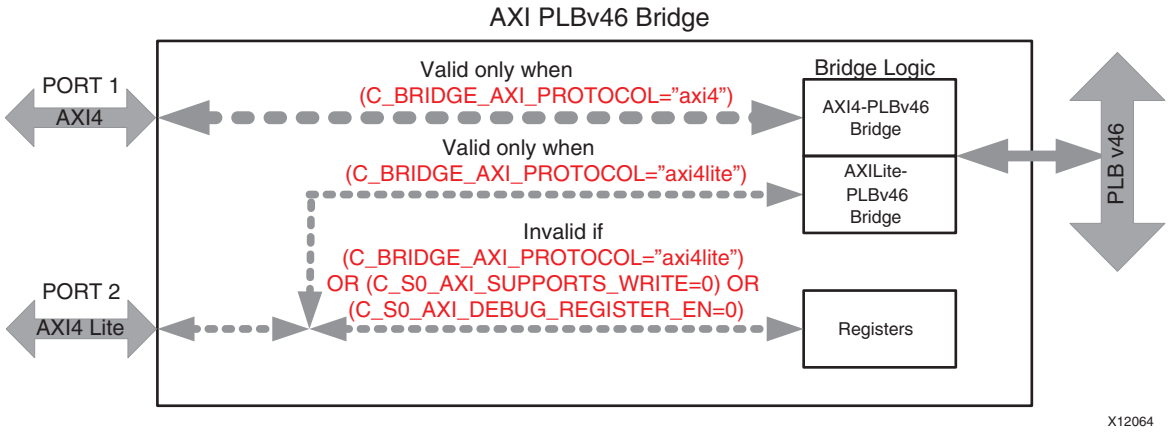


Figure 4-1: **AXI to PLBv4.6 Block Diagram**

- The `PORT-2` is valid when `C_EN_DEBUG_REG=1`, `C_S_AXI_PROTOCOL="AXI4"`, and `C_S_AXI_SUPPORTS_WRITE=1` only.
- The AXI data bus width is 32- and 64-bit and the PLBv4.6 master is a 32- and 64-bit device (for example, `C_MPLB_NATIVE_DWIDTH= 32/64`).
  PLBv4.6 data bus widths of 32-bit, 64-bit, and 128-bit are supported with the AXI to PLBv4.6 Bridge performing the required data mirroring.
- AXI transactions are received on the AXI Slave Interface (SI), then translated to PLBv4.6 transactions on the PLBv4.6 bus master interface.
- Both read data and write data are buffered (when `C_S_AXI_PROTOCOL="AXI4"`) in the bridge, because of the mismatch of AXI and PLBv4.6 protocols where AXI protocol allows the master to throttle data flow, but PLBv4.6 protocol does not allow PLB masters to throttle data flow.

The bridge:

- Buffers the write data input from the AXI port before initiating the PLBv4.6 write transaction.
- Implements a read and write data buffer of depth 32x32/64x32 to hold the data for two PLB transfers of highest (16) burst length.
- Supports simultaneous read and write operations from AXI to PLB.

Migrating Local-Link to AXI4-Stream

Local-Link is an active-Low signaling standard, which is used commonly for Xilinx IP cores. Many of these signals have a corollary within AXI4-Stream, which eases the conversion to AXI4-Stream protocol.

Required Local-Link Signal to AXI4-Stream Signal Mapping

The Local-Link has a set of required signals as well as a set of optional signals. Table 4-1 shows the list of required signals the mapping to the AXI4-Stream protocol signals.

Table 4-1: **Required Local-Link Signals**

Signal Name	Direction	Description	Mapping to AXI4-Stream Signals
CLK	Input	Clock: all signals are synchronous to this clock	ACLK
RST_*N*	Input	Reset: When reset is asserted, control signals deassert	ARESETN (or some other reset)
DATA	src to dst	Data Bus: Frame data is transmitted across this bus	TDATA
SRC_RDY_*N*	src to dst	Source Ready: Indicates that the source is ready to send data	TVAILD
DST_RDY_*N*	dst to src	Destination Ready: Indicates that the sink is ready to accept data	TREADY
SOF_*N*	src to dst	Start-of-Frame: Indicates the first beat of a frame	Optional TUSER
EOF_*N*	src to dst	End-of-Frame: Indicates the last beat of a frame	TLAST

- You can map clock and reset signals directly to the appropriate clock and reset for the given interface. The ARESETN signal is not always present in IP cores, but you can instead use another system reset signal.
- In AXI4-Stream, the TDATA signal is optional. Because DATA is required for a Local-Link, it is assumed that an interface converted to AXI4-Stream from Local-Link must create the TDATA signal.
- The Source and Destination Ready are active-Low signals. You can translate these signals to TVALID and TREADY by inverting them to an active-High signal.

  Note: The TREADY signal is optional. It is assumed that when you convert an interface, you chose to use this signal.

- The EOF_N is an active-Low signal used to indicate the end of a frame. With an inversion, this will connect directly to TLAST, which is an optional signal.
- The SOF_N signal does not have a direct map to AXI4-Stream. The start of frame is implied, because it is always the first valid beat after the observation of TLAST (or reset).
  In most cases the signal is no longer needed for the interface. If a start of frame signal is needed, it could be applied to the TUSER field.

Figure 4-2 shows a typical single Local-Link waveform.

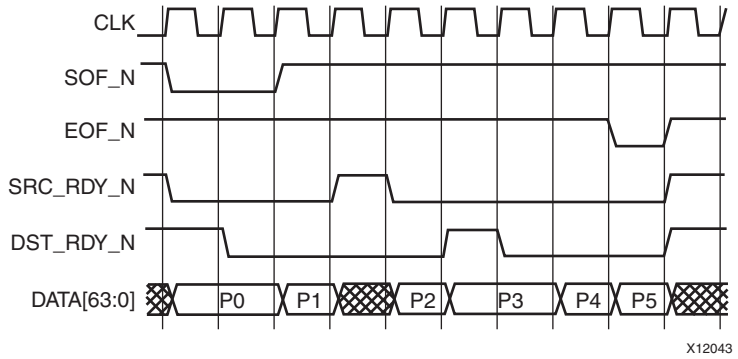


Figure 4-2: **Single Local-Link Waveform**

The preceding figure shows how the flow control signals (SRC_RDY_N and DST_RDY_N) restrict data flow. Also, observe how SOF_N and EOF_N signals frame the data packet.

Figure 4-3 shows the same type of transaction with AXI4-Stream. Note the only major difference is the absence of an SOF signal, which is now implied.

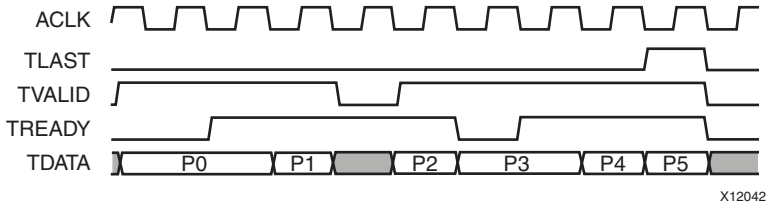


Figure 4-3: **AXI4-Stream Waveform**

Optional Local-Link Signal to AXI4-Stream Signal Mapping

Table 4-2 shows a list of the more common optional signals within Local-Link.

Table 4-2: **Optional Local-Link Signals Mapped to AXI4-Stream Signals**

Signal Name	Direction	Description	Mapping to AXI
SOP_*N*	src to dst	Start-of-Packet: Packetization within a frame.	TUSER
EOP_*N*	src to dst	End-of-Packet: Packetization within a frame.	TUSER
REM	src to dst	Remainder Bus: Delineator between valid and invalid data.	TKEEP
SRC_DSC_*N*	src to dst	Source Discontinue: Indicates the source device is cancelling a frame.	TUSER
DST_DSC_*N*	dst to src	Destination Discontinue: Indicates the destination device is cancelling a frame.	<none>
CH	src to dst	Channel Bus Number: Used to specify a channel or destination ID.	TID
PARITY	src to dst	Parity: Contains the parity that is calculation over the entire data bus.	TUSER

- Any optional signal that is not represented in this table must be sent using the TUSER signal.
- The SOP_*N* and EOP_*N* signals are rarely used in Local-Link. They add granularity to the SOF/EOF signals. If there is a need for them, they must be created in the TUSER field.
- The REM signal specifies the remainder of a packet. AXI4-Stream has TKEEP bus that might have deasserted bits when TLAST = 1 to signal the location of the last byte in the packet.
- Source discontinue, SRC_DSC_*N*, is a mechanism to end a transaction early by the source. This can be done through the TUSER field. The source device must apply the TLAST signal also.
- There is no native mechanism within the AXI4-Stream protocol for a destination discontinue. TUSER cannot be used because it always comes from the source device. You can use a dedicated sideband signal that is not covered by AXI4-Stream. Such a signal would not be considered within the bounds of the AXI4-Stream.
- The CH indicator can be mapped to the thread ID (TID). For parity, or any error checking, the TUSER is a suitable resource.

Variations in Local-Link IP

There are some variations of Local-Link to be aware of:

- Some variations use active-High signaling. Because AXI4-Stream uses active-High signaling (except on ARESETN), the resulting polarity should still be the same.
- Some users create their own signals for Local-Link. These signals are not defined in the Local-Link specification.
  - In cases where the signal goes from the source to the destination, a suitable location is TUSER.
  - If the signals go from the destination to the source, they cannot use TUSER or any other AXI4-Stream signal. Instead, one of the preferable methods is to create a second return AXI4-Stream interface. In this case, most of the optional AXI4-Stream signals would not be used; only the TVALID and TUSER signals.

Local-Link References

The Local-Link documentation is on the following website:

http://www.xilinx.com/products/design_resources/conn_central/locallink_member/sp06.pdf.

Using System Generator for Migrating IP

You can migrate both DSP and PLBv4.6 IP using the System Generator tool.

Migrating a System Generator for DSP IP to AXI

When migrating a System Generator for DSP design to use AXI IP, there are some general items to consider. This subsection elaborates on those key items, and is intended to be an overview of the process; IP-specific migration details are available in each respective data sheet.

Resets

In System Generator, the resets on non-AXI IP are active-High. AXI IP in general, and in System Generator, has an active-Low reset, aresetn. System Generator "Inverter" blocks are necessary when using a single reset signal to reset both AXI and non-AXI IP.

A minimum aresetn active-Low pulse of two cycles is required, because the signal is internally registered for performance. Additionally, aresetn always takes priority over aclken.

Clock Enables

In System Generator, the clock enables on both non-AXI and AXI IP are active-High. AXI IP in System Generator use an active-High clock-enable, `aclken`.

TDATA

In AXI protocols, data is consolidated onto a single `TDATA` input stream. This is consistent with the top-level ports on DSP IP in CORE Generator. For ease of connecting data ports, System Generator breaks `TDATA` down into individual ports in the block view.

An example of this is the AXI Complex Multiplier block as shown in Figure 4-4:

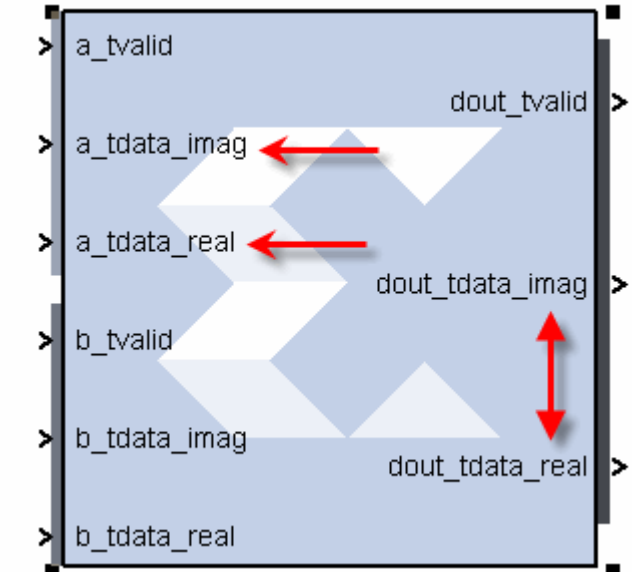


Figure 4-4: **AXI Complex Multiplier Block**

Port Ordering

When comparing non-AXI and AXI IP, such as the Complex Multiplier 3.1 and 4.0, respectively, the real and imaginary ports appear in opposite order when looking at the block from top to bottom. You must be careful to not connect the AXI block with the data paths accidentally crossed.

Figure 4-5 shows an example of the port signals.

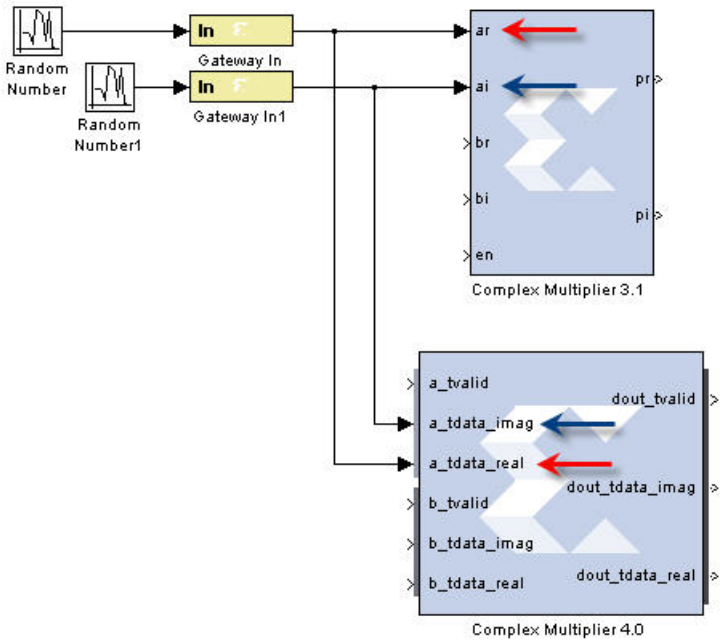


Figure 4-5: **Port Signal Example**

Latency

With AXI IP in System Generator, the latency is handled in a different fashion than non-AXI IP. In non-AXI IP, you can specify latency directly using either a GUI option or by specifying **-1** in the **maximum performance** option. With AXI IP, the latency is either *Automatic* or *Manual*:

- **Automatic** replaces the -1 in the **maximum performance** option.
- To manually set the latency, the parameter is called **Minimum Latency** for AXI blocks because:
  - In *blocking* mode, the latency can be higher than the minimum latency specified if the system has back pressure.
  - In a *non-blocking* AXI configuration, the latency is deterministic.

With DSP IP that support the AXI4-Stream interface, each individual AXI4-Stream slave channel can be categorized as either a blocking or a non-blocking channel. A slave channel is blocking when some operation of the core is inhibited until a transaction occurs on that channel. In general, the latency of the DSP IP AXI4-Stream interface is static for non-blocking and variable for blocking mode. To reduce errors while migrating your design, pay attention to the "Latency Changes" and "Instructions for Minimum Change Migration" sections of the IP data sheet.

Output Width Specification

Specification for the output word width is different for non-AXI and AXI IP in System Generator. In some non-AXI IP, you must specify both the Output Most Significant Bit (MSB) and Output Least Significant Bit (LSB). With AXI IP, you need to specify only the Output Width (total width); the binary point location is determined automatically. Output rounding behavior is consistent with non-AXI and AXI versions of the IP.

Migrating PLBv4.6 Interfaces in System Generator

System Generator provides a straight-forward migration path for designs that already contain PLBv4.6 interfaces. Select the AXI4 interface in the EDK Processor block before generating a processor core, as shown in Figure 2-1, page 14.

When you use the processor import mode, the **Bus Type** field is grayed out because System Generator auto-detects the bus used by the imported MicroBlaze™ processor and uses the appropriate interface. In this case, you must create a new XPS project with a MicroBlaze processor that has an AXI4 interface before importing into System Generator.

Migrating a Fast Simplex Link to AXI4-Stream

When converting a Fast Simplex Link (FSL) peripheral to an AXI4-Stream peripheral there are several considerations. You must migrate the:

- Slave FSL port to an AXI4-Stream slave interface
- Master FSL port to an AXI4-Stream master interface

The following tables list the master-FSL and slave-FSL to AXI4-Stream signals conversion mappings.

Master FSL to AXI4-Stream Signal Mapping

Table 4-3 shows the AXI4-Stream sgnal mapping.

Table 4-3: **AXI4-Stream Signal Mapping**

Signal	Direction	AXI Signal	Direction
FSL_M_Clk	Out	M_AXIS_<*Port_Name*>ACLK	In
FSL_M_Write	Out	M_AXIS_<*Port_Name*>TVALID	Out
FSL_M_Full	In	M_AXIS_<*Port_Name*>TREADY	In
FSL_M_Data	Out	M_AXIS_<*Port_Name*>TDATA	Out
FSL_M_Control	Out	M_AXIS_<*Port_Name*>TLAST	Out

Slave FSL to AXI4-Stream Signal Mapping

Table 4-4 shows the FSO to AXI4-Stream signal mapping.

