AXI Reference Guide

UG761 (v13.1) March 7, 2011

XILINX®

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
09/21/2010	1.0	Initial Xilinx release in 12.4.
03/01/2011	2.0	Second Xilinx release in 13.1. Added new AXI Interconnect features. Corrected ARESETN description in Appendix A.
03/07/2011	3.0	Corrected broken link.

Table of Contents

Chapter 3: AXI Feature Adoption in Xilinx FPGAs

Chapter 4: Migrating to Xilinx AXI Protocols

Appendix A: AXI Adoption Summary

Appendix B: AXI Terminology

XILINX®

Chapter 1

Introducing AXI for Xilinx System Development

Introduction

Xilinx® has adopted the Advanced eXtensible Interface (AXI) protocol for Intellectual Property (IP) cores beginning with the Spartan®-6 and Virtex®-6 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*, along with the *AMBA4 AXI4-Stream Protocol v1.0*.

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 5, 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 19.

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, page 7 shows how an AXI4 Read transaction uses the Read address and Read data channels:

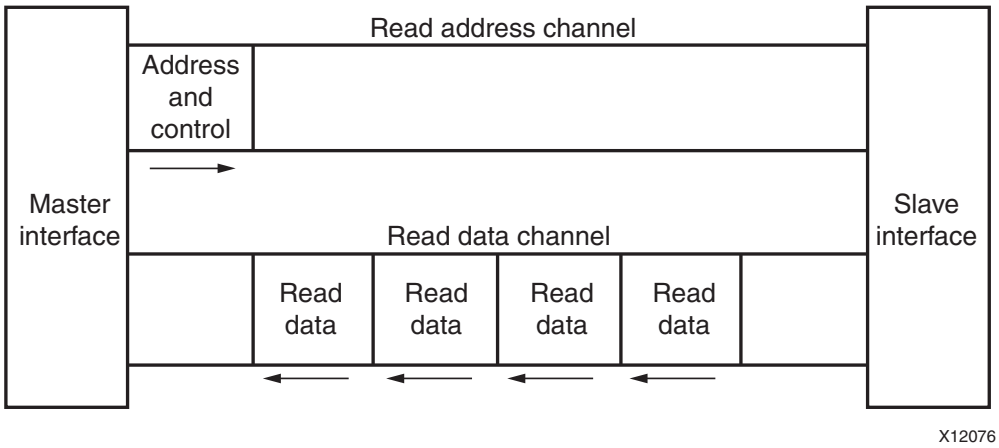

Figure 1-1: **Channel Architecture of Reads**

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

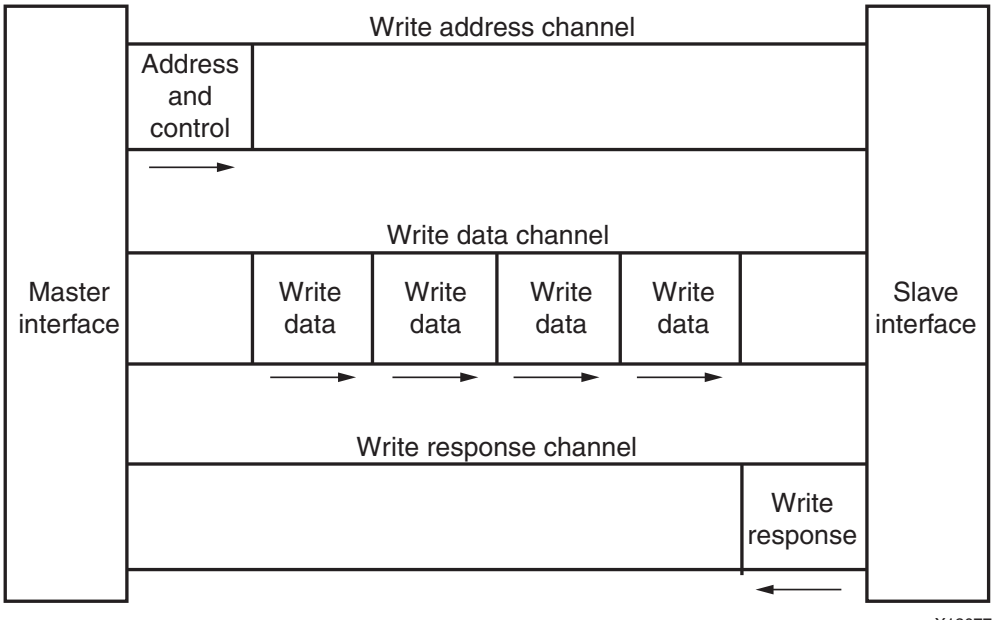

Figure 1-2: **Channel Architecture of Writes**

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 *AXI4-Stream Protocol Specification*. The specification describes how AXI4-Stream-compliant interfaces can be split, merged, interleaved, upsized, and downsized. Unlike AXI4, AXI4-Stream transfers cannot be reordered.

With regards to AXI4-Stream, it should be noted that even if two pieces of IP are designed in accordance with the AXI4-Stream specification, and are compatible at a signaling level, it does not guarantee that two components will function correctly together due to higher level system considerations. Refer to the AXI IP specifications at http://www.xilinx.com/ipcenter/axi4.htm, and AXI4-Stream Signals, page 45 for more information.

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.

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.

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.

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.

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

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.

Additional References

Additional reference documentation:

- *ARM AMBA AXI Protocol v2.0 Specification*
- *AMBA4 AXI4-Stream Protocol v1.0*

See the Introduction, page 5 for instructions on how to download the ARM® AMBA® AXI specification from http://www.amba.com.

Additionally, this document references the following documents, located at the following Xilinx website:
http://www.xilinx.com/support/documentation/axi_ip_documentation.htm.

- *AXII Interconnect IP (DS768)*
- *AXI-To-AXI Connector IP Data Sheet (DS803)*
- *AXI External Master Connector (DS804)*
- *AXI External Slave Connector (DS805)*
- *MicroBlaze Processor Reference Guide (UG081)*

This document lists the following Xilinx websites:

- AXI IP document website: http://www.xilinx.com/ipcenter/axi4.htm
- EDK website: http://www.xilinx.com/tools/embedded.htm
- CORE Generator® tool: http://www.xilinx.com/tools/coregen.htm
- Memory Control: http://www.xilinx.com/products/design_resources/mem_corner
- System Generator: http://www.xilinx.com/tools/sysgen.htm
- Local-Link: http://www.xilinx.com/products/design_resources/conn_central/locallink_member/sp06.pdf
- Targeted Designs: http://www.xilinx.com/products/targeted_design_platforms.htm
- Answer Record: http://www.xilinx.com/support/answers/37425.htm

XILINX®

Chapter 2

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 ISE. Xilinx recommends using the BSB to start new designs. Refer to 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 software 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.) Refer to SDK Help for more information.

More 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 Xilinx Answer Record 37856 for more information.

Using System Generator: DSP Edition

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

- AXI4 interface is supported in conjunction with the EDK Processor Block.
- AXI4-Stream interface is supported in IPs found in 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.

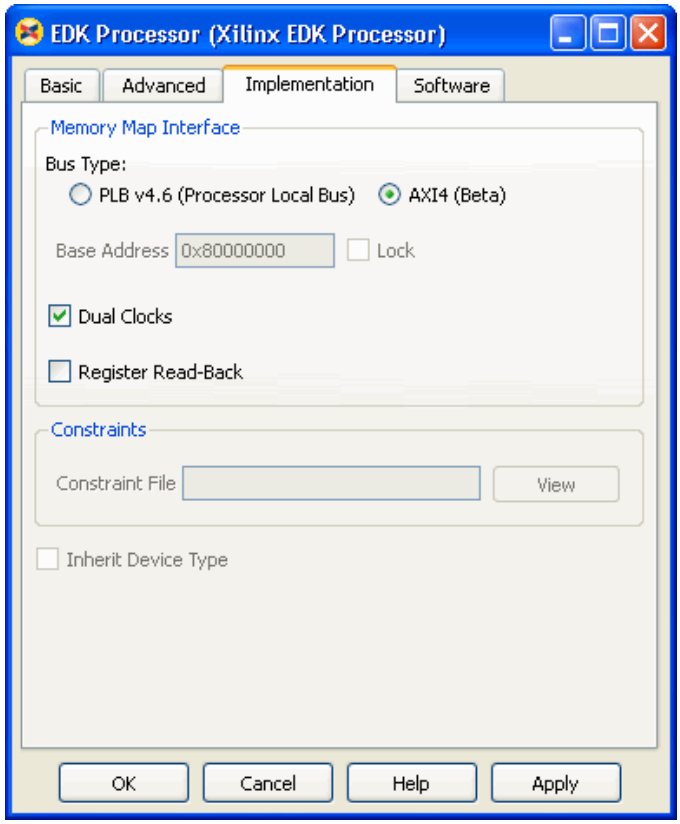

Figure 2-1: **EDK Processor Interface Implementation Tab**

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, is such an example.

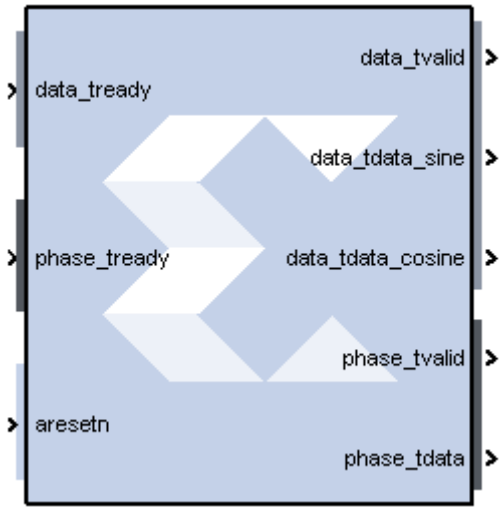

Figure 2-2: **Block Signal Groupings**

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.

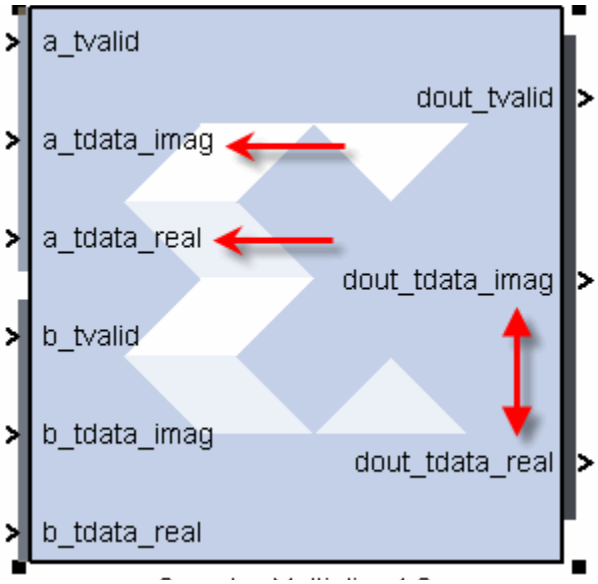

Figure 2-3: **Multi-Channel TDATA**

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.

