DesignWare® DW_axi_x2x

Databook

DW_axi_x2x – Product Code

1.08a
March 2020

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.01c onward.

Version	Date	Description
1.08a	March 2020	■ Version change for 2020.03a ■ Updated synthesis results in "Performance" on page 93 ■ Updated "Verification" on page 89 ■ Updated "Quasi-Synchronous Clocking" on page 38 ■ Updated "Reset Control" on page 39 ■ Updated "Synchronizers used in DW_axi_x2x" on page 100 ■ Renamed Synchronizer chapter to "Basic Core Module (BCM) Library" ■ Renamed Clocking to "Clocks and Resets" on page 37 ■ "Signal Descriptions" on page 53, "Parameter Descriptions" on page 41, "Internal Parameter Descriptions" on page 87 auto-extracted from the RTL
1.07a	February 2018	■ Version change for 2018.02a ■ Updated synthesis results in "Performance" on page 93 ■ Removed Chapter 2 Building and Verifying a Component or Subsystem from the databook and added the contents in the newly created user guide ■ "Signal Descriptions" on page 53, "Parameter Descriptions" on page 41, "Internal Parameter Descriptions" on page 87 auto-extracted from the RTL with change bars
1.06a	March 2016	■ Version change for 2016.03a ■ Added "Running SpyGlass® Lint and SpyGlass® CDC" section ■ Added"Running Spyglass on Generated Code with coreAssembler" section ■ "Signal Descriptions" on page 53 and "Parameter Descriptions" on page 41 auto-extracted from the RTL ■ Added chapter "Internal Parameter Descriptions" on page 87 ■ Added Appendix A, "Basic Core Module (BCM) Library" ■ Updated power and area numbers in "Performance" on page 93
1.05a	October 2014	Version change for 2014.10a release. Updated the "Performance" section in the "Integration Considerations" chapter.

Version	Date	Description
1.04d	Jun 2013	Version change for 2013.06a.
1.04c	May 2013	Version change for 2013.05a release. Updated the template.
1.04b	Oct 2012	Added the product code on the cover and in Table 1-1.
1.04b	Feb 2012	Added information on when exclusive writes are not supported.
1.04b	Oct 2011	Added material for extended burst mode.
1.04a	Jun 2011	Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface.
1.04a	May 2011	Revised material regarding fan-out and burst length translation.
1.04a	Jan 2011	Increased legal X2X_MAX_UWIDA and X2X_MAX_URIDA values from 32 to 64.
1.03b	Nov 2010	Corrected value of X2X_CLK_MODE from 0 to 1 in “Quasi-Synchronous Clocking” section; corrected dependencies for when X2X_HAS_PIPELINE parameter is disabled.
1.03a	Sep 2010	Added discussion for X2X_HAS_WRAP_BURST regarding performance impact of supporting wrapped busts for resize configurations; enhanced material for latency calculations; corrected names of include files and vcs command used for simulation.
1.02c	Dec 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.02c	May 2009	Removed references to QuickStarts, as they are no longer supported.
1.02c	Mar 2009	Removed references to X2X_HAS_WI_FAN_OUT and X2X_HAS_NOPX.
1.02b	Jan 2009	Matched Sideband Signals descriptions in parameters table with GUI descriptions; added “Locking Sequences” section; grammatical fix in X2X_MAX_UWIDA and X2X_MAX_URIDA descriptions.
1.02a	Oct 2008	Version change for 2008.10a release.
1.01d	Jul 2008	Clarified minimum buffer depth; corrected Write WRAP timing diagram.
1.01d	Jun 2008	Version change for 2008.06a release.
1.01c	Aug 2007	Corrected errors/omissions in “Latency” on page 95.
1.01c	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information that you need to interface the DW_axi_x2x component to the Synopsys DesignWare Synthesizable Components for AMBA environment. This component conforms to the AMBA 3 AXI specifications defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

The information in this databook includes a functional description, and signal and parameter descriptions, as well as information on how you can configure, create RTL for, simulate, and synthesize the component using coreConsultant. Also provided are an overview of the component testbench, a description of the tests that are run to verify the coreKit, and synthesis information for the coreKit.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_axi_x2x.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_x2x.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_axi_x2x signals.
- Chapter 5, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals and Parameters chapters.
- Chapter 6, "Verification" provides information on verifying the configured DW_axi_x2x.
- Chapter 7, "Integration Considerations" includes information you need to integrate the configured DW_axi_x2x into your design.
- Appendix A, "Basic Core Module (BCM) Library" documents the synchronizer methods (blocks of synchronizer functionality) used in DW_axi_x2x to cross clock boundaries.
- Appendix B, "Glossary" provides a glossary of general terms.