Table 4-4: **FSO to AXI-4 Stream Signal Mapping**

Signal	Direction	AXI Signal	Direction
FSL_S_Clk	Out	S_AXIS_<*Port_Name*>ACLK	In
FSL_S_Exists	In	S_AXIS_<*Port_Name*>TVALID	In
FSL_S_Read	Out	S_AXIS_<*Port_Name*>TREADY	Out
FSL_S_Data	In	S_AXIS_<*Port_Name*>TDATA	In
FSL_S_Control	In	S_AXIS_<*Port_Name*>TLAST	In

Differences in Throttling

There are fundamental differences in throttling between FSL and AXI4-Stream, as follows:

- The `AXI_M_TVALID` signal cannot be deasserted after being asserted unless a transfer is completed with `AXI_TREADY`. However, a `AXI_TREADY` can be asserted and deasserted whenever the AXI4-Stream slave requires assertion and deassertion.
- For FSL, the signals `FSL_Full` and `FSL_Exists` are the status of the interface; for example, if the slave is full or if the master has valid data
- An FSL-master can have a pre-determined expectation prior to writing to FSL to check if the FSL-Slave can accept the transfer based on the FSL slave having a current state of `FSL_Full`
- An AXI4-Stream master cannot use the status of `AXI_S_TREADY` unless a transfer is started.

The MicroBlaze processor has an FSL test instruction that checks the current status of the FSL interface. For this instruction to function on the AXI4-Stream, MicroBlaze has an additional 32-bit Data Flip-Flop (DFF) for each AXI4-Stream master interface to act as an output holding register.

When MicroBlaze executes a `put fsl` instruction, it writes to this DFF. The AXI4-Stream logic inside MicroBlaze moves the value out from the DFF to the external AXI4-Stream slave device as soon as the AXI4-Stream allows. Instead of checking the AXI4-Stream `TREADY/TVALID` signals, the FSL test instruction checks if the DFF contains valid data instead because the `AXI_S_TREADY` signal cannot be directly used for this purpose.

The additional 32-bit DFFs ensure that all current FSL instructions to work seamlessly on AXI4-Stream. There is no change needed in the software when converting from FSL to AXI4 stream.

For backward compatibility, the MicroBlaze processor supports keeping the FSL interfaces while the normal memory mapped AXI interfaces are configured for AXI4.

This is accomplished by having a separate, independent MicroBlaze configuration parameter (C_STREAM_INTERCONNECT) to determine if the stream interface should be AXI4-Stream or FSL.

Migrating HDL Designs to use DSP IP with AXI4-Stream

Adopting an AXI4-stream interface on a DSP IP should not change the functional, or signal processing behavior of the DSP function such as a filter or a FFT transform. However, the sequence in which data is presented to a DSP IP could significantly change the functional output from that DSP IP. For example, one sample shift in a time division multiplexed input data stream will provide incorrect results for all output time division multiplexed data.

To facilitate the migration of an HDL design to use DSP IP with an AXI4-Stream interface, the following subsections provide the general items to consider:

- DSP IP-Specific Migration Instructions
- Demonstration Testbench
- Using CORE Generator to Upgrade IP
- Latency Changes
- Mapping Previously Assigned Ports to An AXI4-Stream Video Protocol

DSP IP-Specific Migration Instructions

This section provides an overview with IP specific migration details available in each respective data sheet. Before starting the migration of a specific piece of IP, review the "AXI4-Stream Considerations" and "Migrating from earlier versions" in the individual IP data sheets.

Figure 4-6 shows an example.

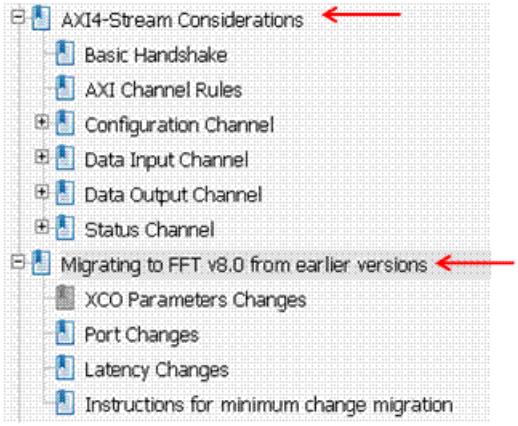


Figure 4-6: **Example IP Data Sheet**

Demonstration Testbench

To assist with core migration, CORE Generator generates an example testbench in the `demo_tb` directory under the CORE Generator project directory. The testbench instantiates the generated core and demonstrates a simple example of how the DSP IP works with the AXI4-stream interface. This is a simple VHDL testbench that exercises the core.

The demonstration testbench source code is one VHDL file, `demo_tb/tb_<component_name>.vhd`, in the CORE Generator output directory.

The source code is comprehensively commented. The demonstration testbench drives the input signals of the core to demonstrate the features and modes of operation of the core with the AXI4-Stream interface. For more information on how to use the generated testbench refer to the "Demonstration Testbench" section in the individual IP data sheet.

Figure 4-7 shows the demo_tb directory structure.

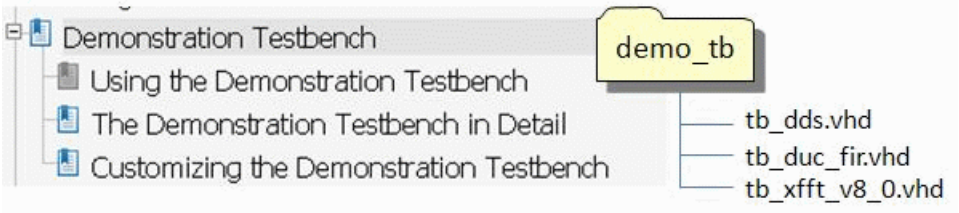


Figure 4-7: **demo_tb Directory Structure**

Using CORE Generator to Upgrade IP

When it is available, you can use the CORE Generator core upgrade functionality to upgrade an existing XCO file from previous versions of the core to the latest version. DSP IP cores such as FIR Compiler v6.0, Fast Fourier Transform v8.0, DDS Compiler v5.0, and Complex Multiplier v4.0 provide the capability to upgrade XCO parameters from previous versions of the core. Figure 4-8 shows the upgrade function in CORE Generator.

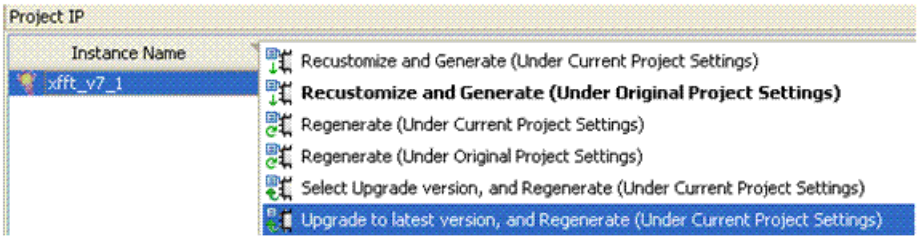


Figure 4-8: **CORE Generator Upgrade Function**

Note: The upgrade mechanism alone will not create a core compatible with the latest version but will provide a core that has equivalent parameter selection as the previous version of the core. The core instantiation in the design must be updated to use the AXI4-Stream interface. The upgrade mechanism also creates a backup of the old XCO file. The generated output is contained in the `/tmp` folder of the CORE Generator project.

Latency Changes

With DSP IP that support the AXI4-Stream interface, each individual AXI4-Stream slave channel can be categorized as either a *blocking* or a *non-blocking* channel. A slave channel is blocking when some operation of the core is inhibited until a transaction occurs on that channel. In general, the latency of the DSP IP AXI4-Stream interface is static for non-blocking and variable for blocking mode. To reduce errors while migrating your design, pay attention to the "Latency Changes" and "Instructions for Minimum Change Migration" sections of the IP data sheet.

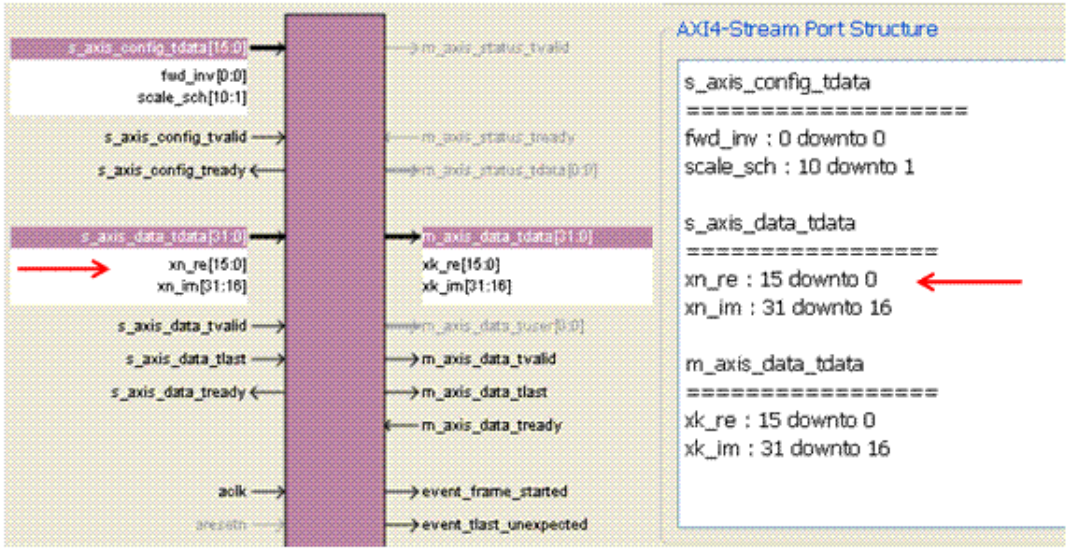
Mapping Previously Assigned Ports to An AXI4-Stream Video Protocol

The individual DSP IP datasheet provides a table about the changes to port naming, additional or deprecated ports, and polarity changes from previous version to the latest version of the core. Noteworthy changes are:

- **Resets:** The DSP IP AXI4-stream interface `aresetn` reset signal is active-Low and must be asserted for a minimum length of two clock cycles. The `aresetn` reset signal always takes priority over the `aclken` clock enable signal.
  Therefore your IP instantiation reset input must change from an active-High "SCLR" signal with a minimum length of one clock cycle, to a active-Low reset input with a minimum of two clock cycles.

- **Input and Output TDATA port structure**: The AXI specification calls for data to be consolidated onto a single TDATA input stream. For ease of visualization you can view the TDATA structure from the IP symbol and implementation details tab in the IP GUI. For ease of IP instantiation the demonstration example testbench also shows how to connect and functionally split signals from the TDATA structure. The demonstration testbench assigns TDATA fields to aliases for easy waveform viewing during simulation.

  Figure 4-9 shows the TDATA port structure.




```
-------------------------------------------------------------------------
-- Assign TDATA / TUSER fields to aliases, for easy simulator waveform viewing
-------------------------------------------------------------------------

-- Config slave channel alias signals
s_axis_config_tdata_fwd_inv    <= s_axis_config_tdata(0);
s_axis_config_tdata_scale_sch  <= s_axis_config_tdata(10 downto 1);

-- Data slave channel alias signals
s_axis_data_tdata_re           <= s_axis_data_tdata(15 downto 0);
s_axis_data_tdata_im           <= s_axis_data_tdata(31 downto 16);

-- Data master channel alias signals
m_axis_data_tdata_re           <= m_axis_data_tdata(15 downto 0);
m_axis_data_tdata_im           <= m_axis_data_tdata(31 downto 16);
```

Figure 4-9: **TDATA Port Structure**

Migrating Designs from XSVI to the AXI4-Stream Video Protocol

Bridging Xilinx Streaming Video Interface with AXI4-Stream Protocol (XAPP521) [Ref 17] describes how to migrate from Xilinx Streaming Video Interface (XSVI) to AXI4-Stream.

It is also recommended that you review Video IP: AXI Feature Adoption, page 67. This describes how the Xilinx Video IP manage Video data using AXI4-Stream, and is useful when migrating designs to use the AXI4-Stream Video Protocol.

Tool Considerations for AXI Migration (Endian Swap)

When a MicroBlaze™ PLBv4.6-based Big-Endian system is converted to an AXI-based Little-Endian system, you need to consider the implications on software flows. This section describes general considerations while developing software for such a design.

The tool flows through the Xilinx Software Development Kit (SDK) remain the same. The IDE and underlying tools detect the MicroBlaze processor interface automatically and infer the correct options to use.

For example, the compiler flag `-mlittle-endian` is added automatically when you build board support packages and user applications in SDK-managed builds, but you must add this option explicitly when directly calling the MicroBlaze GNU compiler from the command line.

Libgen, XMD, and other command line tools generate output or interact with the processor system while taking into account its endianness. In addition, drivers are properly associated with AXI embedded processing IP used in the design, including an updated CPU driver for MicroBlaze v8.00.a.

However, the Xilinx Platform Studio IDE does not support the creation of board support packages or user applications for AXI-based designs. Use SDK instead or invoke tools directly from the command line.

End-user applications written for a MicroBlaze Big-Endian system can work with the equivalent little-endian system with minimal or no changes if the code is not sensitive to endian orientation.

After the hardware migration is completed, migrate applications using the same approach used when updating hardware designs previously (see the SDK online help for suggested steps). Function calls in user applications to board support package functions for OS libraries and drivers might need to be modified if the API has changed.

TIP: *You might need to rewrite code or custom drivers, if the code is sensitive to the representation of data in Big-Endian or Little-Endian formats.*

General Guidelines for Migrating Big-to-Little Endian

The following guidelines summarize general considerations when migrating software flows and code used in a MicroBlaze Big-Endian representation to a Little-Endian design.

1. Be aware of situations where software is sensitive to endian orientation, especially when Reading or Writing data from, or to, memory, files, and streams.
2. Existing software that ran on a MicroBlaze PLBv4.6 big-endian system might need to change:
   a. XPS tool flows do not support AXI; SDK must be used.
   b. In SDK, import the new AXI hardware handoff, create new board packages, and import the software applications.
   c. Modify driver calls as needed and ensure user code is endian-safe. User-developed drivers might need to be re-written if their behavior is affected by endian orientation.
   d. If running the GNU tools on the command line and writing make files, the correct compiler flag `-mlittle-endian` must be used.
3. MicroBlaze Little-Endian and Big-Endian software are not compatible:
   a. *Do not* mix object files (`.o`) and libraries created with different endian data representations.
   b. *Do not* mix drivers unless they are known to be compatible.
   c. *Do not* use ELF files built for Big-Endian systems on a Little-Endian system (and vice versa).
   d. *Do not* use generated Xilinx data files that are affected by endian orientation (for example BIT files that include block RAM data - like ELF files) across systems.
   e. Block RAM initialization and data sharing should reflect the endian orientation requirements of the master.
   f. *Do not* use old application data files used by the application if it is affected by the endian orientation (byte ordering in the file).
   g. Be aware of the MicroBlaze endian orientation, and write code appropriately.

h. When exchanging data between big-endian and little-endian masters, user application code and/or drivers need to manipulate the ordering of data to ensure interoperability.

The following tables summarize the organization of data when MicroBlaze uses the Big-Endian or Little-Endian format to represent various data types. This information was reproduced from the *MicroBlaze Processor Reference Guide (UG081)* [Ref 21].

Data Types and Endian Orientation

MicroBlaze uses Big-Endian or Little-Endian format to represent data, depending on the parameter C_ENDIANNESS. The hardware-supported data types for the MicroBlaze processor are word, half word, and byte. When using the reversed load and store instructions LHUR, LWR, SHR, and SWR, the bytes in the data are reversed, as indicated by the byte-reversed order.

The bit and byte organization for each type is shown in Table 4-5 through Table 4-7, page 101.

Table 4-5: **BIt and Byte Organization by Word Data Type**

Word Data Type	Value			
Big-Endian Byte Address	n	n+1	n+2	n+3
Big-Endian Byte Significance	MSByte			LSByte
Big-Endian Byte Order	n	n+1	n+2	n+3
Big-Endian Byte-Reversed Order	n+3	n+2	n+1	n
Little-Endian Byte Address	n+3	n+2	n+1	n
Little-Endian Byte Significance	MSByte			LSByte
Little-Endian Byte Order	n+3	n+2	n+1	n
Little-Endian Byte-Reversed Order	n	n+1	n+2	n+3
Bit Label	0			31
Bit Significance	MSBit			LSBit

Table 4-6: **Half Word Data Types**

Half Word Data Type	Value	
Big-Endian Byte Address	n	n+1
Big-Endian Byte Significance	MSByte	LSByte
Big-Endian Byte Order	n	n+1
Big-Endian Byte-Reversed Order	n+1	n

Table 4-6: **Half Word Data Types *(Cont'd)***

Half Word Data Type	Value	
Little-Endian Byte Address	n+1	n
Little-Endian Byte Significance	MSByte	LSByte
Little-Endian Byte Order	n+1	n
Little-Endian Byte-Reversed Order	n	n+1
Bit Label	0	15
Bit Significance	MSBit	LSBit

Table 4-7: **Byte Data Types**

Byte Address	n	
Bit Label	0	7
Bit Significance	MSBit	LSBit

High End Verification Solutions

Many third-party companies (such as Cadence Design Systems, ARM, Mentor Graphics, and Synopsys) have tools whose function is to allow system-level verification and performance tuning for system-level design. When designing large AXI-based systems, if the highest possible verification and performance are required, it is recommended that third-party tools be used.