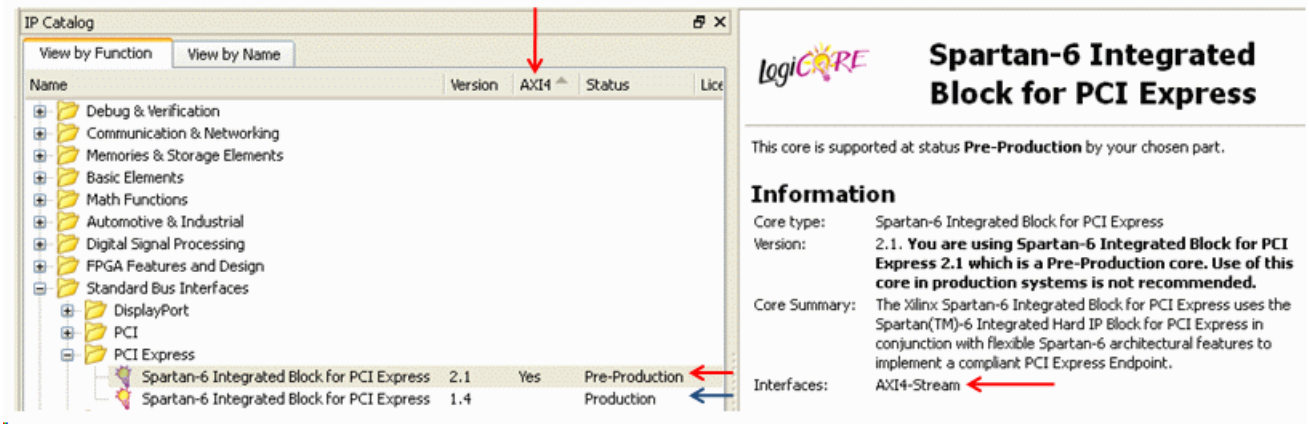

Figure 2-4: **IP Catalog in Xilinx Software**

Figure 2-5, page 18 shows the IP catalog in PlanAhead with the equivalent AXI4 column and the supported AXI4 interfaces in the IP details panel.

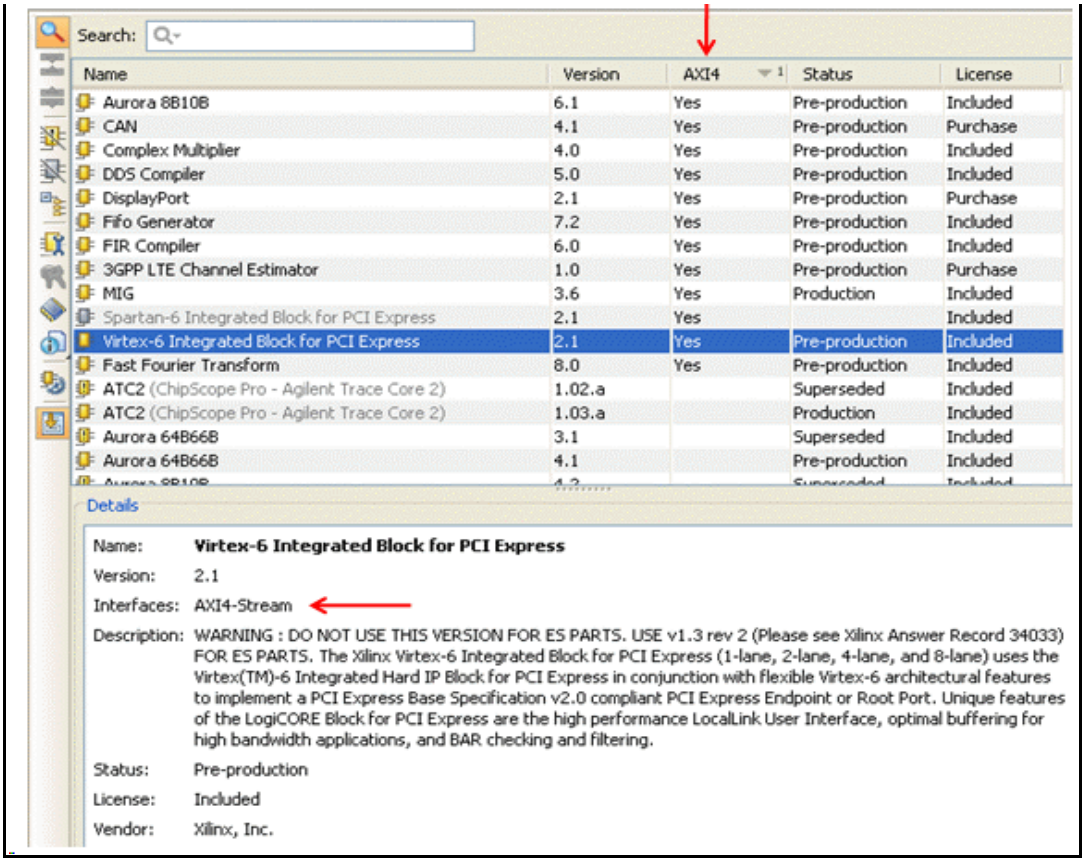

Figure 2-5: **IP Catalog in PlanAhead Software**

Xilinx AXI Infrastructure IP

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

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

- Xilinx AXI Interconnect Core IP
- Connecting AXI Interconnect Core Slaves and Masters
- External Masters and Slaves
- 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/ipcenter/axi4.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. IP with AXI4-Stream interfaces are generally connected to one another, and to DMA IP.

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

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 slaves by splitting transactions.
  - Generates `REGION` outputs for slaves with multiple address decode ranges
  - Propagates `USER` signals on each channel, if any; independent `USER` signal width per channel (optional)
  - Propagates Quality of Service (QoS) signals, if any; not used by the AXI Interconnect core(optional)
- Interface data widths:
  - AXI4: 32, 64, 128, or 256 bits.
  - AXI4-Lite: 32 bits
  - 32-bit address width
- 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, the AXI Interconnect core is implemented as wires, with no resources, no delay and no latency.
- Built-in data-width conversion:
  - Each master and slave connection can independently use data widths of 32, 64, 128, or 256 bits wide:
    - The internal crossbar can be configured to have a native data-width of 32, 64, 128, or 256 bits.
    - Data-width conversion is performed for each master and slave connection that does not match the crossbar native data-width.
  - When converting to a wider interface (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 re-synchronized to the clock rate of each connected master and slave.
- 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 results in a burst of 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 datapaths connecting to each master and each slave.
    - 32-deep LUT-RAM based.
    - 512-deep block RAM based.
- 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 may occur independently and concurrently, provided AXI ordering rules are met.
        - Sparse crossbar data pathways according to configured connectivity map, resulting in reduced resource utilization.

    - One shared Write address arbiter, plus one shared Read address arbiter. Arbitration latencies typically do not impact data throughput when transactions average at least three data beats.
  - Shared Access mode (Area optimized):
    - Shared write data, shared read data, and single shared address pathways.
    - Issues one outstanding transaction at a time.
    - Minimizes resource utilization.
- Supports multiple outstanding transactions:
  - Supports masters with multiple reordering depth (ID threads).
  - Supports up to 16-bit wide ID signals (system-wide).
  - Supports Write response re-ordering. Read data re-ordering, and Read Data interleaving.
  - Configurable Write and Read transaction acceptance limits for each connected master.
  - Configurable Write and Read transaction issuing limits for each connected slave.
- "Single-Slave per ID" method of cyclic dependency (deadlock) avoidance:
  - For each ID thread issued by a connected master, the master can have outstanding transactions to only one slave for Writes and one slave for Reads, at any time.
- Fixed priority and round-robin arbitration:
  - 16 configurable levels of static priority.
  - Round-robin arbitration is used among all connected masters configured with the lowest priority setting (priority 0), when no higher priority master is requesting.
  - Any master that has reached its acceptance limit, or is targeting a slave that has reached its issuing limit, or is trying to access a slave in a manner that risks deadlock, is temporarily disqualified from arbitration, so that other masters may be granted arbitration.
- Supports TrustZone security for each connected slave as a whole:
    - If configured as a secure slave, only secure AXI accesses are permitted
    - Any non-secure accesses are blocked and the AXI Interconnect 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 to the slave.
  - Write interleaving; this feature was retracted by AXI4 protocol. AXI3 masters must be configured as if connected to a slave with Write interleaving depth of one.
- AXI4 QoS signals do not influence arbitration priority. QoS signals are propagated from masters to slaves.
- The AXI Interconnect 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 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.)

AXI Interconnect Core Diagrams

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

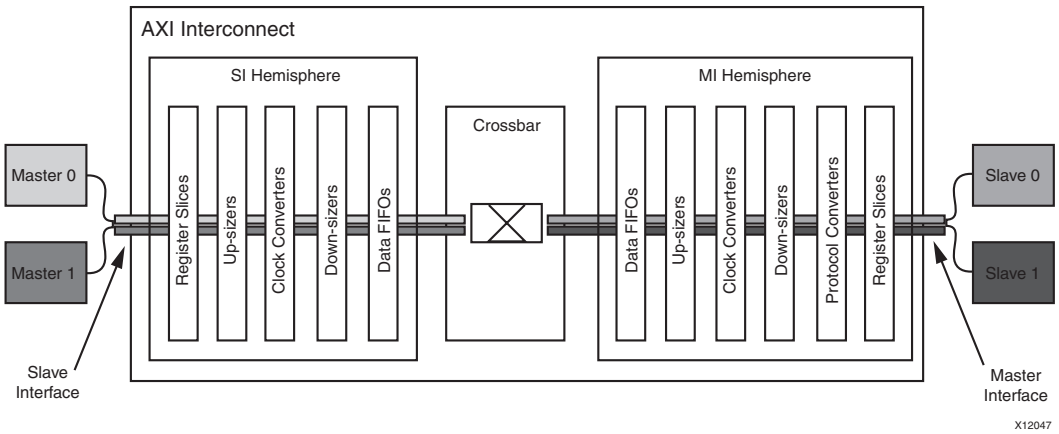

Figure 2-6: **Top-Level AXI Interconnnect**

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)
- N-to-M Interconnect (Shared Access Mode)

Pass Through

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

Figure 2-7 shows the Pass Through diagram.