Related Documentation

- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI/AMBA 4 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 3 AXI and AMBA 4 AXI, refer to the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI*.

Web Resources

- DesignWare IP product information: https://www.synopsys.com/designware-ip.html
- Your custom DesignWare IP page: https://www.synopsys.com/dw/mydesignware.php
- Documentation through SolvNetPlus: https://solvnetplus.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): https://www.synopsys.com/keys

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

- Prepare the following debug information, if applicable:
  - For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the <*core tool startup directory*>/debug.tar.gz file.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD).
    - Identify the hierarchy path to the DesignWare instance.
    - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- *For the fastest response*, enter a case through SolvNetPlus:

  a. https://solvnetplus.synopsys.com

  Note SolvNetPlus does not support Internet Explorer. Use a supported browser such as Microsoft Edge, Google Chrome, Mozilla Firefox, or Apple Safari.

  b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).

  c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

  Ensure to include the following:
  - **Product L1:** DesignWare Library IP
  - **Product L2:** <name of L2>

  d. After creating the case, attach any debug files you created.

  For more information about general usage information, refer to the following article in SolvNetPlus:

  https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
    - Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
    - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
    - Attach any debug files you created.
- Or, telephone your local support center:
    - North America:
      Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
    - All other countries:
      https://www.synopsys.com/support/global-support-centers.html

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare AMBA Fabric.

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multi-layer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI and AMBA 4 AXI
DW_axi_a2x	Configurable bridge between AMBA 3 AXI/AMBA 4 AXI components and AHB components or between AMBA 3 AXI/AMBA 4 AXI components and AMBA 3 AXI/AMBA 4 AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_hmx	Configurable high performance interface from an AHB master to an AMBA 3 AXI or AMBA 4 AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI or AMBA 4 AXI subsystems

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0 (Continued)

Component Name	Description
DW_axi_x2h	Bridge from AMBA 3 AXI or AMBA 4 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or busses

1 Product Overview

The DW_axi_x2x is a configurable bridge between an AXI (Advanced eXtensible Interface) master and an AXI slave with any combination of different data port widths, different clock frequencies, and different endianness.

The DW_axi_x2x component has only an AMBA AXI-compliant interface and is part of the family of DesignWare Synthesizable Components for AMBA.