AXI System Optimization: Tips and Hints

Introduction

AXI-based Xilinx® IP, third party IP, and user IP present a wide range of configuration options and design choices that let you tune a system for size, Fmax, throughput, latency, ease of use, and ease of debug. IP design decisions and system architecture also impact the area and performance of the system. Given that AXI-based systems must span a wide solution space from small Spartan® and Artix® class designs to very large high performance Virtex® designs, there is a large configuration space for AXI IP and systems.

This chapter provides information and presents concepts that help you optimize your IP designs and system configurations. In some cases, optimization for one attribute might conflict with another requiring you to balance competing tradeoffs. For example, improving timing through the use of additional pipelining negatively impacts area.

Table 5-1, page 103 and Table 5-2, page 105 illustrate the impact of different AXI Interconnect and IP features, configuration parameters, and optimization options across various criteria. The impact on each criterion is qualitatively described using a positive to negative scale of Best (++), Better (+), Neutral (0), Worse (-), and Worst (--). When architecting, optimizing, or diagnosing systems, use these tables to help with designing the IP or system to maximize the attributes required by the application while minimizing negative trade-offs.

You need to be familiar with the AXI protocol, the *AXI Interconnect IP (DS768)* [Ref 6] datasheet and general XPS tool usage to better understand the optimization options and strategies described in this chapter.

Table 5-1: **AXI Interconnect Optimization/Feature Impact**

Feature	Configuration	Size/Area	Timing/Fmax	Throughput	Latency	Ease of Use/Flexibility	Ease of Debug	Notes
Clock Domain Conversion (Optional, Default = OFF	Synch Clock Conversion	-	+	0	-	+	0	Synch clock conversions require complex multi-cycle timespecs (core level UCF is automatically generated by AXI Interconnect TCL).
	Async Clock Conversion	--	+	0	--	++	0	Asynchronous clock conversion includes a 32-bit deep FIFO. Synchronized conversion is generally preferred over asynchronous conversion.
Width Conversion (Optional, Default = OFF)	Downsizer	--	--	--	-	+	0	Size converters do not support multi-threading. They will stall when IDs change until transactions on previous IDs are complete.
	Upsizer	--	-	-	-	+	0	
Endpoint Slave Protocol Conversion from AXI4, (Optional, Default = OFF)	AXI4-Lite	0	0	--	0	+	++	AXI3 converter does not support multi-threading. It will stall when IDs change until transactions on previous IDs are completed. AXI3 converter also requires logic to handle splitting of long AXI4 bursts to AXI3 bursts of maximum length of 16. This logic adds size and latency.
	AXI3	-	-	-	-	0	0	
Connectivity Mode	Shared Access Shared Data	++	0	--	0	+	++	
	Crossbar - Sparse (Default)	0	+	+	0	0	0	
	Crossbar - Fully Connected	--	-	+	0	0	0	
ID Threading	Single Thread	+	+	0	0	+	+	Applies when a master declares that it does not use IDs or interconnect is explicitly placed into Single Thread mode. Note that configuration setting such as Shared Address Shared Data mode (SASD), size conversion, or protocol conversion, might automatically cause Single Thread mode to be used.
	Multiple Thread Support	0	0	+	0	0	-	As more threads are used and there is a greater potential for read reordering or read interleaving, the more difficult it is to debug.
Issuance/Acceptance	1 (Default)	+	+	-	0	+	+	
	2, 4, 8, 16, 32	-	0	+	-	0	-	
Data Path Width	32 (Default)	+	+	-	0	0	0	
	64, 128, 256, 512, 1024	--	-	++	0	0	0	

Legend: "++" = Best; "+" = Better, "0" = Neutral, "-" = Worse, "--" = Worst

Table 5-1: **AXI Interconnect Optimization/Feature Impact *(Cont'd)***

Feature	Configuration	Size/Area	Timing/Fmax	Throughput	Latency	Ease of Use/Flexibility	Ease of Debug	Notes
Register Slice (Optional, Default = OFF)	Type 7 (Light Weight)	-	++	-	-	+	0	AXI Interconnect has added a "Type 8" register slice that automatically selects the register slice type based upon interconnect configuration. Type 8 is recommended and should be overridden only when warranted.
	Type 1 (Fully Registered)	--	++	0	-	+	0	
	Type 8 (Automatic)	-	++	0	-	+	0	
Floorplanning (Optional, Default = OFF)	Floorplan IP Blocks And/Or Submodules	0	+	0	0	--	0	
Data Path FIFOs (Optional, Default = OFF)	SRL	-	0	+	-	0	0	SRL FIFO is 32 deep.
	BRAM	--	0	++	-	0	0	BRAM FIFO is 512 deep. Use of additional BRAM FIFO option, to delay `AWVALID/ARVALID` until FIFO occupancy permits interrupted burst transfers, can further improve throughput at the expense of increased latency.
Arbitration Priority (Optional, Default = Round Robin)	Fixed Priority Over Round Robin	0	0	0	+	-	0	Each master can be assigned a higher fixed priority that supersedes masters at the default priority level of 0. Masters set to the default priority of 0 share Round Robin priority.
AXI ChipScope Monitor (Optional, Default = OFF)	ON	-	-	0	0	0	++	
AXI Hardware Protocol Checker (Optional, Default = OFF)	ON	-	-	0	0	0	++	

Legend: "++" = Best; "+" = Better, "0" = Neutral, "-" = Worse, "--" = Worst

Table 5-2: **AXI Endpoint IP Optimization/Feature Impact**

Feature	Configuration	Size/Area	Timing/Fmax	Throughput	Latency	Ease of Use/Flexibility	Ease of Debug	Notes
IP Protocol	AXI4-Lite	++	0	-	0	+	++	AXI Interconnect i s natively AXI4-based. Use of AXI3 protocol not recommended for new designs.
	AXI4	0	0	0	0	0	0	
Narrow Burst Support	ON (Default)	--	-	-	-	0	0	Narrow burst is assumed to be ON unless the AXI master IP specifically designates this as OFF. Xilinx AXI Master IP generally do not use narrow burst and designate themselves as OFF.
	OFF (Recommended)	0	0	0	0	0	0	
Threading	Uses No Thread or Issues Only a Single Thread	+	+	0	0	+	+	Using a single thread while intermixing transactions destined to multiple different AXI slave endpoints could trigger stalling by the deadlock avoidance logic in the AXI Interconnect.
	Issues Multiple Threads	0	0	+	0	0	-	
Ability to Pipeline Transactions	1	+	+	-	0	+	+	New transactions pipelined behind high numbers of pipelined transactions might experience high latency, but throughput might be improved. Head of line blocking can be caused by excessively pipelined transactions.
	> 1 up to 32	-	0	+	0	0	0	
Data Path Width	32	+	+	-	0	0	0	Native Dat Path Width should be minimized while meeting performance requirements of application. The caveat is that support for a wider native width that minimizes size conversion in a system could be more beneficial.
	64, 128, 256, 512, 1024	--	-	++	0	0	0	
AXI4 Burst Length	Short (1-4)	0	0	-	+	0	0	Head of line blocking can be caused by long bursts. Transactions pipelined behind long bursts might experience high latency, but throughput might be improved.
	Long (up to 256)	0	0	++	--	0	0	

Legend: "++" = Best; "+" = Better, "0" = Neutral, "-" = Worse, "--" = Worst

AXI System Optimization

In general, system optimization follows the guidelines in the following subsections.

Size/Area Optimization Guidelines

When considering your AXI IP or system design, use the following size and area guidelines:

1. Minimize the clock domain conversions by reducing the logic associated with clock domain conversion. Use as few clocks as possible and, if clock conversion is necessary, attempt to keep the clocks to synchronous integer ratios.
2. Use Sparse Crossbar connectivity or Shared Address Shared Data (SASD) configurations of the AXI Interconnect. Analyze the connectivity and bandwidth requirements of the system.

   When possible, specify the minimum required connectivity map for the system when AXI Interconnect is used in Crossbar Mode to remove data path logic for master and slave connection paths that are not required. A SASD set of connected data configuration consumes even less logic because a single data path is shared by all devices and only one transaction is outstanding at a time.
3. Reduce use of multi-threading in AXI memory mapped IP including reduced values of issuance or acceptance. Reducing use of threads and transaction pipelining simplifies the transaction handling logic of the AXI Interconnect, but throughput could be impacted.
4. Avoid using AXI3 or AXI4 narrow bursts. Narrow bursts are defined in the AXI protocol but are generally not used by master IP. When a master IP specifically designates that they do not issue narrow bursts, some slaves (such as memory controllers) can detect that they will therefore never receive a narrow burst transaction and can omit narrow burst support logic.
5. Minimize protocol conversions and use AXI4-Lite where possible. Protocol conversion to AXI3 slaves utilizes logic. The AXI4-Lite protocol requires less logic to support, especially when all devices on an AXI Interconnect are AXI4-Lite type. Using AXI4-Lite protocols and grouping AXI4-Lite IP into a separate subsystem can reduce logic.
6. Where appropriate, segment interconnects into smaller, less complex subsystems where each subsystem can be optimized as described in the previous steps. This requires analysis of the protocol types, bandwidth, and master/slave connectivity. Grouping IP into subsystems that minimize connectivity requirements and minimize the number of conversion operations can reduce logic.
7. Reduce data path width and minimize size and width conversions. Design systems to the minimum required data path width while also minimizing width conversions.

8. Be careful not to inadvertently mismatch the AXI Interconnect core data width or core clock with the width and clock of all the attached endpoints; this can result in an excessive number of conversions. If possible, handle width conversion inside the user IP instead of using a general-purpose memory mapped AXI width converter.

   A protocol-compliant AXI memory mapped width converter block is complex due to issues like address calculation, multi-thread support, transaction splitting, unaligned bursts, and arbitrary burst length.

   If width conversion can be performed more efficiently in the user IP or in the application domain before reaching the AXI interconnect, the overall area is reduced.

Timing and Fmax Optimization Guidelines

1. Turn on register slices where appropriate. Register slices act as AXI pipeline stages to break combinatorial timing paths across the register slice. AXI Interconnect provides an optional register slice at the boundary of each attached endpoint. The FIFO Generator can also generate standalone instances of AXI register slices. Different register slice types and the granularity to set them on individual AXI channels provides fine grain control of register slices placement.
2. Large and complex IP blocks such as processors, DDR3 memory controllers, and PCIe bridges are good candidates for having register slices enabled. The register slice breaks timing paths and allows more freedom for Place and Route (PAR) tools to move a large IP block away from the congestion of the interconnect core and other IP logic.
   a. Overuse of register slices, especially in relatively full designs, can become counter-productive to timing by increasing the area and therefore the congestion for PAR tools.
   b. As required by the AXI specification, user IP must avoid combinatorial paths between inputs and outputs of the same AXI interface. This AXI protocol rule helps improve overall system timing.
3. Reduce data path width and minimize size/width conversions (as described in Size/Area Optimization Guidelines, step 7).
4. Where appropriate, segment interconnects into smaller or less complex subsystems where timing critical IP can be isolated away from non-critical IP. For example, a group of low bandwidth IP can be placed on a slower clock, smaller data width AXI Interconnect to free up logic and congestion from the higher performance IP running at higher clock rates and wider data paths.
5. Separate IP using register slices then floorplan the IP blocks (this is an advanced strategy). After placing register slices to provide timing isolation, IP blocks can be floorplanned further away from the interconnect core to reduce congestion around that block core.

Throughput and Bandwidth Optimization Guidelines

1. Increase clock frequencies using timing optimizations described in Timing and Fmax Optimization Guidelines, page 107. Increasing clock frequency, such as through the use of register slices to break long combinatorial paths can improve overall bandwidth.
2. Increase data path widths. Wider data paths carry more information per clock cycle.
3. Turn on data path FIFO buffers. Buffers can provide elasticity to hide temporary stalls or backpressure in the data flow. Use of the additional block RAM FIFO option, to delay `AWVALID/ARVALID` until FIFO occupancy permits interrupted burst transfers, can further improve throughput at the expense of increased latency.
4. Segment interconnects to group high performance IP together and place lower performance IP in a separate interconnect.
   Isolating high performance IP into a smaller subsystem permits greater flexibility to optimize that subsystem for higher throughput.
5. Increase transaction burst length. Longer bursts reduce the potential for stall cycles caused by address arbitration and control logic overhead.

   Longer bursts also signal to the AXI slave the intent to move a large amount of contiguous data so that slaves, such as memory controllers, can better optimize their response, and Reduce the relative amount of AXI address channel bandwidth.

   This reduces address channel congestion around the shared address arbiter logic in the AXI Interconnect.
6. Increase transaction pipelining including issuance and acceptance. Pipelining transactions allows arbiters and control logic in the slaves to work ahead on the next transaction while completing a previous transaction. This helps to reduce stalling due to arbitration/control cycles between back to back transactions.
7. Exploit parallelism of Sparse Crossbar AXI Interconnect. In Sparse Crossbar Mode, the AXI Interconnect supports parallel data flow when multiple masters transfer data to multiple independent slaves.
8. Avoid issuing read/write address requests until the IP ensures it can provide data while inserting minimal idle cycles in the data stream. Otherwise when a read or write data transfer is in progress, stalling the data phase of the transaction could prevent the AXI Interconnect from servicing other read or write data transfers. If the master or slave stalls, it could be blocking other devices, limiting system throughput.

   For higher throughput, design IP to request reads or writes when they are ready to be serviced with minimal stall cycles. The use of buffering might be beneficial. The worst case is a very slow AXI master requesting write bursts. When the slow master is granted arbitration, it will block other writes to the same slave until it completes its slow write transaction; this can take many clock cycles to transfer each beat of data.

The use of Data Path FIFOs (with delayed `AWVALID/ARVALID` feature) in the AXI Interconnect can help mitigate the system throughput impact of slow masters.

Latency Optimization Guidelines

1. Minimize clock and width conversions. Clock and width conversion require logic that adds additional cycles of latency.
2. Avoid using AXI3/AXI4 narrow bursts. Some AXI slave devices such as memory controllers must use logic to internally convert narrow bursts to full width bursts. This packing logic adds latency. If all masters connected to a given slave can designate that they do not perform narrow bursts, the narrow burst logic in the slaves can be disabled, thereby reducing area and latency.
3. Increase arbitration priority of latency sensitive masters. If some masters are more latency sensitive than others, increasing the priority of the latency sensitive master helps its requests to be serviced more quickly.
4. Reduce transaction burst lengths to prevent prolonged head of line blocking. Long bursts lengths can tie up data paths for longer periods of time while latency sensitive masters have to wait. Reducing burst length provides more frequent arbitration cycles where a latency sensitive master can gain access.
5. Increase clock frequency while trying not to using register slices. This reduces the absolute latency time. If register slices are not added, the number of clock cycles of latency does not change, only the period of each clock cycle.
6. Control Issuance/Acceptance of pipelined transactions from competing IP that are not latency sensitive. Head of line blocking can be introduced by high numbers of pipelined transactions. By limiting issuance/acceptance, the number of pipelined transactions is limited so that there are fewer potential transactions pipelined ahead of a latency sensitive transaction.
7. Arrange system address map and address access patterns to exploit row/bank management features of AXI DDRx (MIG) memory controllers.

   Accessing address locations of open banks and rows (pages) of memory reduces DRAM memory access time.

8. Exploit parallelism of crossbar AXI Interconnect or segment interconnects to reduce congestion and shorten path from latency critical master to slave. AXI Interconnect can be segmented, grouped, and optimized to arrange the latency sensitive masters closest to the slaves they wish to access.

Ease of Use and Debug Optimization Guidelines

1. Greater ease of use is accomplished by leaving each IP in its native, most convenient clock, width, and protocol, and using the per-port configurability of the Interconnect to adapt to the IP.
2. Using full crossbar connectivity provides more flexibility to change active transaction source and destinations, whereas sparse connectivity limits the flexibility of which masters can communicate with which slaves.

TIP: *An even simpler solution is to use the Shared Address Shared Data (SASD) mode of the AXI Interconnect.*

   SASD mode permits only a single read or write transaction to execute at a time with no overlapping or pipelining of transactions. The SASD mode of the AXI Interconnect stalls transactions so only a single one at a time can progress. This eases the debug and understanding of transaction sequences.

3. The AXI4-Lite protocol is much simpler than the AXI3 or AXI4 protocol. If AXI4-Lite is sufficient for an IP, using it simplifies the design.
4. Reducing the use of threading and transaction pipelining makes the system easier to debug and analyze using the AXI ChipScope™ debug monitor. Threading and pipelining make it more difficult to correlate activity on each of the AXI channels with a logical transaction. High levels of threading and pipelining also might be more likely to expose functional bugs in user IP.
5. Enabling the AXI ChipScope monitor permits full waveform capture and triggering in hardware. This enables hardware runtime viewing/triggering of some or all AXI signals at the boundary of the AXI Interconnect. This can be used to help diagnose functional or performance issues in hardware.
6. AXI hardware protocol checkers also help detect and more quickly isolate the source of protocol violations due to functional errors.

AXI4-based Multi-Ported Memory Controller: AXI4 System Optimization Example

AXI4 MPMC Overview

You can create an AXI4-based Multi-Ported Memory Controller (AXI MPMC) using a combination of an AXI Interconnect and an AXI memory controller core (for simplification, this is referred to as *AXI MIG*). This permits multiple AXI4 masters to share a common physical memory.