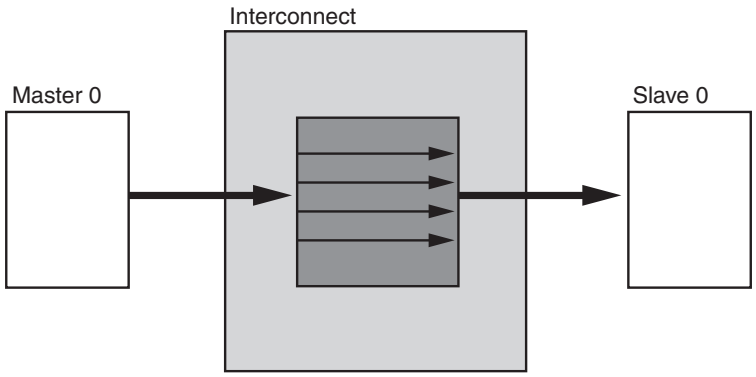

Figure 2-7: **Pass-through AXI Interconnect Use Case**

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 latency incurred, depending on the conversion being performed.

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

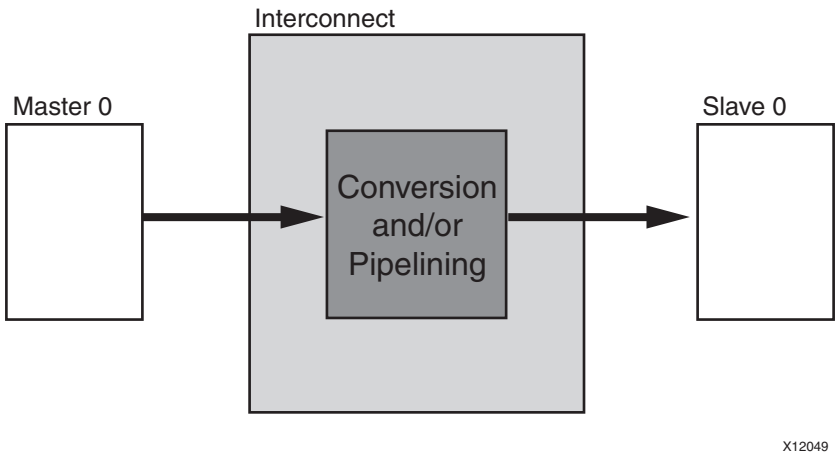

Figure 2-8: **1-to-1 Conversion AXI Interconnect Use Case**

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.

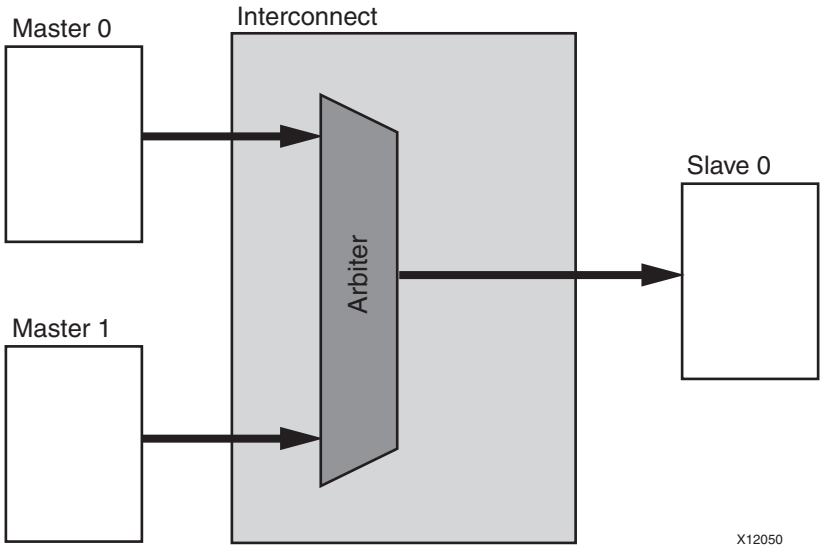

Figure 2-9: **N-to-1 AXI Interconnect**

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, page 25, shows the 1 to N Interconnect use case.

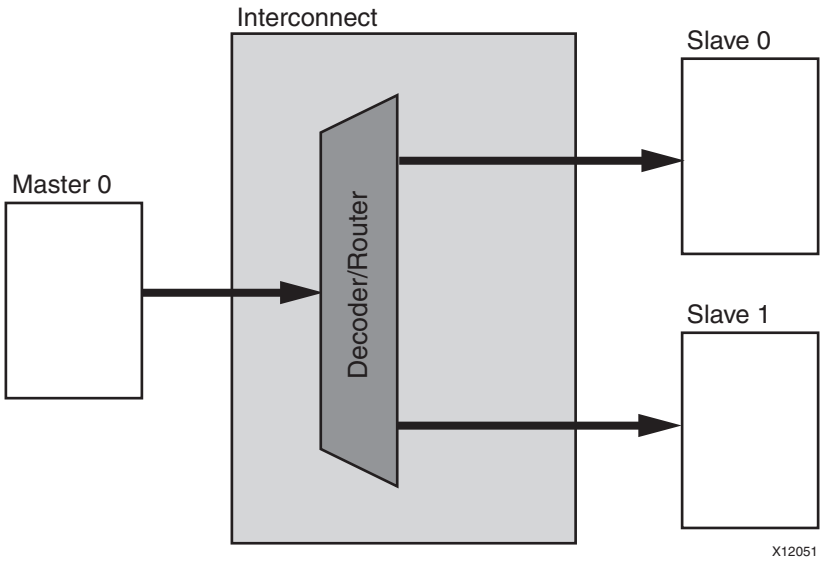

Figure 2-10: **1-to-N AXI Interconnect Use Case**

N-to-M Interconnect (Sparse Crossbar)

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.

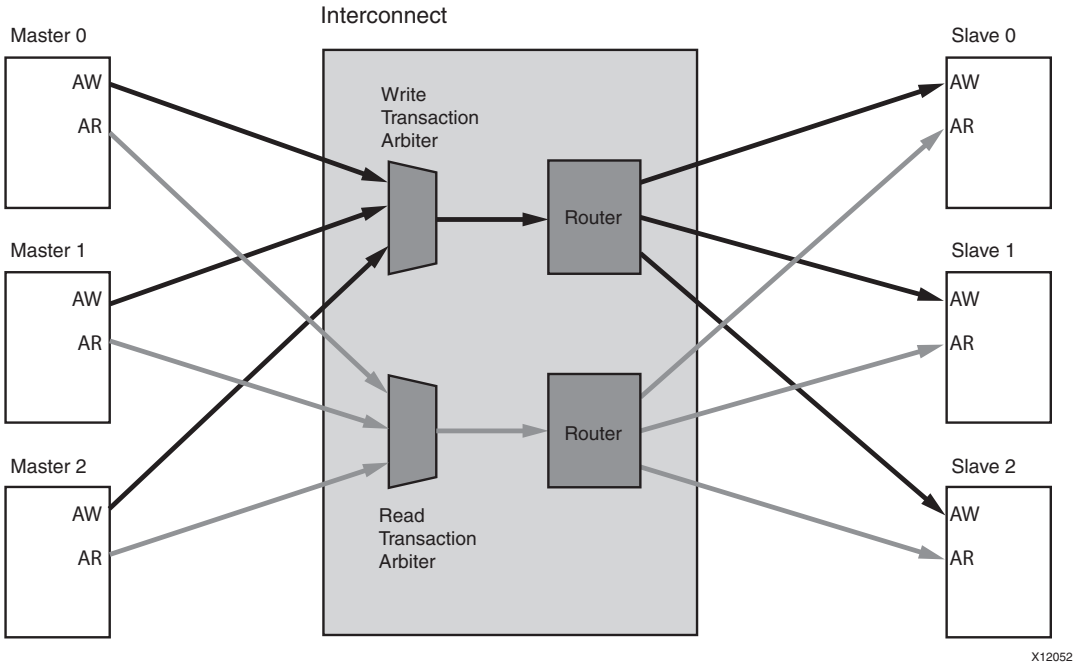

Figure 2-11: **Shared Write and Read Address Arbitration**

Figure 2-12, page 26 shows the sparse crossbar Write and Read data pathways.

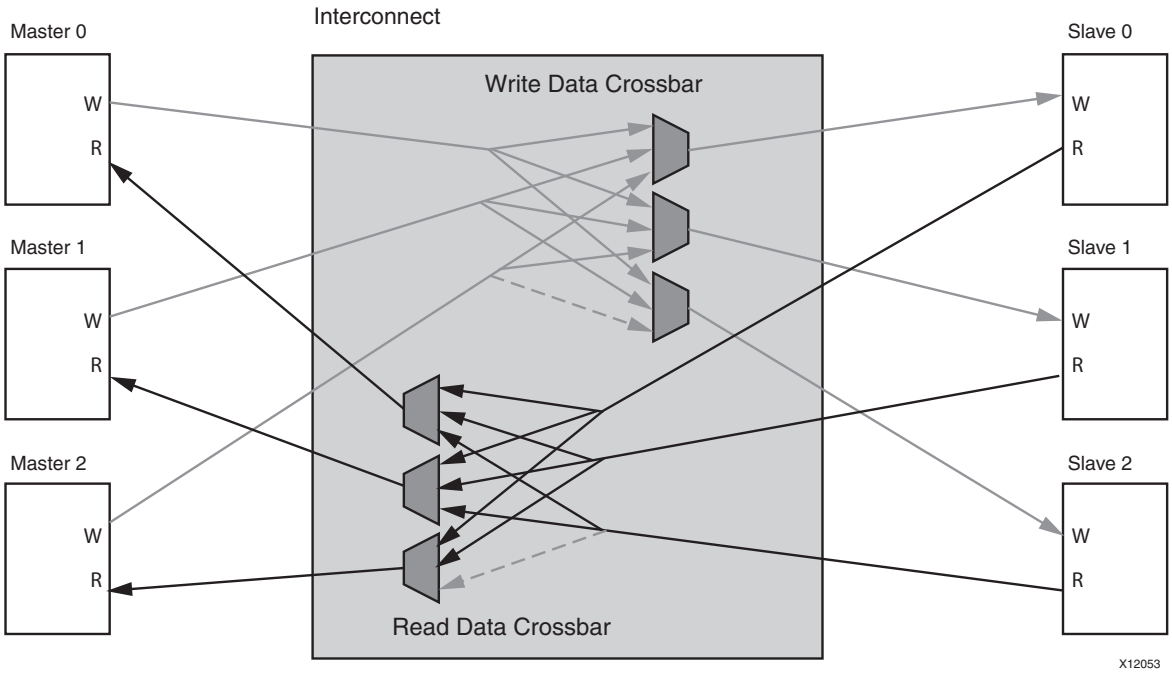

Figure 2-12: **Sparse Crossbar Write and Read Data Pathways**

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.

Width Conversion

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

When any SI slots or MI slots are sized differently, the AXI Interconnect 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 ("up-sizing") or more narrow ("down-sizing") when moving in the direction from the SI toward the MI. The width conversion functions are the same in either the SI hemisphere (translating from the SI to the AXI Interconnect 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 down-sizing 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 down-sizing conversion is performed along the pathways of the MI slot.
- When the data width of an SI slot is narrower than the internal native data width of the AXI Interconnect, an up-sizing conversion is performed along the pathways of the SI slot.
- When the internal native data width of the AXI Interconnect core is narrower than that of an MI slot, an up-sizing conversion is performed along the pathways of the MI slot.