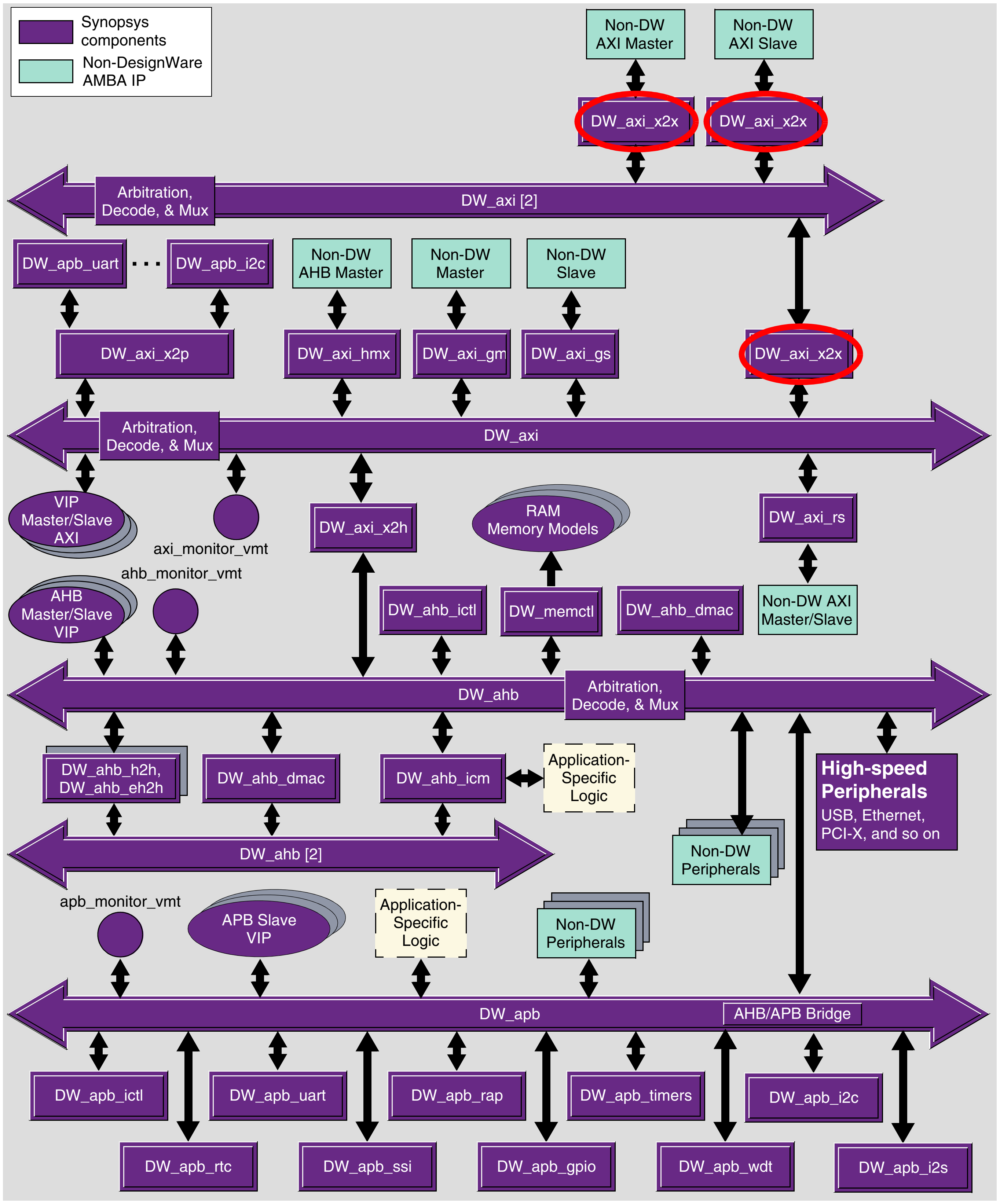
1.1 DesignWare Synthesizable Components for AMBA System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA 3 AXI/AMBA 4 AXI components.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Attention Links resolve only if you are viewing this databook from your $DESIGNWARE_HOME tree, and to only those components that are installed in the tree.

Figure 1-1 Example of DW_axi_x2x in a Complete System



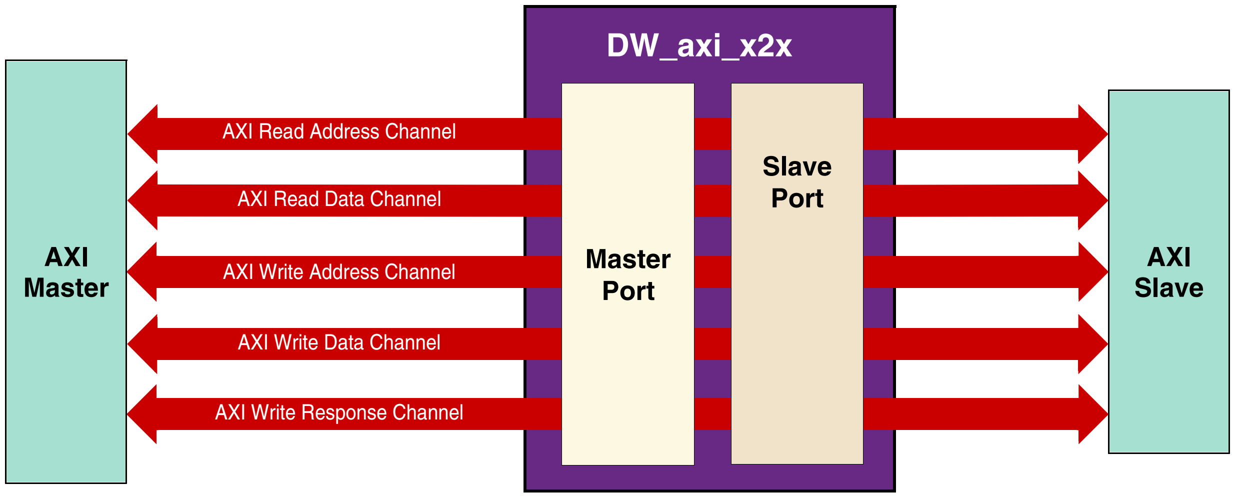
1.2 General Product Description

The DesignWare DW_axi_x2x provides a method to transfer transactions between an AXI master and an AXI slave.