The Interconnect can be configured in an *N* Master to 1 Slave mode with AXI MIG as the slave connected to the AXI Interconnect as shown in Figure 5-1, page 111.

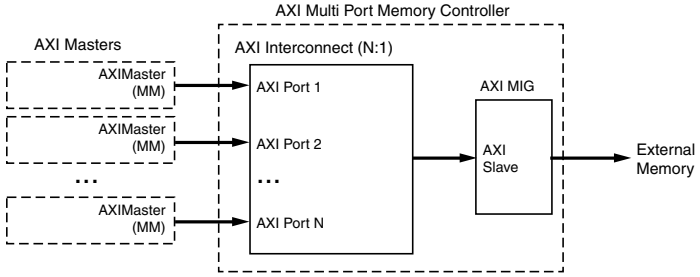


Figure 5-1: **AXI4 MPMC Block Diagram**

IP Configuration decisions across AXI masters, the AXI Interconnect, and AXI MIG can greatly affect the characteristics of the system, such as size, Fmax, throughput, and latency. By using the general optimization information described previously, the AXI MPMC can be tuned for a balance of size and performance. This section works through an example of applying system optimization techniques to tune the AXI MPMC.

For information on how to create an AXI MPMC design using EDK or Project Navigator, see the *AXI Multi-Ported Memory Controller Application Note, (XAPP739)* [Ref 5].

For an example of an AXI MPMC used in a high performance system, see *Designing High-Performance Video Systems with the AXI Interconnect, (XAPP740)* [Ref 19].

Initial Memory Controller Configuration

Assume the AXI MPMC is used for the purpose of transferring multiple data streams to and from a common physical memory. The first step is configuring the memory controller to meet the bandwidth requirements of the system. The AXI MIG supports physical memory widths of 8, 16, 32, 64, and 128 bits wide with a memory clock rate of 300 to 400 MHz for a -1 speed grade Virtex-6 device (check MIG documentation for other clock and width limitations). This equates with a 600 to 800 MHz data rate on the physical data lanes. Assume that four AXI masters are required, each consuming up to 100 MBytes/sec of bandwidth for reads and 100 MBytes/sec of bandwidth for writes with a native 32 bit x 48 MHz AXI4 interface. This implies 4x2x100 MBytes/sec = 800 MBytes/sec of total bandwidth is required.

For the memory controller configuration options, Table 5-3 can be derived:

Table 5-3: **Memory Controller Configuration Options**

Physical DDR3 Data Width (Bits)	Memory Clock (MHz)	Data Rate (MHz)	Max theoretical Bandwidth (MBytes/sec)
8	300	600	600
8	400	800	800
16	300	600	1200
16	400	800	1600
32	300	600	2400
32	400	800	3200
64	300	600	4800
64	400	800	6400
128	300	600	9600

The two smallest memory configurations that would meet the bandwidth requirements of the system are using a 16-bit DDR3 running 300 to 400 MHz memory clock rate, providing 1200 to 1600 Mbytes/sec of theoretical bandwidth (67% to 50% memory utilization at 800 Mbytes/sec).

In theory an 8-bit DDR3 running at 400 MHz meets the bandwidth also, but given overhead (lost clock cycles) for refresh, write-leveling, read-write bus turnaround time, and row/bank address changes, some more efficiency margin is required.

With AXI MIG, the AXI slave interface data width is natively equal to four times the physical memory data width and the AXI slave clock is ½ the memory clock frequency, so a 16-bit DDR3 @ 300 to 400 MHz directly corresponds to an AXI slave interface that is natively 64-bits wide at 150 to 200 MHz.

Initial AXI Interconnect Configuration

To be able to consume all the bandwidth from the memory controller, the AXI Interconnect core must be have at least the same bandwidth as the memory controller. Given the recommendation to avoid width and clock conversion that impact size and timing, the interconnect core and slave side port should be configured as 64-bits at 150 to 200 MHz to match the native AXI interface of the memory controller.

To configure the master side of the AXI Interconnect, note that the AXI master is natively 32 bits at 48 MHz. This requires a 32- to 64-bit size conversion for each master.

In addition, the 48 MHz AXI clock on each AXI master would result in an asynchronous clock conversion if the interconnect is running at 200 MHz.

Clock Conversion Recommendation

The recommendation for clock conversion is to use synchronous ratios over asynchronous ratios to reduce logic.

Instead of a 200 MHz Interconnect clock, the system can be configured to attempt to remove asynchronous clock conversions by employing a:

- 48 MHz AXI master clock
- 48 x 4 = 192 MHz AXI Interconnect clock
- 192 x 2 = 384 MHz memory clock

Note: The 48, 192, and 384 MHz clocks should be driven by the same Mixed Mode Clock Manager (MMCM) block to be phase aligned.

Figure 5-2 shows an example of the AXI Interconnect master side configuration.

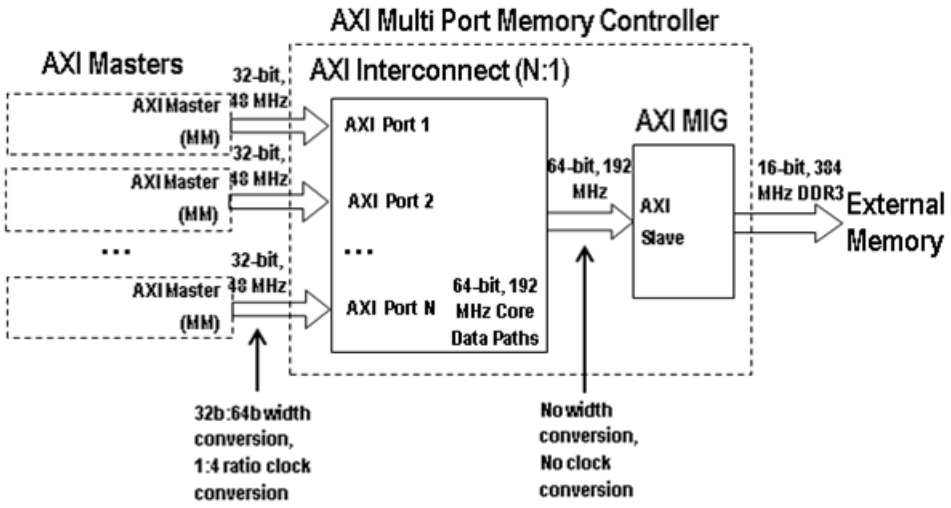


Figure 5-2: **AXI Interconnect Master Side Configuration Block Diagram**

AXI4 Master Configuration

The use of AXI4 transactions by the AXI4 master impacts the performance that can be obtained from the memory controller and system. Because this system requires significant bandwidth from the memory controller, maximizing the burst length of AXI4 transactions to 256 beats helps improve overall data bandwidth.

Maximize Burst Length

Longer bursts reduce address arbitration/control cycles and help keep the memory controller in the same row, bank, and read/write direction longer. Long bursts would normally impact latency, but assuming this application is not very latency sensitive and that data path FIFOs are enabled for elasticity, the use of long bursts should not result in head of line blocking/stalling.

No Narrow Burst Transactions

The AXI4 master should not issue any narrow burst transactions. Narrow bursts are defined in the AXI specification as transactions where the size of the AXI transaction is more narrow than the native data width of the interface. Such bursts are less efficient in terms of bus utilization and require extra logic in the memory controller to handle repacking of any narrow bursts into full width bursts.

In this example:

- Size AXI transactions issued by the masters to 32-bit (`AxSIZE = 0x2`).
- Enable the modifiable bit on AXI transactions (`AxCACHE[3]=1`) to ensure that any downstream upsizer can fully pack data up to wider widths. This allows costly narrow burst support logic to be removed from the memory controller.

In XPS, this is designated by the `C_SUPPORTS_NARROW` parameter that then allows XPS to automatically configure AXI MIG to omit narrow burst support logic. In a CORE Generator™ context, you must manually configure AXI MIG to omit narrow burst support logic.

Pipeline Transactions

Design the AXI4 master to pipeline transactions so it can issue new address requests while servicing the data transfers for previous transactions. Pipelining transactions helps overlap address and control cycles with data transfer cycles to improve data path efficiency and throughput. However, new address requests should not be made until it is ensured that the master can supply sufficient write data or has sufficient ability to accept read data to complete a full burst with minimal stalling. A master that issues an address request and excessively stalls the data transfer phase of its requested transaction could cause backpressure that could eventually stall or slow the whole system.

Single Thread Transactions

Design the AXI4 master so that it operates using only a single thread for all transactions (declared using the `C_SUPPORTS_THREADS=0` parameter). By not using multiple threads, the logic in the AXI4 master can be simplified because it can be designed to rely upon write responses and read data being returned in order. The use of a single thread also benefits the AXI Interconnect performance because the upsizer is active in this example system.

Upsizers in the AXI Interconnect stall when changing ID threads so using a single thread avoids stalling of transactions passing through the upsizer. Ensure that the AXI4 master declares itself not to use threads so that AXI Interconnect can be configured to omit its multi-thread support logic which reduces area and improves timing. Using a single thread also makes debug easier because AXI transactions observed in the ChipScope monitor are easier to decode and correlate across a system.

Refining the AXI Interconnect Configuration

After a first pass to establish the basic configuration of AXI MIG, AXI Interconnect, and the AXI4 master of the user, the user can then perform a second pass at refining the AXI system configuration.

Independently Configure Converter Banks

When fine tuning the configuration of the AXI Interconnect, it is useful to understand the AXI Interconnect converter bank block. The converter bank handles size, clock, and protocol conversion in addition to register slice and data path FIFO features.

The converter bank can be independently configured at each endpoint of the AXI Interconnect, as shown in Figure 5-3.

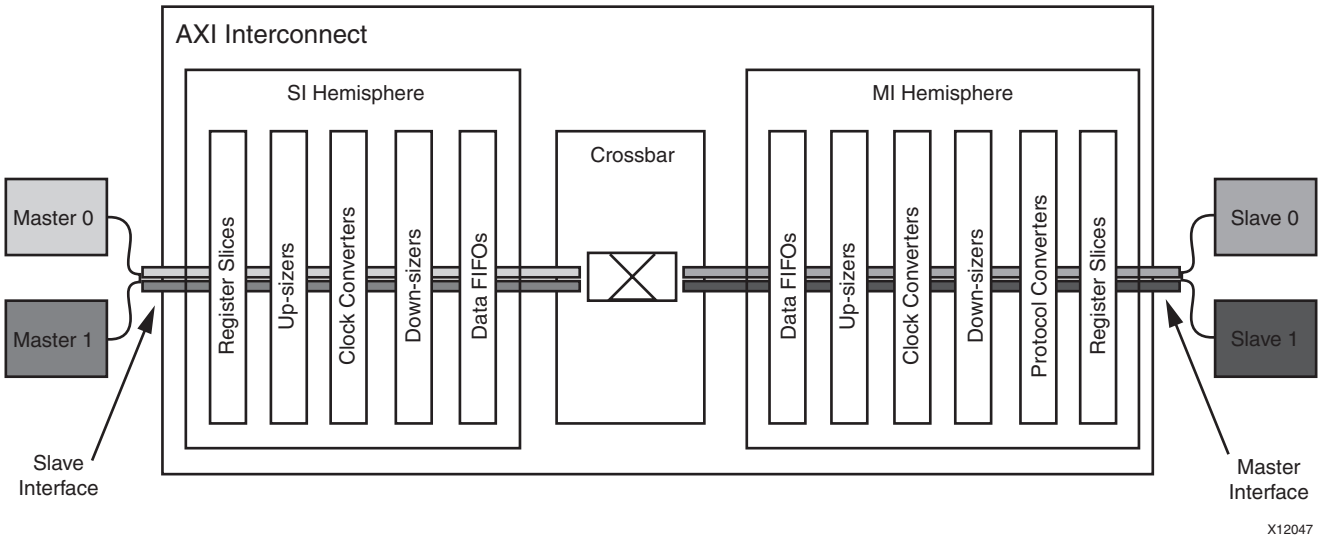


Figure 5-3: **AXI Interconnect: Crossbar Block Diagram**

Notice that from the perspective of the attached AXI master, shown in Figure 5-3, the data path FIFOs are positioned after the upsizer and the clock converter so that the FIFO interfaces to the interconnect core at its higher native width and clock.

Because the AXI masters are at a relatively lower bandwidth than the memory controller (1/2 width, ¼ clock frequency), turning on data path FIFOs allows the interconnect to buffer up the wider width transactions to and from the memory controller and service each of the AXI masters at their slower rates on the other side of the FIFOs. Data path FIFOs reduce stalling of the memory controller due to the slower data rate AXI masters. The AXI Interconnect offers data path FIFOs in options of 32 deep or 512 deep FIFOs. Because the AXI4 master is generating long bursts up to 256 beats in length, configure the FIFOs as 512 deep to fit an entire burst.

The data path FIFOs have an optional feature, called *Packet Mode*, to delay `AWVALID/ARVALID` until FIFO occupancy permits interrupted burst transfers downstream.

The Packet Mode feature:

- Causes write address requests to be withheld from the crossbar until the write data path FIFO has buffered all the data for the transaction.
- Causes read address requests to be withheld from the crossbar until the read data path FIFO has sufficient vacancy to store the entire transaction.
- Ensures that the crossbar does not see a transaction request until the data path FIFO can guarantee that it can source/sink the entire transaction at the full bandwidth of the crossbar without introducing stall cycles in the data transfer.
- Is especially useful in situations similar to the example design, shown in Figure 5-2, page 113, where the master has a relatively lower bandwidth than the slave (memory controller).

Timing Considerations

For timing, the AXI Interconnect should be configured to enable register slices at the interface to the memory controller. Because the AXI interface memory controller operates at the highest width and clock frequency in the system, it is likely a critical path unless a register slice is turned on.

A Type 8 register slice can be enabled on all five channels at the AXI interface of the memory controller to allow the AXI Interconnect to optimize the kind of register slice best suited to each AXI channel.

Note: A register slice at the AXI master interface is not required. This is because the AXI master and the upsizer are both clocked by the slower 48 MHz side of the clock converter.

Also, the clock converter acts as a register slice because it provides timing isolation between 48 MHz and 192 MHz clock domains.

Setting Issuance and Acceptance Values to 2 or Higher

Issuance and acceptance values at each port of the AXI Interconnect can be optimized to support transaction pipelining and to limit the pipelining so that head-of-line blocking is reduced. The default issuance assigned to an AXI masters is 1, unless configured or designated otherwise. An issuance of 1 minimizes logic but does not permit transaction pipelining. Set the issuance to a value of 2 or higher.

Because the target system seeks to maximize throughput, you can calculate the maximum number of outstanding transaction possible without overflowing the data path FIFOs. The data path FIFOs are 64 bits wide x 512 deep as described above. That is equivalent storage to 32 bits wide x 1024 deep.

If the AXI4 master is generating AXI transactions of maximum length 256, then up to four transactions fit into the data path FIFOs.

The AXI Interconnect supports issuance and acceptance values of 1,2,4,8,16, and 32. Reasonable values of issuance for each AXI master would therefore be 2 or 4.

- Given that there are 4 masters, an issuance of 2 means that memory controller would need an acceptance of 8 to fully pipeline 2 transactions from each master.
- Given that transactions are all long bursts, pipelining more than 8 transactions at the memory controller becomes excessive. An issuance setting of 4 at the masters is too high because it would require the slave to consume up to 16 transactions to utilize.
- Given that master issuance of 4 might be excessive while an issuance of 1 prevents transaction pipelining, a setting of 2 is reasonable.

Adding a Processor to the AXI MPMC System

Adding a processor to the example AXI MPMC system complicates the optimization of the system because processors tend to be very latency sensitive with respect to their performance. If the processor must also share the memory controller to run complex software such an operating system or protocol stack, you must take more care to balance the low latency requirements of the processor with the high throughput requirements of the other AXI masters.

Processor traffic could interfere with other devices resulting in reduced throughput from other masters. This is due to random memory accesses that disrupt the row/bank access patterns of the other devices and because the processor can generate a number of small length transactions. Small length transactions corresponding to 4- or 8-word cache lines can consume several memory clock cycles for row/bank access time, read/write turn around, and so forth. Therefore, the actual data bandwidth transferred by the processor might be small, but because they can disrupt the otherwise linear, long burst access patterns of the memory controller, their traffic actually displaces a much larger amount of the theoretical system bandwidth.

For example, 10 Mbytes/sec of delivered data bandwidth to the processor, might actually displace the equivalent of 100 MBytes/sec of the theoretical bandwidth of the memory controller. Optimizations to improve processor performance could force a trade-off in system throughput, further eroding the bandwidth available to other masters.

Considerations When Adding a Processor

If you need to add a processor to the system, you must consider:

- If the memory width or clock should be increased to provide more available margin.
- Whether to reduce the burst length of the other AXI masters to reduce the time that a processor waits for a burst transaction to complete.
- If the highest arbitration priority can be granted to the processor to minimize its latency.
- If the issuance/acceptance values for other devices might be reduced to limit head of line blocking due to pipelined transactions.
- If the system clocking can be altered to favor the memory path of the processor having no clock conversion or only synchronous clock conversion.