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

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

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 down-sizing is performed and, in the transaction issued to the MI side, the number of data beats is multiplied accordingly.

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

When the transfer size of the transaction is equal to or less than the MI side data width, the transaction (address channel values) remains unchanged, and data transfers pass through unchanged except for byte-lane steering. This applies to both writes and reads.

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.

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, page 28. For each connected master, Read transaction requests always take priority over Writes. The arbiter then selects from among the requesting masters. A Write or Read data transfer is enabled to the targeted slave device. After the data transfer (including the Write response) completes, the next request is arbitrated.

Shared Access mode minimizes the resources used to implement the crossbar module of the Interconnect. Figure 2-13 illustrates the Shared Access mode.

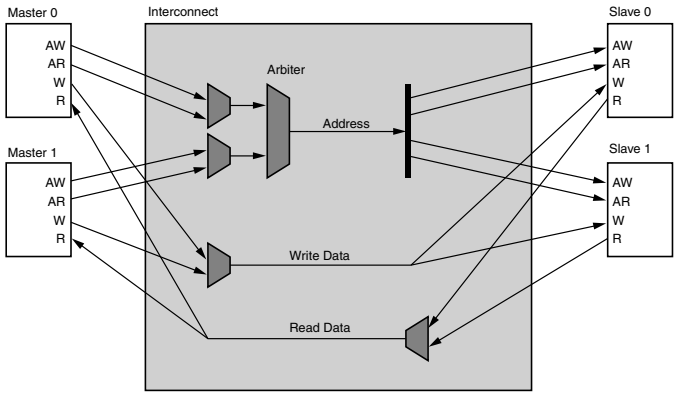

Figure 2-13: **Shared Access Mode**

Clock Conversion

Clock conversion comprises the following:

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

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

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

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

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

Data FIFOs are synchronized to the AXI Interconnect 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)*, available at the Xilinx website: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm.

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, page 30.

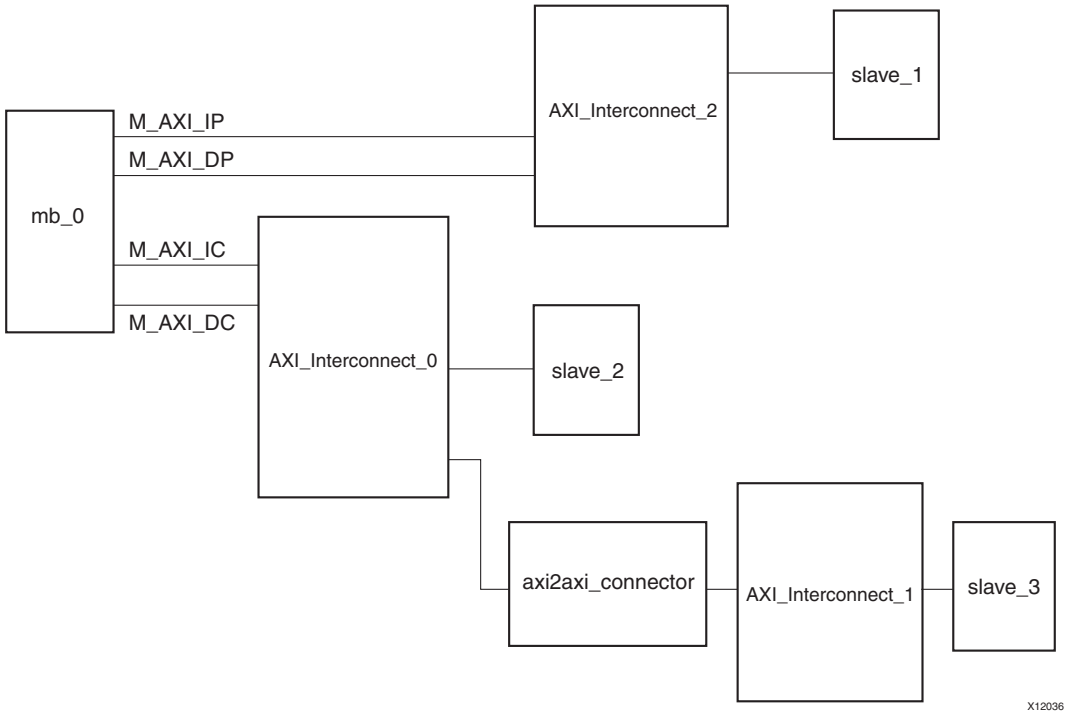

Figure 2-14: **Master and Slave Interface Modules Connecting Two AXI Interconnect cores**

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, refer to the *AXI To AXI Connector IP Data Sheet (DS803)*, available at the Xilinx website: http://www.xilinx.com/support/documentation/axi_ip_documentation.htm.

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, these utility cores can be used for that the purpose. The AXI master or slave module would remain in the top-level of the design, and the AXI signals would be connected to the EDK sub system using this utility pcore.

Features

- Connects an AXI master or slave interface to the AXI Interconnect core IP.
- A master or slave AXI bus interface on one side and AXI ports on the other side.
- Other ports are modeled 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, page 31 is a block diagram of the AXI external master connector.

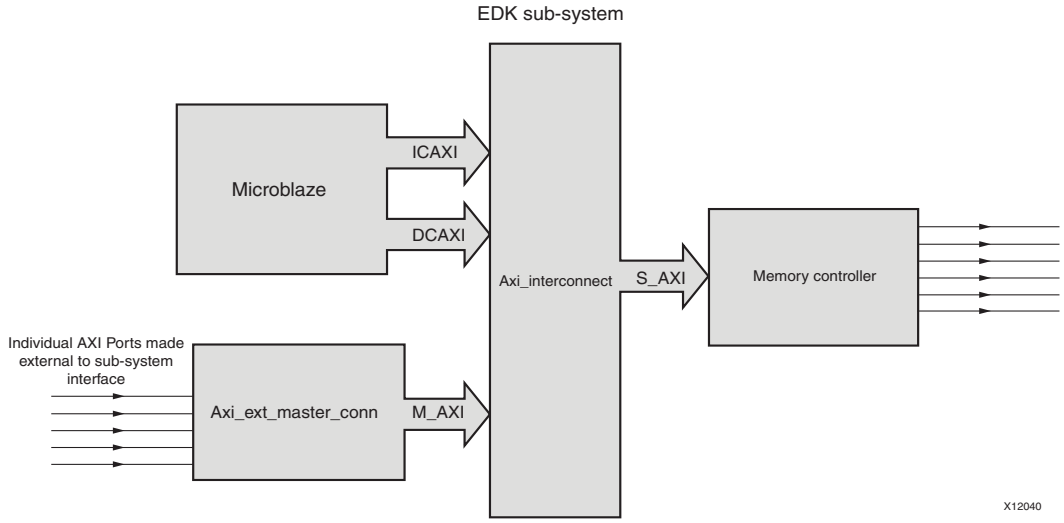

Figure 2-15: **EDK Sub-system using an External Master Connector**

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

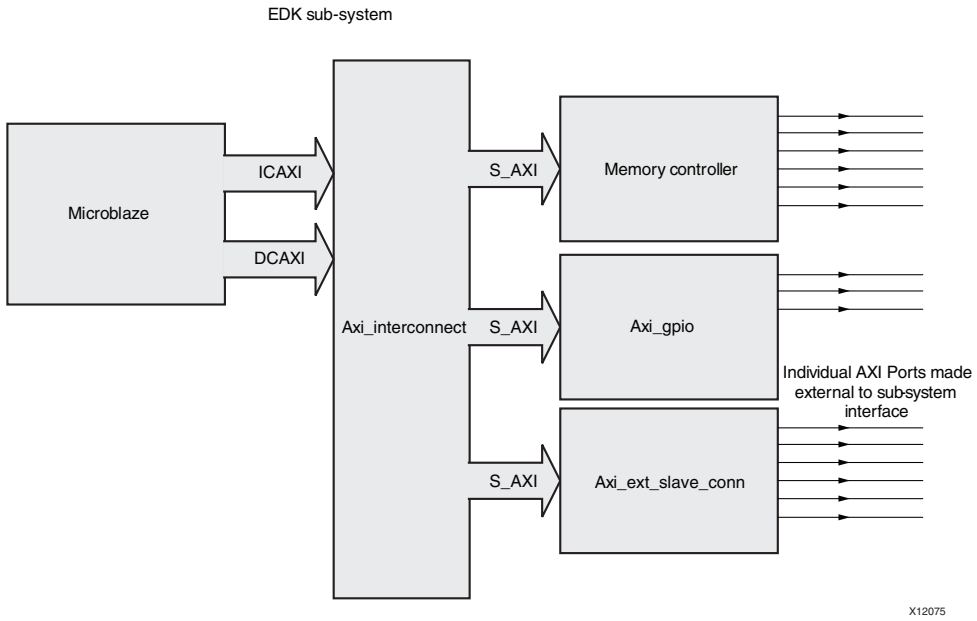

Figure 2-16: **EDK Subsystem using an External Slave Connector**

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/ipcenter/axi4.htm.

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 2-17 shows a typical embedded system architecture incorporating the AXI (AXI4 and AXI4-Lite) Centralized DMA.