1.2.1 DW_axi_x2x Block Diagram

Figure 1-2 illustrates the block diagram of the DW_axi_x2x.

Figure 1-2 DW_axi_x2x Block Diagram



1.3 DW_axi_x2x Features

The following subsections list the features of the DW_axi_x2x.

1.3.1 General Features

The general features of the DW_axi_x2x are:

- Complies with the AMBA 3 AXI protocol specification
- Transparent Operation—transaction attributes are altered only when necessary to fit on the secondary bus
- Accepts simultaneous AXI transfers on all five channels and passes to the primary/secondary AXI bus
- Configurable buffer depths for all 5 channels to allow off loading of the payload source bus to the payload sink bus
- Support for asynchronous clocks on Slave Port and Master Port sides
- Optimizations for quasi-synchronous clocking of any integer ratio
- Extended burst mode

1.3.2 Address Channel Features

The features of the DW_axi_x2x AXI address channels are:

- Address width configurable to be any value in the range 32->64 on both Slave and Master Ports. Internally the address will be limited to the smaller of the two sides.
- For transactions that cannot fit on the secondary bus the DW_axi_x2x will change their attributes. The DW_axi_x2x will issue more than one transaction if necessary to complete the transaction from the primary bus.
- For upsizing configurations the DW_axi_x2x can increase the SIZE of a transaction and decrease the length to generate a transaction of a SIZE equal to the maximum allowed by the secondary bus.
- The burst length width on both the master and slave ports can be configured between 4 and 8 (extended burst mode). Both the master and slave port can only have the same burst length width. Only the master port burst length width (X2X_MP_BLW) is configurable in coreConsultant/coreAssembler; the slave port burst length width parameter inherits this value automatically.

1.3.3 Data Channel Features

The features of the DW_axi_x2x AXI data channels are:

- Data Ports: 8, 16, 32, 64, 128, 256, 512 bits wide.
- Master Port and Slave Port data widths may be configured to different widths. Transfer unpacking will be done to translate a larger Master Port data width to a smaller Slave Port data width. On the return from Slave Port to Master Port the transfers will be packed to match the size of the original transaction from the primary bus.
- The DW_axi_x2x can upsize transactions i.e. pack data from the primary into a shorter transaction of larger SIZE on the secondary.
- AXI little and big-endian Byte Ordering (byte invariant) is individually configurable for each interface.
- Configurable read interleaving depth. Allows external slave to reorder and/or interleave read data up to this depth.
- Configurable write interleaving depth.

1.3.4 Low-Power Interface

The features of the DW_axi_x2x AXI low-power interface are:

- Transaction activity status to an external low-power controller (LPC) for enabling or disabling the clock.
- Configurable maximum number of clock cycles the system must remain idle before entering low-power mode.

1.3.5 Unsupported Features

The following features are not supported:

- Configurable write interleave fan-out
- Logic to handle lock sequences for data-width-altering configurations
- Different burst length widths on each port; for example, burst length width of 4 on the master port and 8 on the slave port

1.4 Standards Compliance

The DW_axi_x2x conforms to the *AMBA AXI and ACE Protocol Specification* from ARM. Readers are assumed to be familiar with these specifications.

1.5 Verification Environment Overview

The DW_axi_x2x includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation through the simulation waveforms to see how the DW_axi_x2x performs under the provided test conditions.

The DW_axi_x2x testbench environment is described in "Verification" on page 89.

1.6 Where To Go From Here

At this point, you may want to get started working with the DW_axi_x2x component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, refer to *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_axi_x2x component, refer to *"Overview of the coreConsultant Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_axi_x2x component within a DesignWare subsystem using coreAssembler, refer to *"Overview of the coreAssembler Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

The DW_axi_x2x connects an AXI master and an AXI slave. Most applications connect the DW_axi_x2x AXI master port interface to the DW_axi interconnect fabric for use in an AXI subsystem. The DW_axi_x2x can be used on any point-to-point (master to slave) AXI connection, such as from a master to an interconnect.

The DW_axi_x2x connects to AXI masters through a Master Port block—DW_axi_x2x_mp—and it connects to AXI slaves through a Slave Port block—DW_axi_x2x_sp. The DW_axi_x2x retains all transaction attributes of an AXI master or AXI slave that pass through it, unless there is a need for resizing, endian mapping, or RESP/exclusive access signal alteration.