  Note: The MicroBlaze™ processor can support a native 128-bit and 256-bit and 512-bit wide AXI interface. This is an example of application domain size conversion that is more efficient that generic AXI width conversion. This MicroBlaze wide cache configuration is ideal for connecting to an equally wide memory controller to remove the latency impact of size conversion.

The optimizations described to improve processor performance are often the opposite step for maximizing system throughput.

Therefore, either more margin is needed by using a larger memory controller or you must carefully optimize your software (to minimize cache misses) and be more willing to experiment with the system to find the right balance between latency and throughput.

Additional Potential Optimizations for AXI MPMC

The previous AXI MPMC optimization information steps through an example of working through the design optimization process, while attempting to balance tradeoffs between various design criteria.

The following subsections describe further optimizations ideas than might be applicable. These optimizations might or might not be suitable for a given AXI MPMC design and could require experimentation to see if the technique is useful in a given situation.

AXI Interconnect: Shared Address Shared Data Mode

If there is enough extra unused bandwidth, the AXI Interconnect can be configured in Shared Address Shared Data (SASD) mode.

In this mode, the AXI Interconnect core is simplified to operate on only a single read or a single write at a time and even shares read and write addresses over the same wires.

This mode removes support for pipelined transactions and prevents simultaneous read and writes, but significantly reduces logic. Given that long bursts are used, the penalty of stall cycles between transactions and lack of simultaneous read and write data flow might be acceptable within the bandwidth requirements of the system. SASD also makes system debug and waveform analysis of AXI transactions substantially easier. SASD is also generally more lenient about functional bugs and protocol violations from endpoint IP.

Separate IP Groups into Separate AXI Interconnect Subsystems

If an AXI MPMC design has many masters, and the design has difficulty meeting timing, one possible strategy is to group two or more IP into a separate *N* x1 AXI Interconnect that then feeds the main AXI Interconnect. This breaks a wide fan-in Interconnect into multiple smaller fan-in Interconnects. Each smaller Interconnect is easier to route and meet timing, and also provides a greater range of options for the location of register slices, FIFOs, size, clock, and width converters.

For example, when two AXI Interconnects connect directly to each other, a set of back-to-back register slices can be enabled using one register slice from each adjacent interconnect. This can be used to span longer routing distances in large AXI MPMC systems.

In some cases using multiple AXI Interconnects can even reduce overall system size.

When an AXI MPMC requires a large number of upsizers, especially with large steps like 32- to 128-bits, separating the masters into subgroups using smaller width AXI Interconnects can reduce the number of upsizers which consume area and impact timing.

Note: In XPS, the connection of two cascaded AXI Interconnect instances together requires that an AXI-to-AXI Connector IP be instantiated. This bridge IP provides a tool mechanism for XPS to cascade interconnect, but logically it contains only wires and consumes no logic.

Debug and Analysis: Using AXI ChipScope Monitor and AXI Hardware Protocol Checkers

In XPS, the AXI ChipScope™ Debug monitor is a feature that provides waveform capture and triggering of AXI interface signals in hardware. The AXI ChipScope monitor can be used to help debug functional issues in hardware or to help diagnose performance issues.

IMPORTANT: *Analyze Complex System Activity.*

You can place multiple AXI ChipScope monitors around the system and cross-trigger between each of them to analyze more complex system level activity. The AXI Hardware Protocol Checker feature is available that can trigger the AXI ChipScope monitor when some types of AXI protocol violations occur. The AXI Hardware Protocol Checker can more quickly isolate the source of protocol violations.

Floorplanning

AXI IP connected to the AXI Interconnect can be floorplanned to improve placer results and reduce routing congestion. To make floorplanning easier in large FPGAs, enable extra register slices to provide a more distinct flip flop-based boundary at the AXI IP interface.

Note: After any significant changes to the AXI Interconnect configuration, floorplan locations might need to be rechecked and updated as necessary. Otherwise subsequent changes to the AXI Interconnect, such as turning on data path FIFOs, can change the footprint and necessary placement of the AXI Interconnect.

Cadence Bus Functional Models

XPS supports instantiation of AXI Bus Functional Models (BFMs) and AXI Protocol Monitors for use in simulation to exercise and test AXI IP. This feature is a Cadence product available for XPS.

Xilinx recommends that you use these BFMs for verification of user IP under development. Given the potential complexity of understanding AXI transactions, especially across pipelined transactions and multi-threaded traffic, it can be extremely difficult to debug subtle functional errors or isolate the root cause of protocol violations solely in hardware. The simulation domain is usually a far less expensive method for verifying and debugging new AXI IP before use in complex systems.

See the *AXI Bus Functional Models User Guide (UG783)* [Ref 10] and the *AXI Bus Functional Model Data Sheet (DS824)* [Ref 11] for more information.

Simpler but Wider Interconnect and Memory Controller

Another potential strategy for an AXI MPMC system is to oversize the width of the memory controller and AXI Interconnect core, such as by doubling the memory width to double the theoretical system bandwidth. By adding extra potential system throughput, the configuration of the rest of the system is much more simple.

For example, a system initially requiring a 16-bit DDR3 at 400 MHz with the AXI Interconnect core configured as 64-bits at 200 MHz could be reconfigured with a 32-bit DDR3 at 300 MHz and an AXI Interconnect configured as 128 bits at 150 MHz.

The extra bandwidth from doubling the physical memory width can be used to allow greater system simplifications, including:

- Reducing AXI clocks from 200 MHz to 150 MHz to improve timing:
  - Reductions in clock frequency could permit register slices to be turned off
  - Reductions in clock frequency could permit use of a slower speed grade FPGA device
- Allowing the crossbar to be reconfigured into SASD (this disables transaction pipelining and multi-threading support):
  - Simplifies system debug
  - Provides room for future system bandwidth expansion (you can later increase clock frequencies, enable crossbar, and so forth.)
- Allowing masters to use shorter burst lengths:
  - Reduces latency or reduces system FIFO/buffering requirements

Note: Increasing memory controller and AXI Interconnect data width could introduce new size conversion requirements and board-level requirements into the system that might offset these AXI system simplifications so analysis and experimentation is required to determine if this approach is an overall improvement for the given application.

Cascading Interconnects

In some situations where the AXI Interconnect is a large Nx1 configuration with upsizing, width conversion, with enabled data path FIFOs, it might be beneficial to deploy multiple cascaded interconnects instead of a single large Interconnect. This is often the case in high performance systems that contain high bandwidth memory controllers shared by multiple lower bandwidth masters.

For example, consider an AXI MPMC with 8 masters each with 32 bit at 100 MHz interfaces connected to a memory controller with a 256 bit interface at 200 MHz as shown in Figure 5-4, page 122.

Assume that each master requires 70% of the available bandwidth of the interface:

$$32 \times 100 \times 0.7 = 2240 \text{ mbits/sec.} \qquad \textit{Equation 5-1}$$

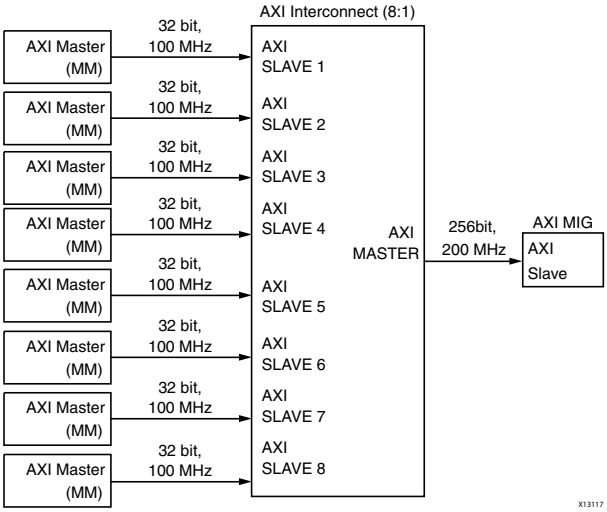


Figure 5-4: **AXI MPMC with 8 Masters**

For highest performance, the interconnect would enable 32 to 256 bit upsizing, 100 MHz to 200 MHz clock conversion, and Packet Mode data path FIFOs as shown in Figure 5-5.

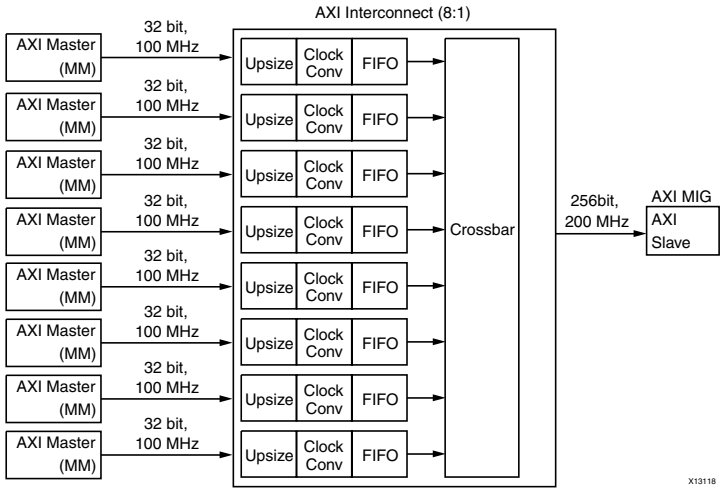


Figure 5-5: **Enabled Upsizing**

The order of the sub-modules in the interconnect is:

Upsizer followed by Clock Converter followed by FIFO

This ordering is also shown in Figure 5-3, page 115 as the architecture of the AXI Interconnect.

Note: The clock converter and FIFO are located after the upsizer, therefore each block essentially contains a 256 bits wide internal data path for read and write channels.

As a rough estimate, assume this that each sub-module uses two FFs and two LUTs per data path bit. The approximate size of the data path logic used by the converter blocks is estimated with the following equation:

Size (LUTs and FF) = <number of ports> x <number of sub-modules> x <data width in bits>x < number of LUTs and FFs per bit> *Equation 5-2*

or

*8 ports x 3 sub-modules x 256 bits data path x 2 channels for read/write * 2 LUTs and FFs per bit = 24576 LUTs and 24576 FFs.* *Equation 5-3*

It can also be assumed that generally the size of the data paths dominates as the percentage of the total size of the interconnect relative to the size of the control path logic.

An alternative implementation of the design to explore is to use two levels of cascaded Interconnects to partition the design into multiple smaller, more gradually scaled Interconnect building blocks as shown in Figure 5-6.

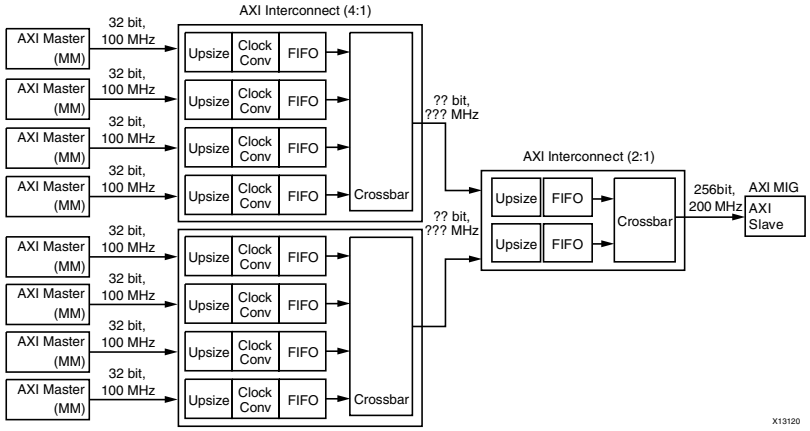


Figure 5-6: **4x1 Interconnect and 2x1 Interconnect**

The scaling shown in Figure 5-6 is 4x1 Interconnect + 4x1 Interconnect followed by a 2x1 Interconnect that uses three smaller Interconnect blocks instead of one large block.

This is likely a more optimal design than a 2x1 Interconnect + 2x1 Interconnect + 2x1 Interconnect + 2x1 Interconnect design followed by 4x1 Interconnect because it uses five blocks and therefore, is more likely to lead to optimal partitioning than the one shown in Figure 5-5, page 122.

Using the partitioning shown in Figure 5-7, the next step is to find the best size and width for the intermediate interface between the cascaded interconnects.

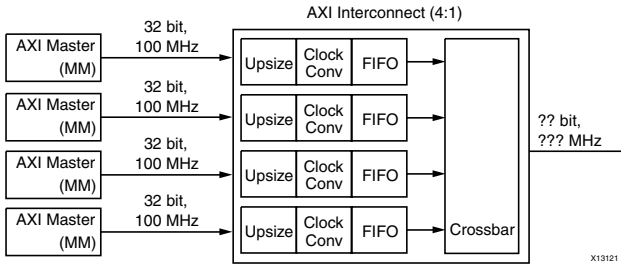


Figure 5-7: **Partitioning for Best Size and Width**

The options are:

- Width can be 32, 64, 128, or 256 bits wide.
- Cock frequency is either 100 or 200 MHz (to avoid the use of asynchronous clock converters).

The choice of intermediate signal width and clock frequency determine where width converters and clock converters are placed in the system and what their data path sizes are.

IMPORTANT: *Assume that Packet Mode data path FIFOs are required in all Interconnects because that is needed to ensure there are no bottlenecks in the data flow toward the memory controller.*

Each 4x1 Interconnect aggregates traffic from four masters each requiring 70% of the bandwidth of the 32 bits x 100 MHz interface for a total maximum throughput requirement of 4x32x100x0.7 = 8960 mbits/sec.

By looking at the maximum throughput of the connections as detailed in Table 5-4, some options can be eliminated because the given intermediate interface configuration cannot support this aggregate throughput requirement.

Table 5-4: **Maximum Connection Throughput**

Width (Bits)	Clock Domain (MHz)	Raw Throughput (mbits/sec)	Capable of 8960 mbits/sec
32	100	3200	No
32	200	6400	No
64	100	6400	No
64	200	12800	Yes
128	100	12800	Yes
128	200	25600	Yes
256	100	25600	Yes
256	200	51200	Yes

Table 5-5 shows the results of using Equation 5-2 for estimating the size of the interconnect sub-module data paths, for each intermediate interface option that offers sufficient throughput.

Table 5-5: **Interconnect Size by Width and Clock Domain**

Width (Bits)	Clock Domain (MHz)	Size Estimate
64	200	10240
128	100	14336
128	200	16384
256	100	20480
256	200	26624

Therefore, if the intermediate interface is configured as 64 bits x 200 MHz, the data path area for the cascaded scaled Interconnects is estimated at 10240 LUTs/FFs compared to 24576 LUTs/FFs used by the original single Interconnect design.

TIP: *Cascading Interconnects reduces area and potentially improves timing through reduced routing congestion, but there is a trade-off in higher system complexity and higher latency.*

Figure 5-8 shows the optimized system configuration.

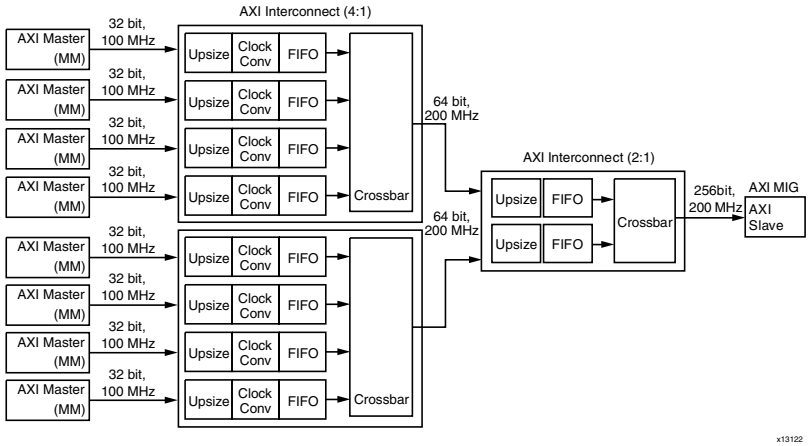


Figure 5-8: **Optimized System Configuration**

Using this approach of estimating the size of scaled Interconnects and removing unfeasible options, you can narrow down the choices to experiment with cascaded Interconnects to further reduce area.

IMPORTANT: *Verify any estimates through implementation of the complete system to validate the improvement.*

Common Pitfalls Leading to AXI Systems of Poor Quality Results

This section describes common pitfalls designers might encounter leading to the design of AXI systems that are of a larger than expected area, have poor performance, or have poor timing.

Oversizing a Memory Controller

AXI Virtex-6 MIG supports DDR3 physical memories in 8, 16, 32, 64, and 128 bit widths, which translates into a 4 times wider native AXI data width of 32, 64, 128, 256, and 512 bits wide.

If a system contains only a 32-bit microprocessor and associated AXI4-Lite peripherals, then connecting it to a 64-bit physical DDR3 memory is wasteful of logic and would actually degrade performance.

Such an AXI MIG would have very large area for the physical interface logic and the 256 bit data paths inside the AXI MIG. Also, the native 256-bit AXI interface would have to be upsized from the 32-bit AXI interface of the processor adding further area and latency.

IMPORTANT: *Size the AXI MIG to meet the bandwidth needs of the system while minimizing unnecessary size conversions.*

This situation can be common when using fixed evaluation boards, like the ML605, that contain a 64-bit DDR3 DIMM. You might be working from a reference design containing an AXI MIG configured for a full, 64-bit DDR3 DIMM that is oversized for a simple MicroBlaze processor application.