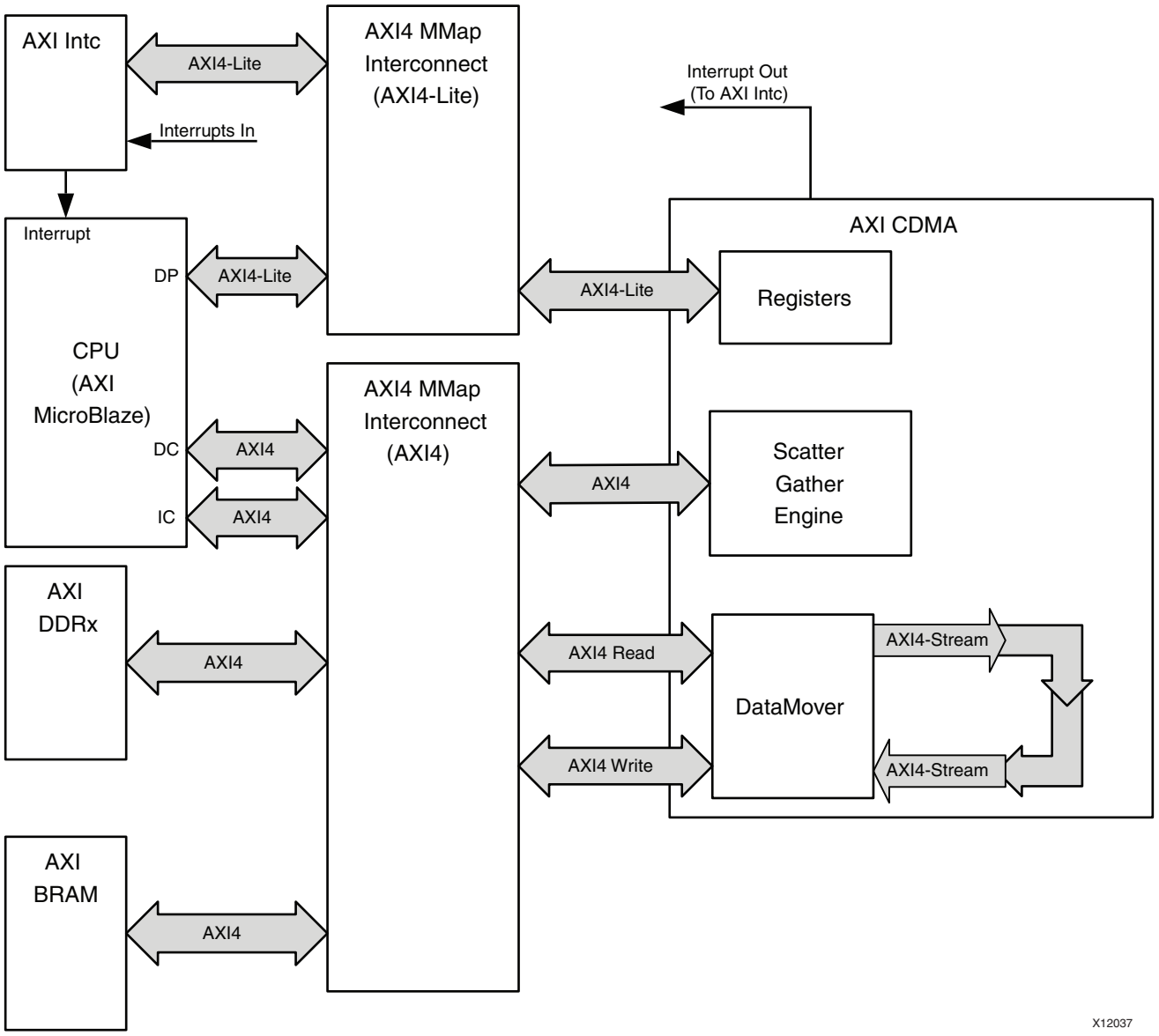

Figure 2-17: **Typical Use Case for AXI Centralized DMA**

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 ar e 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 (only available for 32- and 64-bit data transfer bus widths)
- Specify the main data transfer bus width (32, 64, 128, or 256 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	Reads the transfer payload data from the memory mapped source address. The data width is parameterizable to be 32, 64, 128, and 256 bits wide.
Data MMap Write	AXI4 Write master	32, 64, 128, 256	Writes the transfer payload data to the memory mapped destination address. The data width is parameterizable to be 32, 64, 128, and 256 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 2-18, page 35 provides high-level block diagram of the AXI DMA.

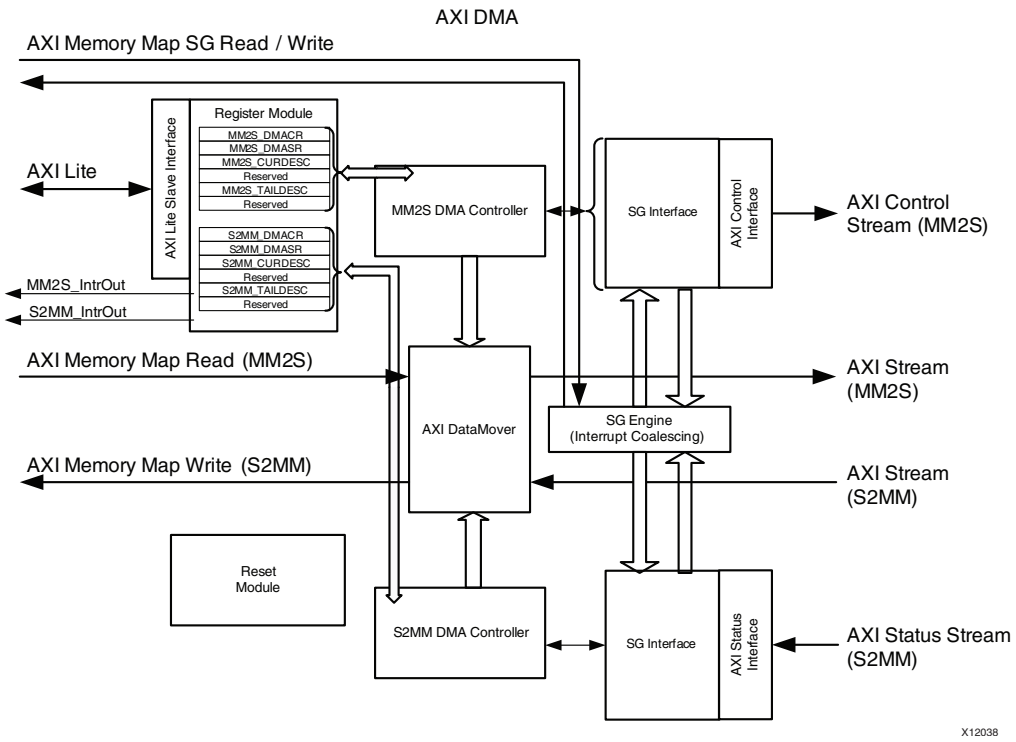

Figure 2-18: **AXI DMA High Level Block Diagram**

Figure 2-19 shows a typical system architecture for the AXI Ethernet.

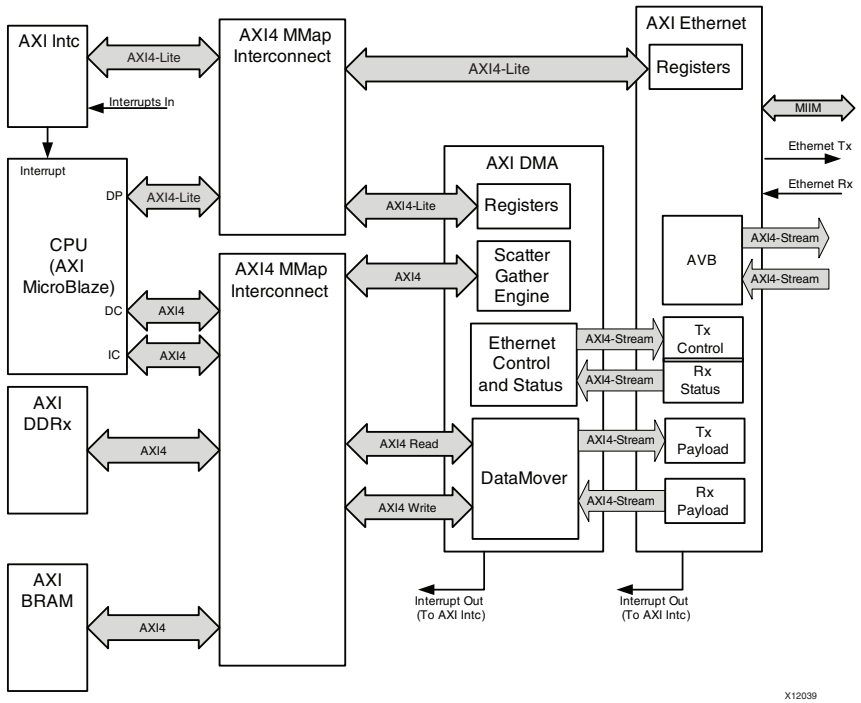

Figure 2-19: **Typical Use Case for AXI DMA and AXI4 Ethernet**

As shown in Figure 2-19, page 35, 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.

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 uses the AXI4 capabilities extensively. The following table 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	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	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	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	32, 64, 128, 256	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 2-20 shows a top-level AXI4 VDMA block diagram.

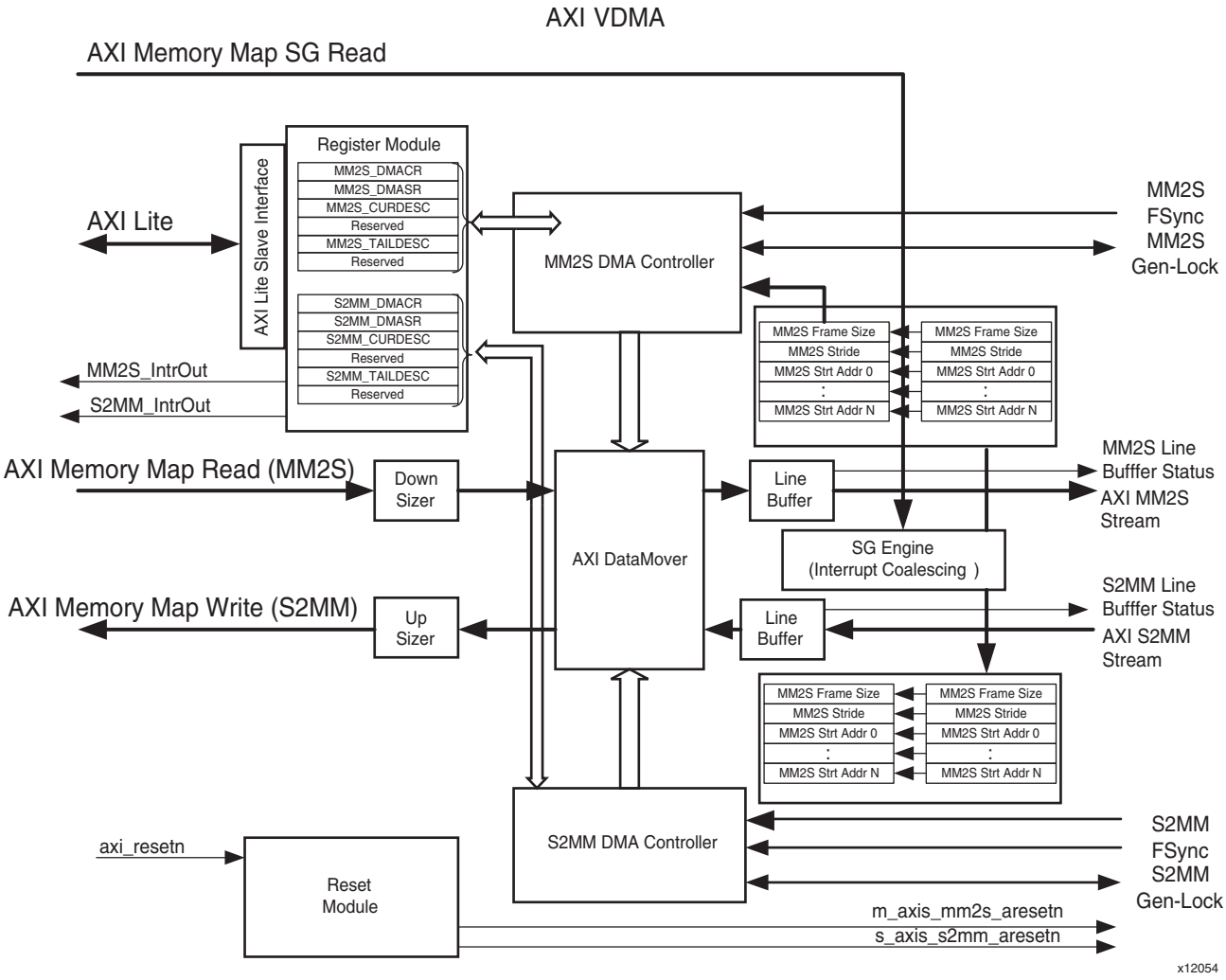

Figure 2-20: **AXI VDMA High-Level Block Diagram**

Figure 2-21, page 39 illustrates a typical system architecture for the AXI VDMA.