2.1 DW_axi_x2x Application Examples

The DW_axi_x2x can connect:

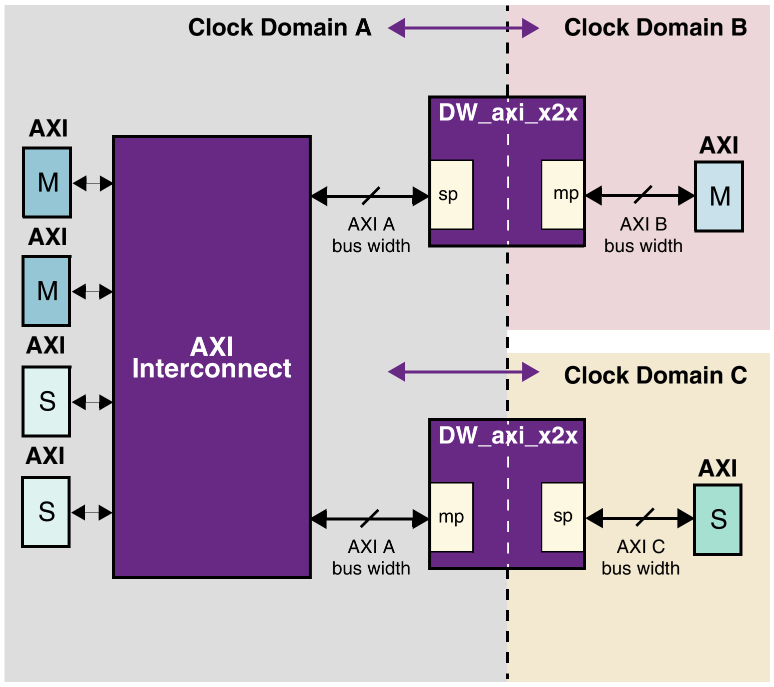
- An AXI master to an AXI slave
- An AXI master to an AXI subsystem
- An AXI subsystem to an AXI slave
- Incompatible AXI interconnect subsystems, which can occur because of different data widths, clock frequencies, or endianness

The following sections show some system overview examples of DW_axi_x2x configurations.

2.1.1 Incompatible AXI Master and Slave in Interconnect Subsystem

Figure 2-1 illustrates a system overview using the DW_axi_x2x as a connection between incompatible AXI masters and slaves in an interconnect subsystem.

Figure 2-1 DW_axi_x2x Connecting Incompatible Masters and Slaves

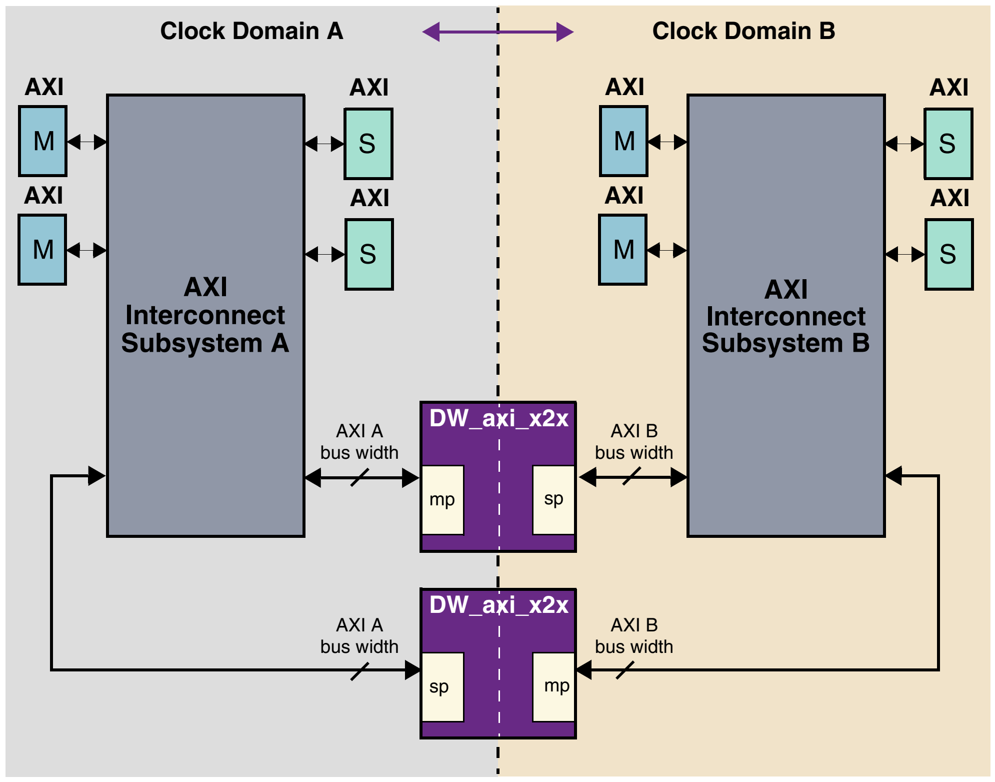


M = Master S = Slave

2.1.2 Incompatible AXI Interconnect Subsystems

Figure 2-2 illustrates a system overview where the DW_axi_x2x acts as a connection between two incompatible AXI interconnect subsystems.