Incorrect Core Data Width or Core Clock for AXI Interconnect

Incorrectly setting the core data width or connecting the wrong core clock to the AXI Interconnect can severely impact the system.

For example, consider a system with five masters and five slaves connected to AXI Interconnect. Assume that each master and slave is 64-bits wide at 100 MHz.

If the AXI Interconnect is also configured to be 64-bits wide at 100 MHz, then no clock or width converters are used. However, if the interconnect core data width is accidently configured to be 32-bits wide at 75 MHz, then the same system would then need to incur a 64:32 bit downsizer for each master, a 32:64 bit upsizer for each slave and asynchronous clock converters at every master and slave. The extra cost of 10 size converters and 10 asynchronous clock converter results in poor timing, very large area, and very high latencies in the system.

Additionally, the throughput in the system is greatly constrained because wide 64-bit AXI traffic from the masters and slaves are funneled through a much more narrow and slower 32-bit data path in the AXI Interconnect. This would result in stall cycles between every data beat the masters and slaves are trying to transfer.

TIP: *Watch For Wrong Clock and Interconnect Data Widths.*

Because XPS handles width and clock conversion configurations automatically, connecting the wrong interconnect clock or setting the wrong interconnect data width causes XPS to automatically activate all the necessary conversions to make the system function. The result is a system that might appear to function and completes all AXI transactions, but the system bandwidth, area, latency, and timing could be very undesirable.

Overuse of Register Slices

In general, register slices are useful for helping to close timing in a system. However, excessive use of register slices can be counterproductive.

For example, enabling all register slices on all AXI interfaces in a large system might increase the area of a system leading to routing congestion and longer map, place, and route times while not improving timing.

TIP: *Incrementally Add Register Slices.*

The recommended approach is to incrementally add register slices when timing fails starting at the interfaces with highest clock frequencies and data widths. Register slices might also be needed for large crossbar interconnects or at AXI interfaces where size conversion is performed. If large numbers of register slices are required to meet timing in a large system, floorplanning might be needed to help guide the place and route tools.

TIP: *Do Not Place Register Slices on AXI4-Lite IP.*

Generally, Xilinx recommends that you do not place register slices on AXI4-Lite IPs.

The recommended approach is to segment AXI4-Lite IP onto a separate SASD AXI Interconnect and clock this AXI Interconnect and its attached AXI4-Lite IP at a slower common clock frequency to better meet timing.

If timing improvement is needed on an AXI4-Lite Interconnect, first enable the special internal register slice rank inside the SASD Interconnect; then add register slices only on specific channels of AXI4-Lite IP that fail timing.

TIP: *Watch For Wrong Register Slice Types.*

Also, using the wrong type of register slice can lead to undesirable effects.

For example, type 1 register slices support back-to-back data beats without inserting stalls while type 7 register slices use less area, but insert a stall after every data transfer.

- The type 7 register slice is ideal for AXI4-Lite interfaces or for AW, AR, and B channels of an AXI interface where back-to-back transfers do not occur or occur infrequently.
- The type 1 register slice is designed for R and W channels where burst transactions occur.
- For convenience, a type 8 register slice option is provided which automatically switches between type 1 and type 7 based on the AXI Interconnect configuration. Type 8 is recommended and should only be overridden when specifically warranted.

Skipping Simulation-Based Verification of New IP

AXI4 provides a rich protocol that can:

- Scale into more complex systems
- Support sophisticated protocol features such as multi-threading and transaction pipelining
- Manage transaction ordering and completion rules to manage traffic among multiple AXI masters and slaves in a system.

The richness of the AXI protocol and the possible concurrency of data transfer in a crossbar, make hardware-only based debug and verification of new AXI IP much more challenging.

RECOMMENDED: *Verify new IP in simulation using AXI Bus Functional Models (such as the Cadence® BFM available for XPS) and AXI protocol checkers/assertions (available from Cadence or from the ARM® website).*

Simulation-based verification results in far shorter debug cycle time, easier identification and isolation of functional problems, and greater variation of AXI traffic than hardware-only based verification.

Hardware-only based AXI IP verification requires full synthesis and Place and Route (PAR) time per debug cycle, and the visibility of signals from an AXI ChipScope monitor is more limited than in a simulation domain. The potential complexity of AXI4 traffic even in a relatively typical system makes hardware-only verification very expensive.

However, if you must rely on hardware-only AXI IP verification, Xilinx recommends that the AXI Interconnect be configured as simply as possible.

For example, use SASD (which limits issuance/acceptance to 1), and minimize the use of converter bank functions (size conversion, clock conversion, data path FIFOs, and so forth). Register slices can also be enabled for hardware-only verification because register slices acts as a filter for traffic patterns that can insulate the system from some protocol violations.

Enable AXI ChipScope monitors and hardware protocol checkers at strategic points in the system when performing hardware-only verification.

Using Base System Builder Without Analyzing and Optimizing Output

Base System Builder (BSB) provides a starting point for a functional AXI system that can run on an evaluation board or be used to begin a custom board based design. However, the system produced by BSB is a point solution within a broad solution space that AXI IP can offer.

BSB offers a basic choice between an area- or throughput-optimized design to establish a baseline architecture for the system.

RECOMMENDED: *Optimize, Adapt, and Transform BSB Output.*

The AXI system output from BSB should still be further adapted, optimized, and incrementally transformed to fit the desired end application using the techniques described in AXI System Optimization, page 106. Failure to tune the output of BSB to meet the specific requirements of an application could result in poor quality of results and low performance.

The architecture and optimizations necessary for a good AXI IP-based solution can differ greatly from that of an IBM CoreConnect™ or a Xilinx MPMC-based system. The output from BSB for AXI systems might not be designed with the same type of system architecture as was output from BSB for CoreConnect or MPMC based systems. The output from BSB for AXI systems must be significantly modified to match similar area, performance, and features trade-offs as a CoreConnect or an MPMC system created by BSB.

AXI4-Stream IP Interoperability: Tips and Hints

Introduction

The AXI specification provides a framework that defines protocols for moving data between IP using a defined signaling standard. This standard ensures that IP can exchange data with each other and that data move across a system. The AXI protocol does not specify or enforce the interpretation of data; therefore, the data contents must be understood, and the different IP must have a compatible interpretation of the data.

This chapter provides information and presents concepts that help you construct systems by configuring your IP designs to be interoperable. Interoperability in the context of this document is defined as the ability to design two or more components in a system to exchange information and to use that information without extra design effort.

This chapter also describes areas where converters or additional effort are required to achieve IP interoperability.

Generally, components can achieve interoperability with other components using either or both of the following approaches:

- By adhering to published interface standards
- By making use of a “broker” of services (bridge) that can convert the interface of one component to the interface of another product.

Key Considerations

The key considerations for achieving IP Interoperability are:

- **Understand the IP Domain**: The interfaces used by the IP for exchanging information and the data type representation that is used for the transferred information can be classified by IP domain. Xilinx® IP generally follows a common set of guidelines to describe data contents and interface signaling within a given domain. This chapter focuses on the following main domains:
  - DSP/Wireless
  - Video
  - Communications
  - AXI Infrastructure IP domains
- **Follow the Published Standard for the IP Domain**: Chapter 3, AXI Feature Adoption in Xilinx FPGAs provides an overview of the adoption of AXI4-Stream by Xilinx IP. This chapter further describes the various AXI4-Stream interface conventions and guidelines for IP configuration and use.
- **Validate the Data**: Understanding the IP domain and following the published standard lets you focus your effort on the key elements to achieve IP interoperability; however, it is imperative that you confirm that the IP operates as expected in the system using simulation or hardware testing.

Figure 6-1 illustrates how you need to approach the design and development process from IP selection to IP configuration for implementing systems with a high degree of IP interoperability.

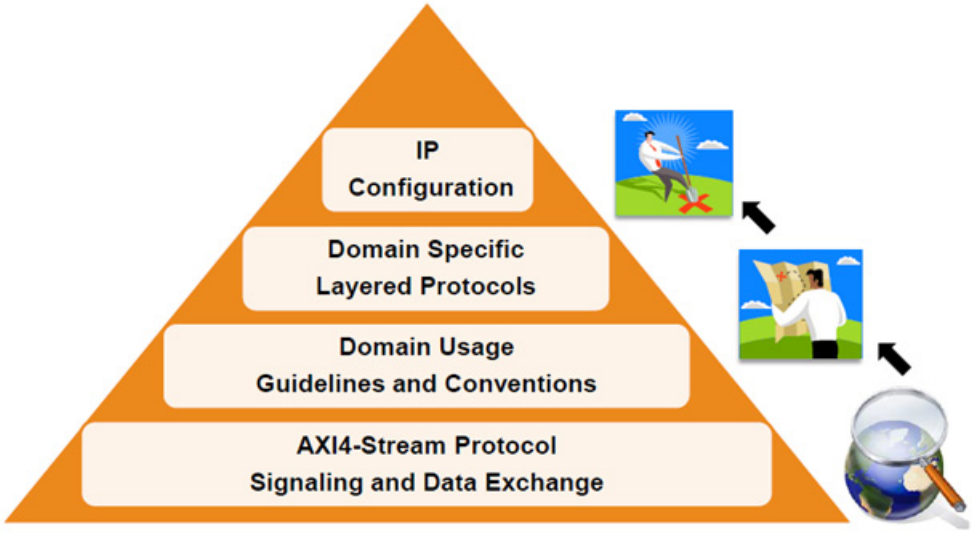


Figure 6-1: **Process Tier for Understanding IP Interoperability**

Begin the design process by understanding the AXI4-Stream protocol because it is the basis for data exchange. You can then move to higher levels of refinement by understanding the domain-level AXI4-Stream usage conventions, domain-level data organization and interpretation of data, and finally focus on the exact configuration settings and functions of each IP in the system. In this process, you narrow the solution space for each IP in the system.

AXI4-Stream Protocol

Review and use the AXI4-Stream signaling and data exchange interface protocol as described in Chapter 3, AXI Feature Adoption in Xilinx FPGAs. AXI4-Stream is an interface-level protocol for IP to exchange raw data. Building on top of the signaling protocol, the various IP application domains can then establish common data types to enable IP to use exchanged data. AXI4-Stream defines optional signals with default tie-off rules and byte-aligned data widths with width conversion formulas. Some of the key IP interoperability considerations to focus on at the AXI4-Stream signaling levels are:

- Use of optional signals between two IP being connected to each other, as shown in Figure 6-2, page 133.
- Data width of the connected interfaces.
- Burst length (such as the size of the data frame, block, or packet).
- Data representation (Layered Protocols).

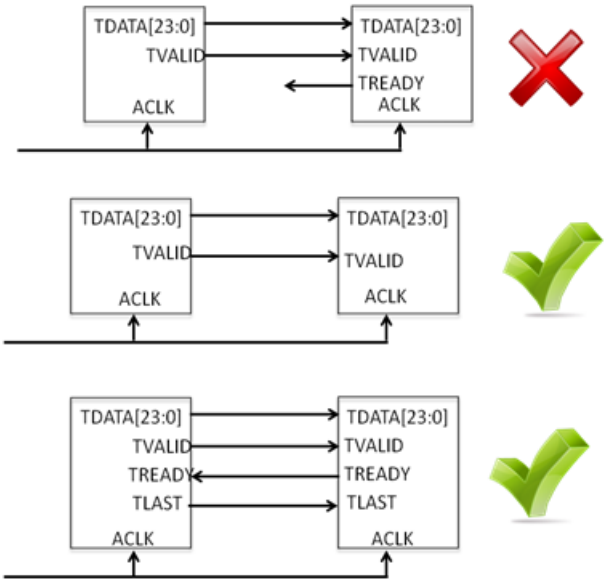


Figure 6-2: **Establish AXI4-Stream Signaling-Level Data Exchange Compatibility**

Domain Usage Guidelines and Conventions

Each IP application domain recommends common guidelines for usage of optional AXI4-Stream signals to facilitate IP interoperability. The four major IP application domains are shown in Figure 6-3.

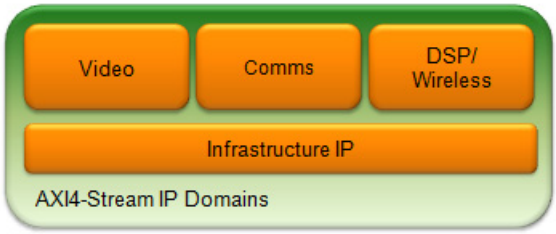


Figure 6-3: ***Figure 6+-3:*** **AXI4-Stream IP Domains**

Video IP

AXI4-Stream IP in this domain carries framed video and image pixel data, and exchanges only pixel data and video line and frame markers. Other signals associated with physical interfaces such as hsync, vsync, active_video, blanking, or other ancillary data signals are not carried over AXI4-Stream.

AXI4-Stream video IP supports backpressure and elasticity in data flow around the TVALID/TREADY handshake.

Pixel data retains the relative location in the datastream; however, video pixel timing relative to hsync or vsync at a physical display is not preserved.

AXI4-Stream video IP uses layered protocols to encode a variety of video formats and resolutions. For more details on video IP adoption of AXI4-Stream, see IP Using AXI4-Stream Video Protocol in Chapter 3.

Table 6-1 summarizes video IP domain AXI4-Stream signaling usage and guidelines.

Table 6-1: **Video IP Domain AXI4-Stream Signaling Usage**

Signal	Endpoint
ACLK	Used and Supported
ACLKEN	Limited Support
ARESTEN	Used and Supported
TVALID	Used and Supported
TREADY	Used and Supported
TDATA	Used and Supported
TID	Not Supported
TDEST	Not Supported
TKEEP	Not Supported
TSTRB	Not Supported
TUSER	Used and Supported
TLAST	Used and Supported

DSP/Wireless IP

AXI4-Stream IP in the DSP and Wireless IP application domain operates on numerical streaming data paths. IP in this domain exchange data in either blocking (with backpressure) or non-blocking (continuous) modes.

You can organize data in either Time Division Multiplexing (TDM) or parallel paths, and use optional AXI4-Stream interfaces to perform configuration, control, and status reporting.

IP with multiple AXI4-Stream interfaces must account for IP core-specific synchronization rules between configuration and data channels (for example: a configuration packet must precede each data packet for block-based processing in some wireless IPs). See DSP and Wireless IP: AXI Feature Adoption in Chapter 3.

Table 6-2 summarizes DSP and Wireless IP AXI4-Stream signaling usage and guidelines.

Table 6-2: **DSP/Wireless IP Domain AXI4-Stream Signaling Usage**

Signal	Endpoint
ACLK	Used and Supported
ACLKEN	Limited Support
ARESTEN	Used and Supported
TVALID	Used and Supported
TREADY	Used and Supported
TDATA	Used and Supported
TID	Not Supported
TDEST	Not Supported
TKEEP	Not Supported
TSTRB	Not Supported
TUSER	Used and Supported
TLAST	Used and Supported

Communications IP

AXI4-Stream IP in the communications application domain refers to *Endpoint* IP that implement high-speed communications protocols using transceivers or I/Os. Depending on the relationship with transceivers or I/Os that cannot accept backpressure, these AXI4-Stream interfaces can have limited handshaking options (for example no support for the `TREADY` signal with some IP that are closely tied to the physical interface).

AXI4-Stream communications IP are tightly coupled to the underlying protocol (such as PCIe, Ethernet, and SRIO) with explicit data formats and handshaking rules that limit IP interoperability across protocols.

AXI4-Stream communications IP are usually connected to custom logic in a user design, AXI infrastructure IP, or other protocol-specific IP (for example: Ethernet IP using AXI Ethernet DMA or PCIe bridging IP to other protocols).

More details on Communications IP are available at:

http//www.xilinx.com/products/technology/connectivity/index.htm

Table 6-3 summarizes communications IP AXI4-Stream signaling usage and guidelines.

Table 6-3: **Communications IP Domain AXI4-Stream Signaling Usage**

Signal	Endpoint
ACLK	Used and Supported
ACLKEN	Not Supported
ARESTEN	Used and Supported
TVALID	Used and Supported
TREADY	Optionally Supported
TDATA	Used and Supported
TID	Not Supported
TDEST	Not Supported
TKEEP	Optionally Supported
TSTRB	Not Supported
TUSER	Optionally Supported
TLAST	Used and Supported

AXI Infrastructure IP

AXI4-Stream infrastructure IP refers to IP that generally exchanges or moves data within a system without using the contents of data or being tied to a specific data interpretation. Typically AXI4-Stream infrastructure IP are used as system building blocks or as test and debug IP. Common use models for infrastructure IP include width conversion, data switching and routing, buffering, pipelining, and DMA.

AXI infrastructure IP is required to support a wide range of optional and flexible signal interface configurations to meet the signaling needs of IP from all domains. This also helps to exchange data between mismatched AXI4-Stream master and slave signaling interfaces. More details on AXI4-Stream interconnect IP are available at:

http://www.xilinx.com/products/intellectual-property/axi4-stream_interconnect.htm

For information about DMA IP that implement AXI4-Stream to AXI4 (Memory Mapped) data transfer, see Xilinx AXI Infrastructure IP in Chapter 2.