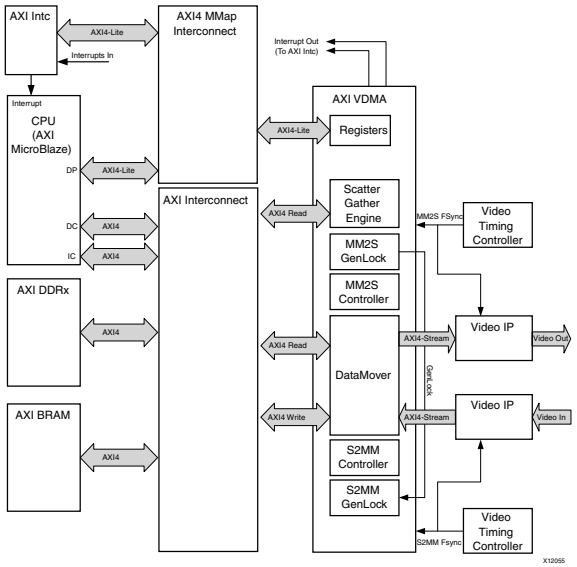

Figure 2-21: **Typical Use Case for AXI VDMA and Video IP**

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 16 frame buffers and provides the tools to transfer portions of video frames or full video frames.

The VDMA provides the ability to “park” on a frame also, allowing the same video frame data to be transferred repeatedly.

VDMA provides 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 such that 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 256 bits wide. 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.

VDMA AXI4 Interfaces

Table 2-3 lists and describes six AXI4 interfaces of the AXI DMA function.

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	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	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	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	Receives data from the source IP using the AXI4-Stream protocol. The data received is then transferred to the Memory Map system via 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 software 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 2-22 shows a block diagram of the Virtex-6 memory solution with the AXI4 interface.

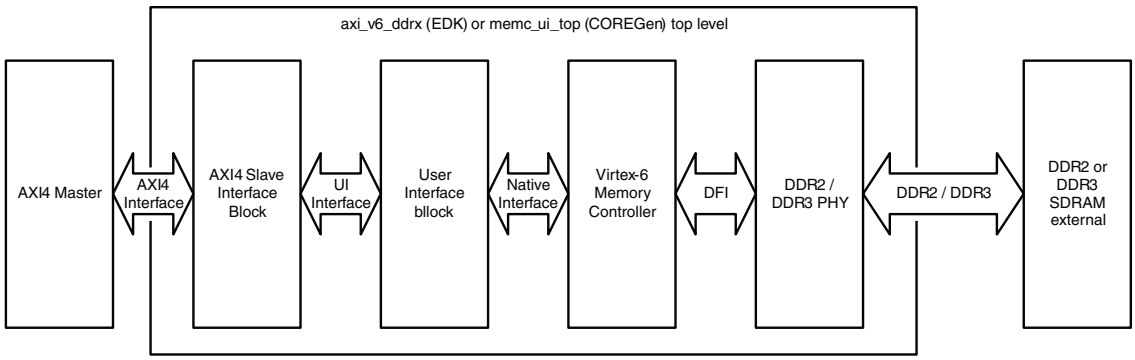

Figure 2-22: **Virtex-6 Memory Control Block Diagram**

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 2-23 shows a block diagram of the AXI Spartan-6 memory solution.

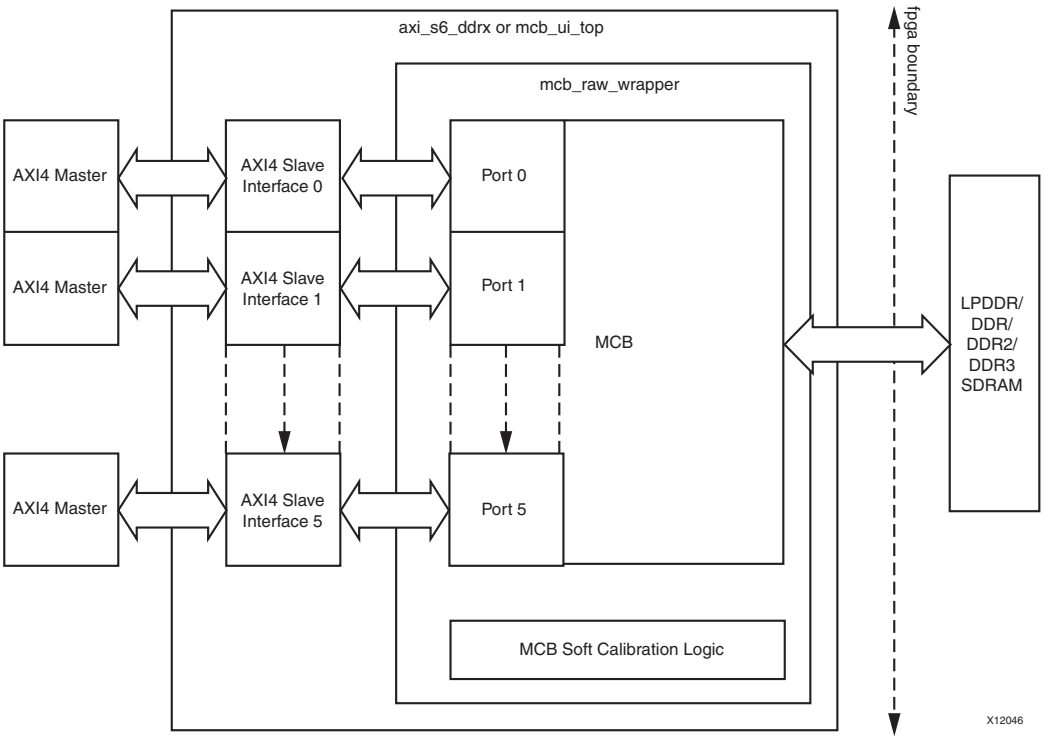

Figure 2-23: **Spartan-6 Memory Solution Block Diagram**

For more detail on memory control, refer to the memory website documents at http://www.xilinx.com/products/design_resources/mem_corner.

XILINX®

Chapter 3

AXI Feature Adoption in Xilinx FPGAs

This chapter describes how Xilinx® utilized specific features from the AXI standard in Xilinx IP. The intention of this chapter is 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 and · 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, and 256 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.
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.

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

AXI Feature	Xilinx IP Support
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 outputs within eight cycles of reset, and have a reset pulse width requirement of 16 cycles or greater. Holding AXI `ARESETN` asserted for 16 cycles of the slowest AXI clock is generally a sufficient reset pulse width for Xilinx IP.
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 simplify interoperability, Xilinx IP requiring streaming interfaces use a strict subset of the AXI4-Stream protocol.

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 end-point IP; available for sparse stream signalling. Note: For marking packet remainders, TKEEP rather than TSTRB is used.
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 end-point IP; available for use by infrastructure IP.
TDEST	Optional	Not typically used by end-point 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 rom 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.

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.

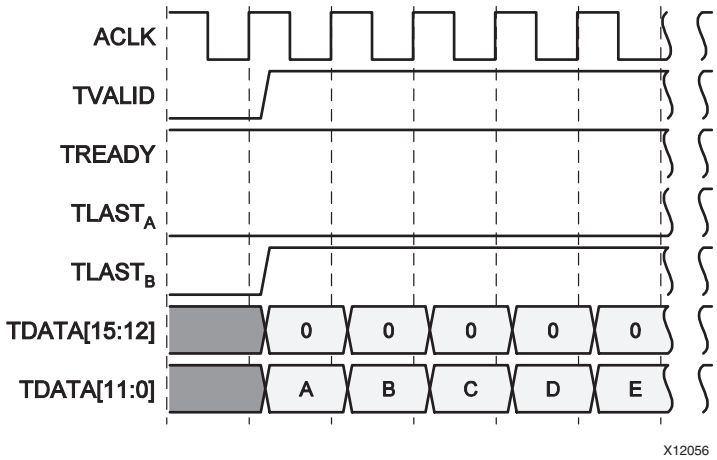

Figure 3-1: **Real Scalar (Unsigned) Data Example in an AXI4-Stream**

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:

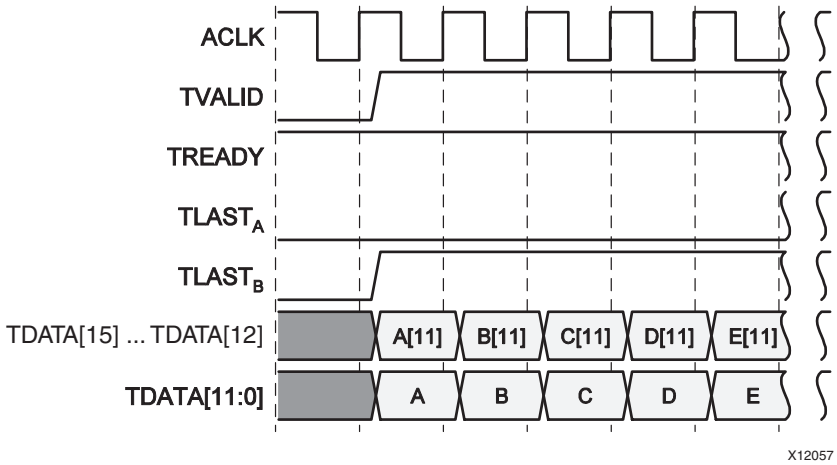

Figure 3-2: **Alternative (Sign-Extended) Scalar Value Example**

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:

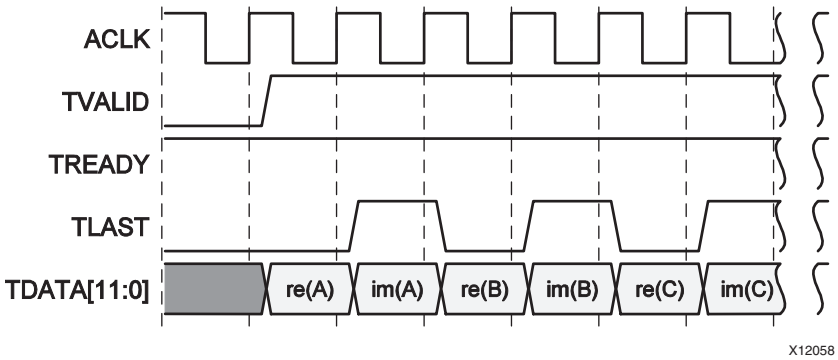

Figure 3-3: **Complex Scalar Data Example in AXI4-Stream**

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:

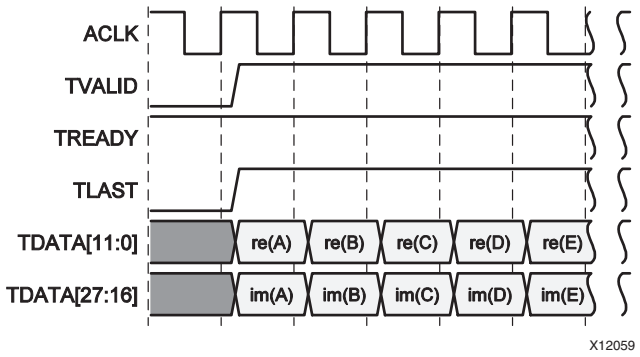

Figure 3-4: **Complex Scalar Example with 32-Bit TDATA Signal**

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.

Vector Data Example

Consider a numerical stream with the characteristics listed in the following table:

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:

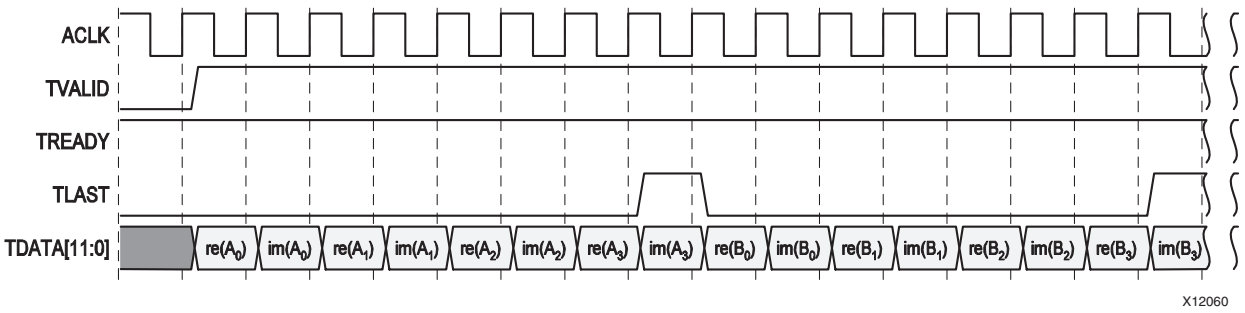

Figure 3-5: **Numerical Stream Example**

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:

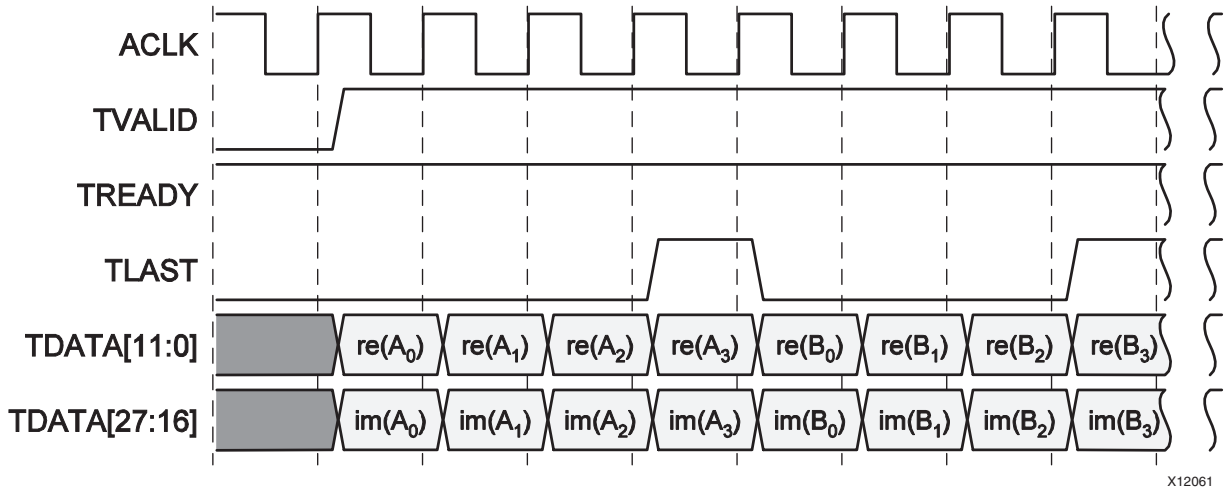

Figure 3-6: **AXI4-Stream Scalar Example**

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

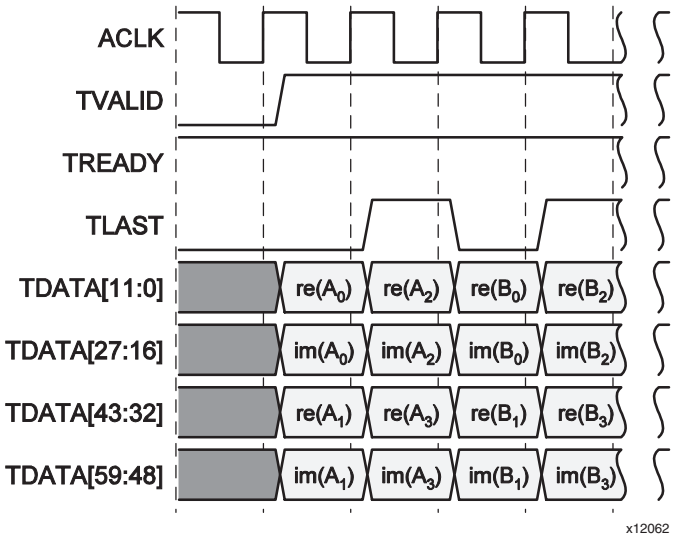

Figure 3-7: **TDATA Example with 64-Bits**

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

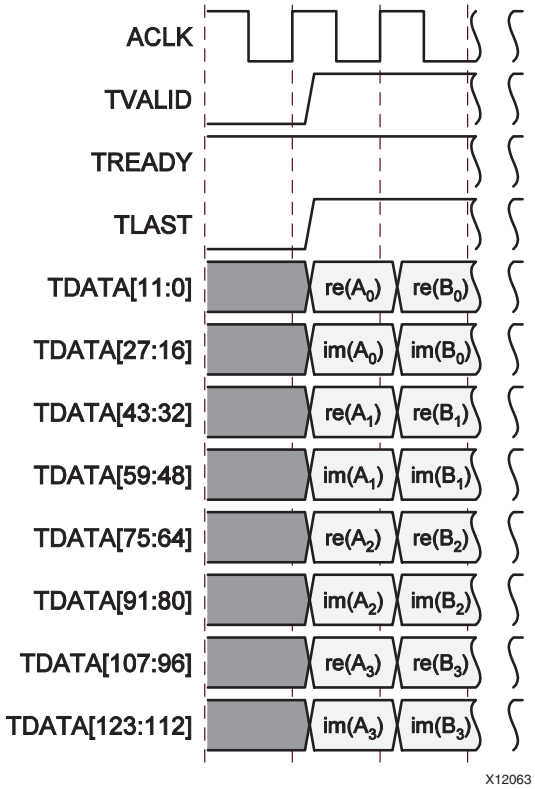

Figure 3-8: **128 Bit TDATA Example**

As shown for the scalar data in the preceding figures, there are multiple representations that can be tailored 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 needed 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 may 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 since Xilinx IP will 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 un-scaled 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.

It should be noted that 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.

XILINX®

Chapter 4

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 *endianan-ness* to go from having a Big-endian orientation, (which aligned with the PLB interfaces of the PowerPC® embedded processors), to Little-endian (which aligns with ARM® processor requirements and the AXI protocol). Migration considerations for software are covered in Using CORE Generator to Upgrade IP, page 68.

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:

- Were created from scratch.
- Were created using the Create and Import IP 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; refer to Memory Mapped IP Feature Adoption and Support, page 43 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 used the Create and Import Peripheral (CIP) Wizard in previous version of Xilinx tools can be migrated using templates provided in solution record: http://www.xilinx.com/support/answers/37425.htm.

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 PLB Bridge," 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.

DSP IP: General guidelines on converting this broad class of IP is covered in.

Local-Link Interface: Local-Link is a generic streaming, FIFO-like interface that has been in service at Xilinx for a number of software generations. See Migrating Local-Link to AXI4-Stream, page 60 for more information.

The AXI To PLB Bridge

For Xilinx embedded processor customers who have developed their own PLBv4.6-based IP, the AXI to PLB Bridge provides a mechanism 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 interfaces
- 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.

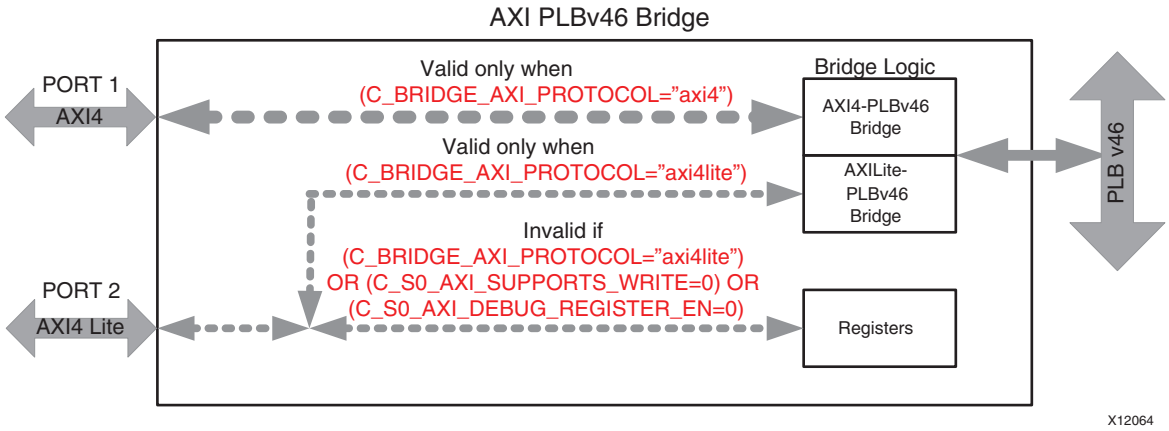

Figure 4-1: **AXI to PLBv4.6 Block Diagram**

- 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	<none>
EOF_N	src to dst	End-of-Frame: Indicates the last beat of a frame	TLAST
CLK	Input	Clock: all signals are synchronous to this clock	ACLK

- 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 Read 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). Consequently, the signal is no longer needed for the interface. This signal could be applied to the TUSER field.