Table 6-4 describes the main sub-categories of AXI4-Stream infrastructure IP and their useful characteristics. Table 6-5, page 137 lists the infrastructure IP domain AXI4-Stream signaling usage.

Table 6-4: **AXI4-Stream Infrastructure IP Sub-Categories**

Type	Key Characteristics	Examples	Interoperability Considerations
Pass-through	◦ Used for buffering, pipelining, or moving data. ◦ Does not change contents of data.	◦ Register Slice ◦ FIFO ◦ Clock Converter ◦ Crossbar Switch	◦ Does not change contents or organization of data. ◦ Generally compatible with all AXI4-Stream IP.
Modifier	◦ Potential to change contents or organization of data.	◦ Width converter ◦ Bus Rip/Concatenation ◦ (Split/Combine) ◦ MUX/DeMUX ◦ Subset Converter ◦ Packer	◦ Performs specific algorithmic operations on data. ◦ Compatible with most AXI4-Stream IP with proper usage.
Stream Endpoint	◦ Entry/Exit Point for a stream subsystem or processing pipeline. ◦ Usually the logical data source or terminus in a chain of IPs.	◦ DMA (general purpose) ◦ (MicroBlaze™ processor stream ports) ◦ AXI4-Lite to AXI4-Stream bridge ◦ Virtual FIFO Controller	◦ Usually the first or last IP in a processing pipeline ◦ Compatible with most AXI4-Stream IP with proper usage ◦ Might have limited support for `TUSER`, `TID`, and `TDEST`
Monitor	◦ Attaches to an AXI interface for observation only. ◦ Does not alter the contents of data.	◦ AXI Chipscope Monitor ◦ AXI HW Protocol Checker ◦ Performance Monitor	◦ Observes but does not alter data. ◦ Taps an AXI4-Stream connection for viewing. ◦ Generally compatible with all AXI4-Stream IP.

Table 6-5: **Infrastructure IP Domain AXI4-Stream Signaling Usage**

Signal	Pass- Through	Modifier	Endpoint	Monitor
ACLK	Used and Supported	Used and Supported	Used and Supported	Used and Supported
ACLKEN	Used and Supported	Used and Supported	Not Supported	Not Supported
ARESTEN	Used and Supported	Used and Supported	Used and Supported	Used and Supported
TVALID	Used and Supported	Used and Supported	Used and Supported	Used and Supported
TREADY	Used and Supported	Used and Supported	Used and Supported	Used and Supported
TDATA	Used and Supported	Used and Supported	Used and Supported	Used and Supported

Table 6-5: **Infrastructure IP Domain AXI4-Stream Signaling Usage *(Cont'd)***

Signal	Pass- Through	Modifier	Endpoint	Monitor
TID	Used and Supported	Used and Supported	Not Supported	Used and Supported
TDEST	Used and Supported	Used and Supported	Limited Support	Used and Supported
TKEEP	Used and Supported	Used and Supported	Limited Support	Used and Supported
TSTRB	Used and Supported	Used and Supported	Limited Support	Used and Supported
TUSER	Used and Supported	Used and Supported	Limited Support	Used and Supported
TLAST	Used and Supported	Used and Supported	Used and Supported	Used and Supported

Domain-Specific Data Interpretation and Interoperability Guidelines

Domain-specific protocols can be layered on top of the AXI4-Stream signaling layer so that IP can interpret and use the data that has been exchanged. This section summarizes key domain-specific layered protocol usage information and presents guidelines to help users focus on key concepts when constructing IP and systems to be interoperable.

Video IP Layered Protocols

Video IP use layered protocols to represent the video format and resolution. Video IP must be configured to use the same video format and resolution to transfer information, such as industry recognized YUV/YUVA, RGB/RGBA video formats and 1920x1080P60 resolutions. Where necessary, format conversion IP such as color space converters can be used to convert the video between IP blocks in a system.

Video IP also has common conventions for packing the data bits for the color components (such as red, green, and blue components) into `TDATA`. AXI4-Stream signals such as `TLAST` and `TUSER` encode line and frame boundaries for a given video resolution.

Video IP also contain optional AXI4-Lite interface that can change the layered protocol during runtime, typically under microprocessor control.

See Video IP: AXI Feature Adoption in Chapter 3 for details on the encoding of video layered protocols. Table 6-6 and Table 6-7 summarize some of the key characteristics, interoperability considerations, and guidelines for layered protocols used in the Video IP domain.

Table 6-6: Video IP Layered Protocol Summary

Type	Key Characteristics	Examples	Interoperability Considerations
Video Format	• Video IP support industry standard formats	• RGB • YUV 4:2.2	• Use conversion IP video formats.
Pixels Encoding (Components)	• Pixels (TDATA beats) consist of 1 to 4 components. • Each component is 8, 10, 12, or 16 bits wide. • Components are concatenated and 0-padded up to an overall byte width.	• 24-bit wide TDATA carrying RGB (3x8-bit components). • 6-bit wide TDATA carrying YUV 4:2:2 (8-bit alternating V/U +8-bit Y components).	• The relative placement order of the components in the TDATAbeat is fixed. • When the width of components mismatch, rules apply on how to scale the data.
Video Resolution / Framing	• AXI4-Stream `TLAST/TUSER` signaling is used to mark end-of-line and frame boundaries • Only active video pixel data is transferred	• `TUSER[0]` marks the start of a frame • `TLAST` marks end of line. • Examples: • 1024x768 would have 1024 `TDATA` beats per `TLAST`. • 768 `TLAST` beats per `TUSER[0]`.	• `TLAST` and `TUSER` must be preserved and placed at correct intervals relative to pixels. • Some video IP are capable of recovering from corrupted or incomplete frame data and relocking to the framing signals. • Connected Video IP must have the same frame resolution or rescaling IP. is required.

Table 6-7: **Video IP Interoperability, Considerations, and Guidelines**

Guideline	Description	Rule to Achieve Interoperability	User Effort/Notes
Data Type	1-4 components per pixel, 8,10,12,16 bit per component	Converter cores provided. All cores configurable to support 8,10,12, 16 bit data.	Seamless if video format and resolution match; Standard adapters provided to change formats and resolution.
Data Burst	Video standards with up to 8k pixels per line (burst) supported	All cores configurable to support standard burst sizes.	Ensure that connected IP use the same settings.

Table 6-7: **Video IP Interoperability, Considerations, and Guidelines *(Cont'd)***

Guideline	Description	Rule to Achieve Interoperability	User Effort/Notes
AXI4-Stream Optional Signals	Optional: `ACLKEN`, `ARESETN`, Else Fixed set (`TUSER[0]`, `TLAST`, `TREADY`, `TVALID`, `TDATA`)	AXI-FIFO needed to bridge between different ACLK or ACLKEN domains.	Standard adapter might be needed (AXI4-Stream FIFO or AXI4-Stream Interconnect)
AXI4-Stream TUSER Signals	Only TUSER0signal used consistently across all video cores	TUSER0 is required and is used to signal frame boundaries.	Special considerations might be needed for IP that can generate or recover from partial frames (for example, handling when cable is removed and reconnected)
Number of Channels and AXI4-Lite Dependency	Generally single AXI4-Stream through IP, but some cores have multiple inputs/output streams; Most IP have an optional AXI4-Lite control interface.	For cores with multiple input/output streams or when AXI4-Lite is used, read the datasheet to understand data relationships.	Core can permit format/resolution to be changed using AXI4-Lite requiring care to coordinate any runtime changes across system.

DSP/Wireless IP Layered Protocols

DSP/Wireless IP use layered protocols to represent numerical information and structures to perform processing such as filtering, arithmetic operations. DSP/ Wireless IP usually have a flow through architecture with input and output stream interfaces to take in data, perform operation on the data, and send out the data.

Data flow is usually optional between blocking (with backpressure using TREADY) and non-blocking (continuous data flow without TREADY).

DSP/Wireless IP also support data organized into TDM or parallel paths to operate on numeric data structures such as arrays. The streams can also carry optional sideband status signals to supplement the numeric data with core-specific information. DSP/Wireless IP also often contain control AXI4-Stream interfaces for optional runtime control and status such as the ability to change filter coefficients at runtime using a secondary AXI4-Stream interface.

See AXI Feature Adoption in Xilinx FPGAs in Chapter 3 for details of encoding DSP/wireless layered protocols. Table 6-8, page 114 and Table 6-9, page 115 summarizes some of the key characteristics, interoperability considerations, and guidelines for layered protocols used in the DSP/Wireless IP domain.

Table 6-8: **DSP/Wireless IP Layered Protocol Summary**

Key Characteristi	Description	Examples	Interoperability Considerations
Number of Data Transfers per Invocation	• Sample-Based Processing: IP data processing is applied independently to every single AXI4-Stream transfer. • Block-Based Processing: IP data processing is applied to a "block" "packet" of AXI4-Stream transfers.	• Sample: complex multiplier operates on a data sample at a time. • Block: FFT of a given point size.	Block based IP must have same notion of block size to interoperate.
Number of Data Transfers per Invocation	• Single-Channel: 1 logical stream of data. • Multi-Channel: data being processed in parallel (could be a single AXI4-Stream interface with parallel data concatenated together on TDATA).	• FIR compiler can be configured to operate single or multiple parallel data lanes. Multi-channel mode allows DSP resources to be shared across multiple data paths.	Data must be concatenated together or split out to change number of parallel data streams.

Table 6-8: **DSP/Wireless IP Layered Protocol Summary *(Cont'd)***

Key Characteristi	Description	Examples	Interoperability Considerations
Data type representation of TDATA	• Unit: single datum • Array: multiple data • Structure: tuples or special data structures for control/status interfaces	• FIR compiler can operate on unit data or array of data when configured for single or multiple data paths, respectively. • Control/status AXI4-Stream interfaces usually require structured data such as FFT configuration	• IP must have same notion of data type representation to interoperate. • Structured TDATA is often used in control/status interfaces and need custom logic or programmable IP (like a microprocessor) to generate.
Use of TUSER signal for sideband data	None: No TUSER signal used Pass-through: TUSER signal is passed from an input interface to an output interface IP Specific: TUSER conveys IP-specific sideband data.	• Complex Multiplier can work without TUSER or can Pass-through TUSER. • DDS compiler can include TUSER in its output interface with IP-specific TDM channel markers.	• IP-specific TUSER often requires custom logic to decode. • TUSER Pass-through mode is useful for transmitting user information through a core to match latency with the data.
Use of TLAST signal	None: No TLAST signal used. Pass-thru: TLAST signal is passed from an input interface to an output interface. Block end marker: TLAST indicates the last transfer in a block. IP-specific TLAST used to mark an IP specific location in the data transfers	• Divider Generator can work without TLAST or can Pass-through TLAST. • FFT uses TLAST for block end marker	• IP must have same notion of TLAST when used as block end marker. • TLAST Pass-through mode is useful for transmitting user information through a core with latency matched to that of the data. • IP specific TLAST often requires custom logic to decode TLAST.

Table 6-9: **DSP/Wireless Interoperability Guidelines**

Guideline	Description	Rule to Achieve Interoperability	User Effort/Notes
Data Type	Scalars, arrays, and structures with data type elements for fixed and floating point number representation.	Adhere to defined data types and conventions. For example real versus integer, common binary point and data size.	Must understand data types/structures and ensure consistency; adapters might be needed
Data Burst	Sample or block processing; IP processing applied to a single transfer or a block of transfers	Use of (optional) TLAST to delimit blocks, packets, or frames.	Must align block sizes to match data structure size, adapters might be needed.

Table 6-9: **DSP/Wireless Interoperability Guidelines**

Guideline	Description	Rule to Achieve Interoperability	User Effort/Notes
AXI4-Stream Optional Signals	Optional: ACLKEN, TREADY TLAST, TUSER. Fixed: TDATA, TVALID	Adhere to DSP/Wireless IP specific guidelines in DSP and Wireless IP: AXI Feature Adoption in Chapter 3.	Optional signals must be used consistently or adapters are needed.
AXI4-Stream TUSER Signals	No use, Pass-through, or IP specific use of TUSER. TUSER is generally optional.	For higher interoperability, avoid use of IP specific TUSER.	TUSER signals must be used consistently. Custom logic might be needed to handle IP-specific TUSER.
Data Type	Scalars, arrays, and structures with data type elements for fixed and floating point number representation.	Adhere to defined data types and conventions. For example real versus integer, common binary point and data size.	Must understand data types/structures and ensure consistency; adapters might be needed

Communications IP Layered Protocols

IP in this domain use layered protocols to represent communications protocols, typically networking packets. Packets can be fixed or variable sized depending on the protocol (such as Ethernet and PCIe). IP are usually closely tied to the physical layer interface or logical level interface with transmit and receive AXI4-Stream interfaces, sideband TUSER signals for control/status, and some offer additional AXI4-Lite and AXI4 memory mapped interfaces.

Because of the specific relationship to a communications protocol standard, IP in this domain are often used with custom logic, infrastructure IP, other IP of the same protocol type.

Table 6-10 summarizes some of the key guidelines for layered protocols used in the Communicatons IP domain.

Table 6-10: **Communications IP Layered Protocol Interoperability Guidelines**

Guideline	Description	Rule to Achieve	User Effort/Notes
Data Type	Packetized data, with and without headers/footers. Matched to protocols like Ethernet, PCIe, or SRIO.	Remove header and footer to access raw packet data or transfer data to memory mapped space (for example) using a DMA engine.	Must understand data types and ensure packet data is delivered in the correct order. Adapters might be needed.
Data Burst	Variable size: Minimum can be a single cycle of data. Maximum depends upon the parent protocol	All cores configurable to support standard burst sizes, up to a defined limit for a given protocol	Care must be taken to ensure that legal-sized packets are transferred between cores. Adapters might be needed to break apart too-large packets.

Table 6-10: **Communications IP Layered Protocol Interoperability Guidelines *(Cont'd)***

Guideline	Description	Rule to Achieve	User Effort/Notes
AXI4-Stream Optional Signals	TREADY, TKEEP, TDATA, TUSER, TLAST. In some cases, ACLKEN, TDEST, TID.	Use adapters for infrastructure IP. TDEST/TID can be used for data interleaving.	Adapters might be required.
AXI4-Stream TUSER Signals	Variety of uses and sizes. Common uses: packet discontinue, framing signals, packet details	Avoid using TUSER. Set control/status information in AXI4-Lite register space if possible	Migrate TUSER signals to dedicated AXI4-Lite or AXI4-Stream sideband bus. Adapter might be required.
Number of Channels and AXI4-Lite Dependency	Generally single AXI4-Stream in each direction, through IP. SRIO can have up to 8 streams in each direction.	For cores with multiple input/output streams, read the datasheet to understand data relationships.	Cores must have an appropriate port to which to connect. Refer to the individual datasheet for each core.

AXI Infrastructure IP Layered Protocols

IP in this domain often do not use specific layered protocols but are configurable to Pass-through data or to generate/receive data using a processor or DMA engine.

The key elements for interoperability is to use the AXI4-Stream protocol following the recommendations in Signaling Protocol in Chapter 3. AXI Infrastructure IP is designed to be broadly compatible with IP from different domains because it has highly configurable interfaces, and generally does not use the contents of the data.

Table 6-10 summarizes some of the key characteristics and interoperability considerations for the AXI Infrastructure IP domain.

Table 6-11: **AXI Infrastructure IP Interoperability Guidelines**

Guideline	Description	Rule to Achieve	User Effort/Notes
AXI4-Stream Optional Signals Use	• Endpoint type IP generally limit support for `TID`, `TDEST` (unless multi-channel IP), `TSTRB`, `TUSER`. • Endpoint IP using `TLAST` should be aware of burst size. • `TKEEP` only for packet remainders.	• Use "Continuous" or "Continuous Aligned Streams".	• Low for Pass-through and Monitor IP types. • Generally low when using core signals `TDATA, TVALID, TREADY, TLAST, TKEEP` (remainders). • Medium to high with more complex systems using `TID, TDEST, TSTRB, or TUSER.`
Layered Protocols	• Pass-through and Monitor IP types do not alter data. • Modifiers can transform data. • Endpoints can synthesize or receive layered protocols with proper configuration.	• Minimize use of `TID, TDEST, TSTRB`, and `TUSER`. • Consider how modifiers change the data structures used by layered protocols. • Endpoints require the user to properly program them to generate data contents matching the layered protocol.	• Low for Pass-through and Monitor IP types. • Medium when using modifiers that can alter the data encodings algorithmically. • Medium to High for endpoint IPs that must be configured and programmed properly to match the requirements of layered protocols.
Other Interoperability Factors	• There are Interdependencies, such as Endpoint IP often have additional AXI4-Lite control interfaces.	Pay attention to real time system impact when using infrastructure IP, such as FIFOs might increase latency.	• Low for Monitor types. • Low to Medium for Pass-through and Modifier types that could affect real time behavior. • Medium to high for endpoint types which could have control ports.
Interfacing to IP in other Domains	• Pass-through and Monitor types designed to work with all domains. • Modifier IP can work in video or DSP domains, but need users to validate data structure integrity. • Endpoint IP have limited ability to interface to other domains and might need domain specific endpoint IP (example AXI Video DMA, PCIe DMA, and so forth).	• Care must be taken to configure and program Endpoints to match the layered protocol requirements of other IP domains.	• Low for Pass-through and Monitor IP types. • Medium when using modifiers that can alter data structures. Medium to High for using endpoints that have to synthesize or receive layered protocols.