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

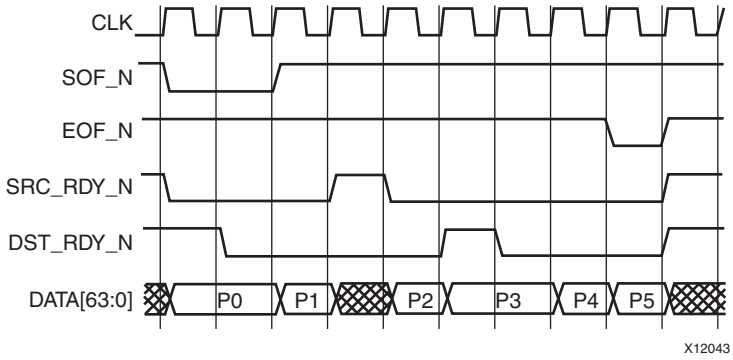

Figure 4-2: **Single Local-Link Waveform**

Figure 4-2 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.

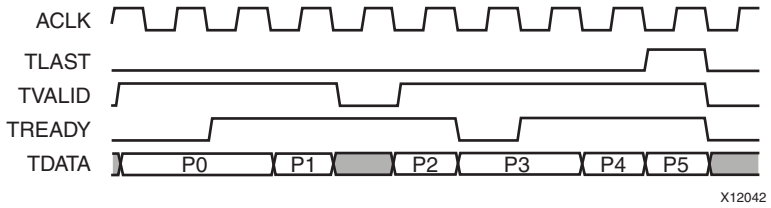

Figure 4-3: **AXI4-Stream Waveform**

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 may 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 software.

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:

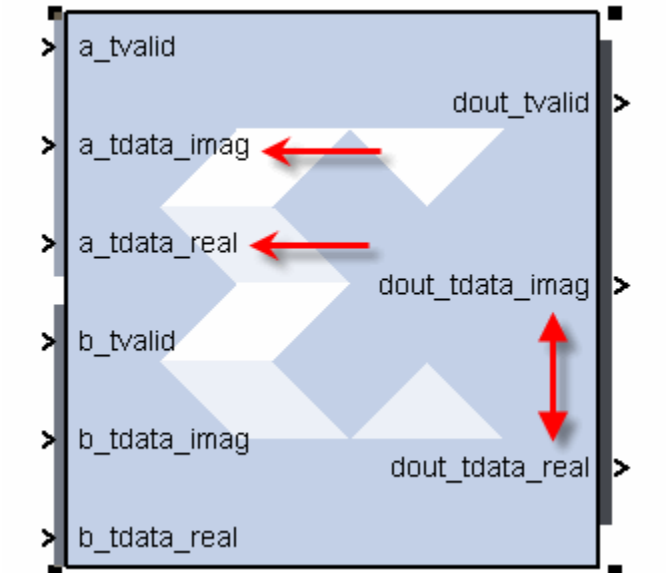

Figure 4-4: **AXI Complex Multiplier Block**

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.

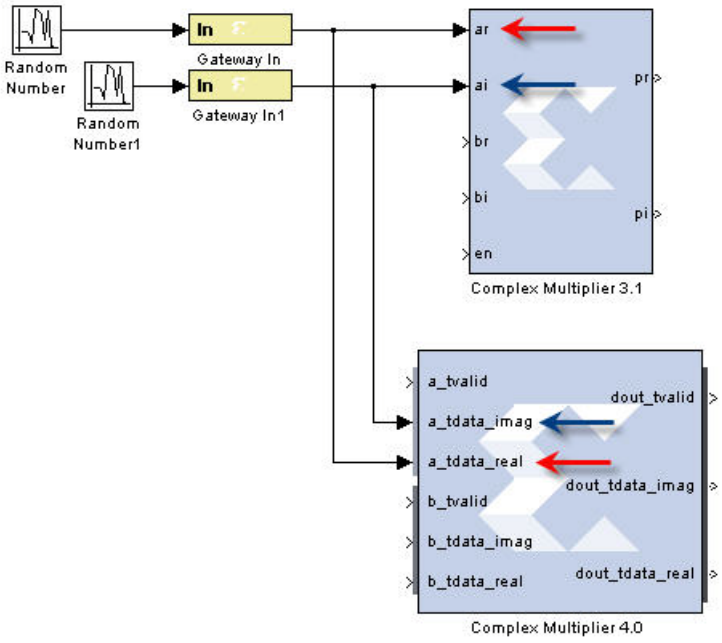

Figure 4-5: **Port Signal Example**

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.

Output Width Specification

Specification for the output word width is different for non-AXI and AXI IP in System Generator. In 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 (pcore), as shown in Figure 2-1, page 15.

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 Signal 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

Table 4-3: **AXI4-Stream Signal Mapping *(Cont'd)***

Signal	Direction	AXI Signal	Direction
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:

Signal	Direction	AXI Signal	Direction
FSL_S_Clk	Out	S_AXIS_<*Port_Name*>ACLK	In
FSL_S_Write	In	S_AXIS_<*Port_Name*>TVALID	In
FSL_S_Full	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 consideration items.

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.

Demonstration Testbench

Figure 4-6 shows an Example IP Data Sheet list.

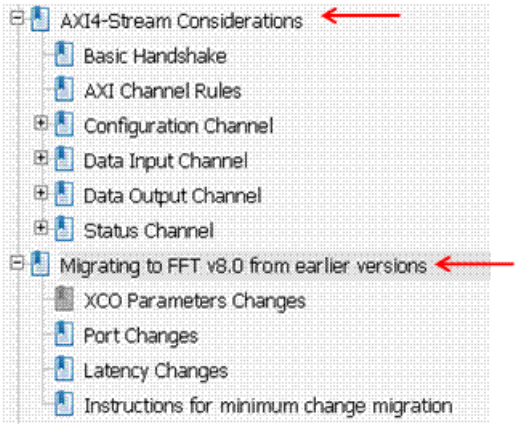

Figure 4-6: **Example IP Data Sheet**

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 Multipler 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.

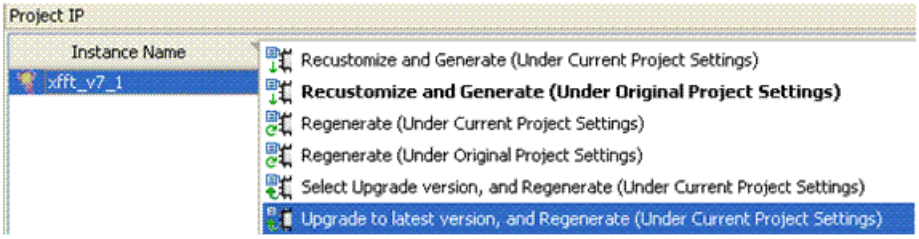

Figure 4-8: **CORE Generator Upgrade Function**

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

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. 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.

Slave FSL to AXI4-Stream Signal Mapping

Table 4-4 shows the FSL to AXI4-Stream Signal Mapping.

Table 4-5: **FSL to AXI4-Stream Signal Mapping**

Signal	Direction	AXI Signal	Direction
FSL_S_Clk	Out	S_AXIS_<*Port_Name*>ACLK	In
FSL_S_Write	In	S_AXIS_<*Port_Name*>TVALID	In
FSL_S_Full	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

Software 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 software tool flows through the Xilinx Software Development Kit (SDK) remain the same. The IDE and underlying software 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 software 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 software 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 endianness. Once the hardware migration has been completed, migrate software 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.

User code, custom drivers or other code might need to be rewritten if the code is sensitive to the representation of data in big-endian or little-endian formats.

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 endianess, 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 software flows do not support AXI; SDK must be used.
   b. In SDK, import the new AXI hardware handoff, create new board software 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 endianess.
   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 endianess (for example BIT files that include block RAM data - like ELF files) across systems.
   e. Block RAM initialization and data sharing should reflect the endianess requirements of the master.
   f. *Do not* use old application data files used by the application if it is affected by endianess (byte ordering in the file).
   g. Be aware of the endianess of MicroBlaze and write software 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)*.

Data Types and Endianness

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 4-6 through Table 4-8, page 72.

Table 4-6: **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 4-7: **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
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 4-8: **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.

XILINX®

Appendix A

AXI Adoption Summary

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.

AXI4 and AXI4-Lite Signals

Global Signals

Table A-1 lists the Global AXI signals.

Table A-1: **Global AXI Signals**

Signal	AXI4	AXI4-Lite
ACLK	Clock source. An associated clock-enable signal, ACLKEN, is sometimes also used. However, note that ACLKEN is used by industry convention, but is not explicitly listed in the ARM specification.	
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 outputs within 8 cycles of reset assertion, and generally require a reset pulse-width of 16 or more clock cycles of the slowest clock.	

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.

Table A-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 may 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 end-point slave, which could result in address aliasing within the slave. ***Note:*** EDK supports 32-bit address only.	

Table A-2: **Write Address Channel Signals *(Cont'd)***

Signal	AXI4	AXI4-Lite
AWLEN	Fully supported. Support bursts: • Up to 256 beats for incrementing (INCR). • 16 beats for WRAP.	Signal not present.
AWSIZE	Transfer width 8 to 256 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.
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.

Table A-3: **Write Data Channel Signals**

Signal	AXI4	AXI4-Lite
WDATA	Native width 32 to 256 bits supported. 512 and 1024 widths may be supported in the future.	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.

Table A-3: **Write Data Channel Signals *(Cont'd)***

Signal	AXI4	AXI4-Lite
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.
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 may 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 end-point 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 may be aliased to `INCR` or undefined.	Signal not present.
ARSIZE	Transfer width 8 to 256 bits supported. 512 and 1024 widths may be supported in the future. 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.

Table A-5: **Read Address Channel Signals *(Cont'd)***

Signal	AXI4	AXI4-Lite
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.	
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 256 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.

Table A-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).

XILINX®

Appendix B

AXI Terminology

This appendix provides a list of AXI-specific acronyms, terminology, and definitions; and gives the IP core types in which the terminology is used.

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.
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.

Table B-1: **AXI Terminology *(Cont'd)***

Term	Type	Description	Usage
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: Vectored AXI slave interface receiving in-bound AXI transactions from all connected master IP.	EDK.
MI	AXI4 AXI4-Lite	AXI Interconnect Master Interface: Vectored AXI master interface generating out-bound AXI transactions to all connected slave IP.	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.	EDK.
MI-side	AXI4 AXI4-Lite	Refers to a module interface closer to the MI side of the Interconnect.	EDK.
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	EDK.
Crossbar	Topology	Module at the center of the AXI Interconnect that routes address, data and response channel transfers between various SI slots and MI slots.	All.