Appendix A

AXI Adoption Summary

Introduction

This appendix provides a summary of protocol signals adopted by Xilinx® in the AXI4 and AXI-Lite, and AXI4-Stream interface protocol IP. Consult the AXI specifications (available at www.amba.com) for complete descriptions of each of these signals.

Global Signals

Table A-1 lists the Global AXI signals.

TableA-1: **Global AXI Signals**

Signal	AXI4	AXI4-Lite
ACLK	Clock source.	
ARESETN	Global reset source, active-Low. This signal is not present on the interface when a reset source (of either polarity) is taken from another signal available to the IP. Xilinx IP generally must deassert VALID and READY outputs within 8 cycles of reset assertion, and generally require a reset pulse-width of 16 or more clock cycles of the slowest clock. Some Xilinx IP might document that they can accept ARESETN asserted for fewer than 16 cycles. For example, DSP IP require ARESETN asserted for a minimum of 2 cycles on their AXI4-Stream interfaces.	

AXI4 and AXI4-Lite Signals

AXI4 and AXI4-Lite Write Address Channel Signals

Table A-2 lists the Write Address Channel signals.

Note: A read-only master or slave interface omits the entire write address channel.

TableA-2: **Write Address Channel Signals**

Signal	AXI4	AXI4-Lite
AWID	Fully supported. Masters need only output the set of ID bits that it varies (if any) to indicate re-orderable transaction threads. Single-threaded master interfaces can omit this signal. Masters do not need to output the constant portion that comprises the Master ID, as this is appended by the AXI Interconnect.	Signal not present.
AWADDR	Fully supported. Width 32 bits, or larger as needed. High-order bits outside the native address range of a slave are ignored (trimmed), by an endpoint slave, which could result in address aliasing within the slave. ***Note:*** EDK supports 32-bit address only.	
AWLEN	Fully supported. Support bursts: • Up to 256 beats for incrementing (`INCR`). • 16 beats for `WRAP`.	Signal not present.
AWSIZE	Transfer width 8 to 1024 bits supported. Use of narrow bursts where `AWSIZE` is less than the native data width is not recommended.	Signal not present.
AWBURST	`INCR` and `WRAP` fully supported. `FIXED` bursts are not recommended. `FIXED` bursts result in protocol compliant handshakes, but effect of `FIXED` transfer can be aliased to `INCR` or undefined.	Signal not present.
AWLOCK	Exclusive access support not implemented in endpoint Xilinx IP. Infrastructure IP will pass exclusive access bit across a system.	Signal not present.
AWCACHE	0011 value recommended. Xilinx IP generally ignores (as slaves) or generates (as masters) transactions as Normal, Non-cacheable, Modifiable, and Bufferable. Infrastructure IP will pass Cache bits across a system.	
AWPROT	000 value recommended. Xilinx IP generally ignores (as slaves) or generates transactions (as masters) with Normal, Secure, and Data attributes. Infrastructure IP passes Protection bits across a system.	
AWQOS	Not implemented in Xilinx Endpoint IP. Infrastructure IP passes QoS bit across a system.	Signal not present.

TableA-2: **Write Address Channel Signals *(Cont'd)***

Signal	AXI4	AXI4-Lite
AWREGION	Can be implemented in Xilinx Endpoint slave IP. Not present on master IP. Generated by AXI Interconnect using corresponding address decoder range settings.	Signal not present.
AWUSER	Generally, not implemented in Xilinx endpoint IP. Infrastructure IP passes USER bits across a system.	Signal not present.
AWVALID	Fully supported.	
AWREADY	Fully supported.	

AXI4 and AXI4-Lite Write Data Channel Signals

Table A-3 lists the Write Data Channel signals.

Note: A read-only master or slave interface omits the entire Write Data Channel.

TableA-3: **Write Data Channel Signals**

Signal	AXI4	AXI4-Lite
WDATA	Native width 32 to 1024 bits supported.	32-bit width supported. 64-bit AXI4-Lite native data width is not currently supported
WSTRB	Fully supported.	Slaves interface can elect to ignore `WSTRB` (assume all bytes valid).
WLAST	Fully supported.	Signal not present.
WUSER	Generally, not implemented in Xilinx endpoint IP. Infrastructure IP will pass USER bits across a system.	Signal not present.
WVALID	Fully supported.	
WREADY	Fully supported.	

AXI4 and AXI4-Lite Write Response Channel Signals

Table A-4 lists the Write Response Channel signals.

Note: A read-only master or slave interface omits the entire write response channel.

Table A-4: **Write Response Channel Signals**

Signal	AXI4	AXI4-Lite
BID	Fully supported. See AWID for more information.	Signal not present.
BRESP	Fully supported.	`EXOKAY` value not supported by specification.

Table A-4: **Write Response Channel Signals *(Cont'd)***

BUSER	Generally, not implemented in Xilinx endpoint IP. Infrastructure IP will pass USER bits across a system.	Signal not present.
BVALID	Fully supported.	
BREADY	Fully supported.	

AXI4 and AXI4-Lite Read Address Channel Signals

Table A-5 lists the Read Address Channel signals.

Note: A write-only master or slave interface omits the entire read address channel.

Table A-5: **Read Address Channel Signals**

Signal	AXI4	AXI4-Lite
ARID	Fully supported. Masters need only output the set of ID bits that it varies (if any) to indicate re-orderable transaction threads. Single-threaded master interfaces can omit this signal. Masters do not need to output the constant portion that comprises the "Master ID", as this is appended by the AXI Interconnect.	Signal not present.
ARADDR	Fully supported. Width 32 bits, or larger as needed. High-order bits outside the native address range of a slave are ignored (trimmed) by an endpoint slave, which could result in address aliasing within the slave. ***Note:*** EDK supports 32-bit address only.	
ARLEN	`INCR` and `WRAP` fully supported. `FIXED` bursts are not recommended. `FIXED` bursts result in protocol compliant handshakes, but effect of `FIXED` transfer can be aliased to `INCR` or undefined.	Signal not present.
ARSIZE	Transfer width 8 to 1024 bits supported. Use of narrow bursts where `ARSIZE` is less than the native data width is not recommended.	Signal not present.
ARBURST	`INCR` and `WRAP` fully supported. `FIXED` bursts are not recommended. `FIXED` bursts result in protocol compliant handshakes, but effect of `FIXED` transfer can be aliased to `INCR` or undefined.	Signal not present.
ARLOCK	Exclusive access support not implemented in Endpoint Xilinx IP. Infrastructure IP passes exclusive access bit across a system.	Signal not present.
ARCACHE	0011 value recommended. Xilinx IP generally ignores (as slaves) or generates (as masters) transactions with Normal, Non-cacheable, Modifiable, and Bufferable. Infrastructure IP will pass Cache bits across a system.	

Table A-5: **Read Address Channel Signals *(Cont'd)***

Signal	AXI4	AXI4-Lite
ARPROT	000 value recommended. Xilinx IP generally ignore (as slaves) or generate transactions (as masters) with Normal, Secure, and Data attributes. Infrastructure IP passes Protection bits across a system.	
ARQOS	Not implemented in Xilinx Endpoint IP. Infrastructure IP passes QoS bit across a system.	Signal not present.
ARREGION	Can be implemented in Xilinx Endpoint Slave IP. Not present on master IP. Generated by AXI Interconnect using corresponding address decoder range settings.	Signal not present.
ARUSER	Generally, not implemented in Xilinx Endpoint IP. Infrastructure IP passes User bits across a system.	Signal not present.
ARVALID	Fully supported.	
ARREADY	Fully supported.	

AXI4 and AXI4-Lite Read Data Channel Signals

Table A-6 lists the Read Data Channel signals.

Note: A read-only Master or slave interface omits the entire read data channel.

Table A-6: **Read Data Channel Signals**

Signal	AXI4	AXI4-Lite
RID	Fully supported. See ARID for more information.	Signal not present.
RDATA	Native width 32 to 1024 bits supported.	32-bit width supported. 64-bit AXI4-Lite native data width is not supported.
RRESP	Fully supported.	`EXOKAY` value not supported by specification.
RLAST	Fully supported.	Signal not present.
RUSER	Generally, not implemented in Xilinx Endpoint IP. Infrastructure IP will pass User bits across a system.	Signal not present.
RVALID	Fully supported.	
RREADY	Fully supported.	

AXI4-Stream Signal Summary

Table A-7 lists the AXI4-Stream signal summary.

TableA-7: **AXI4-Stream Signal Summary**

Signal	Optional	Default (All Bits)	Description
TVALID	No	N/A	No change.
TREADY	Yes	1	No change
TDATA	Yes	0	No change. Xilinx AXI IP convention: 8 through 4096 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TSTRB	Yes	Same as TKEEP else 1	No change. Generally, the usage of `TSTRB` is to encode Sparse Streams. TSTRB should not be used only to encode packet remainders.
TKEEP	Yes	1	In Xilinx IP, there is only a limited use of Null Bytes to encode the remainders bytes at the end of packetized streams. TKEEP is not used in Xilinx endpoint IP for leading or intermediate null bytes in the middle of a stream.
TLAST	Yes	0	Indicates the last data beat of a packet. Omission of `TLAST` implies a continuous, non-packetized stream.
TID	Yes	0	No change. Xilinx AXI IP convention: Only 1-32 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TDEST	Yes	0	No change Xilinx AXI IP convention: Only 1-32 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TUSER	Yes	0	No change Xilinx AXI IP convention: Only 1-4096 bit widths are used by Xilinx AXI IP (establishes a testing limit).

AXI Terminology

Table B 1 lists the AXI terminology.

Table B-1: **AXI Terminology**

Term	Type	Description	Usage
AXI	Generic	The generic term for all implemented AXI protocol interfaces.	General description.
AXI4	Memory mapped block transfers	Addressed interface bursts up to 256 data beats.	Embedded and memory cores. Examples: MIG, block Ram, EDK PCIe Bridge, FIFO.
AXI4-Lite	Control Register Subset	32-bit data, memory mapped, lightweight, single data beat transfers only.	Management registers. Examples: Interrupt Controller, UART Lite, IIC Bus Interface.
AXI4-Stream	Streaming Data Subset	Unidirectional links modeled after a single write channel. Unlimited burst length.	Used in DSP, Video, and communication applications.
Interface	AXI4 AXI4-Lite AXI4-Stream	Collection of one or more channels that expose an IP core function, connecting a master to a slave. Each IP can have multiple interfaces.	All.
Channel	AXI4 AXI4-Lite AXI4-Stream	Independent collection of AXI signals associated with a VALID signal	All.
Bus	Generic	Multiple-bit signal (Not an **interface** or a **channel**).	All.
Transaction	AXI4-Stream	Complete communication operation across a **channel**, composed of one or more **transfers**. A complete action.	Used in DSP, Video, and communication applications.
	AXI4 AXI4-Lite	Complete collection of related read or write communication operations across address, data, and response channels, composed of one or more **transfers**. A complete read or write request.	Embedded and memory cores. Management registers.

Table B-1: **AXI Terminology *(Cont'd)***

Term	Type	Description	Usage
Transfer	AXI4 AXI4-Lite AXI4-Stream	Single clock cycle where information is communicated, qualified by a VALID hand-shake. Data beat	All.
Burst	AXI4 AXI4-Lite AXI4-Stream	Transaction that consists of more than one **transfer**.	All.
master	AXI4 AXI4-Lite AXI4-Stream	An IP or device (or one of multiple interfaces on an IP) that generates AXI transactions out from the IP onto the wires connecting to a slave IP.	All.
slave	AXI4 AXI4-Lite AXI4-Stream	An IP or device (or one of multiple interfaces on an IP) that receives and responds to AXI transactions coming in to the IP from the wires connecting to a master IP.	All.
master interface (generic)	AXI4 AXI4-Lite AXI4-Stream	An interface of an IP or module that generates out-bound AXI transactions and thus is the initiator (source) of an AXI transfer.	All.
slave interface (generic)	AXI4 AXI4-Lite AXI4-Stream	An interface of an IP or module that receives in-bound AXI transactions and becomes the target (destination) of an AXI transfer.	All.
SI	AXI4 AXI4-Lite	AXI Interconnect Slave Interface: • For XPS flow, Vectored AXI slave interface receiving in-bound AXI transactions from all connected master devices. • For CORE Generator tool flow, one of multiple slave interfaces connecting to one master device.	EDK.
MI	AXI4 AXI4-Lite	AXI Interconnect Master Interface: • For XPS flow, Vectored AXI master interface generating out-bound AXI transactions to all connected slave devices. • For CORE Generator tool flow, one master interface connecting to one slave device.	EDK.
SI slot	AXI4 AXI4-Lite	Slave Interface Slot: A slice of the slave Interface vector signals of the Interconnect that connect to a single master IP.	EDK.
MI slot	AXI4 AXI4-Lite	Master Interface Slot: A slice of the Master Interface vector signals of the Interconnect that connect to a single Master Interface slave IP.	EDK.
SI-side	AXI4 AXI4-Lite	Refers to a module interface closer to the SI side of the Interconnect.	All.
MI-side	AXI4 AXI4-Lite	Refers to a module interface closer to the MI side of the Interconnect.	All.

Table B-1: **AXI Terminology *(Cont'd)***

Term	Type	Description	Usage
upsizer	AXI4 AXI4-Lite AXI4-Stream	Data width conversion function in which the data path width gets wider when moving in the direction from the slave interface toward the master interface (regardless of write/read direction).	All.
downsizer	AXI4 AXI4-Lite AXI4-Stream	Data width conversion function in which the data path width gets narrower when moving in the direction from the slave interface toward the master interface (regardless of write/read direction).	All.
SAMD	Topology	Shared-Address, Multiple-Data: Configuration of AXI Interconnect where data transfers can occur independently and concurrently between different master and slave devices.	All.
SASD	Topology	Shared-Address, Shared-Data: Configuration of AXI Interconnect where a single read and write pathway is implemented.	All.
Shared-Access	Topology	Configuration of AXI Interconnect based on SASD topology where only one transaction is issued at a time to minimize resources.	EDK.
Crossbar	Topology	Configuration of AXI Interconnect based on SAMD topology where data pathways are implemented according to sparse connectivity between master and slave devices.	All.
Crossbar	Structural	Module at the center of the AXI Interconnect that routes address, data and response channel transfers between various SI slots and MI slots.	All.

XILINX®

Appendix C

Additional Resources

Third Party Documentation

Additional reference documentation:

1. *ARM AMBA AXI Protocol v2.0 Specification*
2. *AMBA4 AXI4-Stream Protocol v1.0*

See the Introduction, page 3 for instructions on how to download the ARM® AMBA® AXI specification from http://www.amba.com.

3. *International Telecommunications Union (ITU)*: *ITU-R BT.1614:* http://engineers.ihs.com/document/abstract/SUCFEBAAAAAAAAAA
4. *HDTV Standards and Practices for Digital Broadcasting*: *SMTPE 352M-2009*

Xilinx Documentation

IP Documentation

Additionally, this document references documents located at the following Xilinx website: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm

5. *Multi-port Memory Controller (MPMC) Data Sheet (DS643)*
6. *AXI Interconnect IP (DS768)*
7. *AXI-To-AXI Connector IP Data Sheet* (DS803)
8. *AXI External Master Connector* (DS804)
9. *AXI External Slave Connector* (DS805)
10. *AXI Bus Functional Models User Guide (UG783)*
11. *AXI Bus Functional Model Data Sheet (DS824)*
12. AXI Data Mover Product Guide (PS022)
13. *LogiCORE IP FIFO Generator, (DS317)*
14. *AXI4-Stream Interconnect Product Guide* (PG035)
15. *AXI Virtual FIFO Controller* (PG038)

Other Xilinx Documentation

16. *White Paper: Maximize System Performance Using Xilinx Based AXI4 Interconnects (WP417)*
17. *Bridging Xilinx Streaming Video Interface with AXI4-Stream Protocol* (XAPP521):
18. *AXI Multi-Ported Memory Controller Application Note* (XAPP739):
19. *Designing High-Performance Video Systems with the AXI Interconnect* (XAPP740):
20. *AXI Bus Functional Models User Guide* (UG783):
21. *MicroBlaze Processor Reference Guide* (UG081):
22. *Xilinx Design Tools: Installation and Licensing Guide (UG798)*:
23. *Xilinx Design Tools: Release Notes Guide* (UG631):
24. Video Demonstrations: http://www.xilinx.com/design
25. Xilinx Answer Database: http://www.xilinx.com/support/mysupport.htm
26. Xilinx Glossary: http://www.xilinx.com/support/documentation/sw_manuals/glossary
27. EDK Website: http://www.xilinx.com/tools/embedded.htm
28. CORE Generator® Tool: http://www.xilinx.com/tools/coregen.htm
29. Memory Control: http://www.xilinx.com/products/technology/memory-solutions/index.htm
30. System Generator: http://www.xilinx.com/tools/sysgen.htm
31. Local-Link: http://www.xilinx.com/products/design_resources/conn_central/locallink_member/sp06.pdf
32. Targeted Designs: http://www.xilinx.com/products/targeted_design_platforms.htm
33. Answer Record: http://www.xilinx.com/support/answers/37425.htm