Figure 2-2 DW_axi_x2x Connecting Two Incompatible AXI Subsystems



M = Master S = Slave

2.2 Channels

The DW_axi_x2x uses read and write address channels, read and write data channels, and response channels to communicate transactions between AXI masters and AXI slaves.

2.2.1 Channel Buffers

Transfers on every channel are buffered from channel source to channel sink. The size of the buffer for each channel is determined by the bus on the Master Port side so that the buffer can accept or issue a transfer to the Master Port bus in a single cycle. These conditions apply:

- Transfers are accepted from the source as soon as there is space in the channel buffer
- Transfers are issued to the sink as soon as the channel buffer is not empty

You can use coreConsultant to configure the buffer depth for each channel.

The minimum value of the buffer depth for each channel is 2.

In order to avoid a decrease in channel throughput due to the latency through the channel FIFO, the depth of each channel FIFO should not be set lower than the value given by the following equation:

Min_buf_depth = (2 + (X2X_CLK_MODE ? (X2X_[SP/MP]_SYNC_DEPTH + 2) : 0)

Note The above equation assumes that the aclk_m frequency is equal to the aclk_s frequency. If the frequency of the sink clock—that is, aclk_s for AR, AW and W channels; aclk_m for R and B channels—is slower than the frequency of the source clock, a larger buffer depth is recommended due to the longer time taken to pop data from the channel FIFO.

2.2.2 Address Channels

DW_axi_x2x address channels forward AXI commands from the primary bus to the secondary bus. Transfers are accepted from an AXI master by the DW_axi_x2x Master Port and issued to an AXI slave from the DW_axi_x2x Slave Port.

Address widths on both the Slave Port and Master Port are individually configurable to any value from 32 to 64. However, the effective address within the DW_axi_x2x is limited to the smaller of the two interface address widths; unused bits are ignored or tied to 0.

- If the Master Port address width is greater than the Slave Port address width, the upper bits of the DW_axi_x2x Master Port address is ignored.
- IF the Slave Port address width is greater than the Master Port address width, the upper bits of the DW_axi_x2x Slave Port address is tied to 0.

The ID width on the DW_axi_x2x Slave Port is inherited from the DW_axi_x2x Master Port; that is, X2X_SP_IDW is inherited from X2X_MP_IDW.

If resizing is not required, then a transaction is passed between the Slave Port and Master Port unaltered. For more information about resizing, refer to "Transaction Resizing" on page 23.

For configurations that can alter a transaction, the DW_axi_x2x uses several parameters to store information for each active transaction:

- X2X_MAX_URIDA – Defines the number of active unique read transaction IDs
- X2X_MAX_UWIDA – Defines the number of active unique write transaction IDs
- X2X_MAX_RCA_ID – Defines the number of active read transactions per unique ID
- X2X_MAX_WCA_ID – Defines the number of active write transactions per unique ID

These parameters define how many outstanding data and response channel transactions for which the DW_axi_x2x has to store information, which in turn tells the other channels how to handle the various parts of the transaction.

The Slave Port address channel stalls if the limit is reached for:

- Active write transactions per the X2X_MAX_WCA_ID parameter
- Active read transactions per the X2X_MAX_RCA_ID parameter
- Unique write IDs per the X2X_MAX_UWIDA parameter
- Unique read IDs per the X2X_MAX_URIDA parameter

The channel unstalls when a transaction completes, which allows the stalled transaction to progress.

For configurations that cannot alter transactions, there is no limit to the number of active unique IDs or the number of active transactions per unique ID.

2.2.2.1 Address Channel Transactions

The following sections discuss details about transactions and how their size, length, and access attributes are changed.

2.2.2.1.1 Transaction Resizing

Resizing a transaction is any operation where the DW_axi_x2x changes the burst length or burst size of a transaction or breaks a transaction into multiple smaller transactions. In general, transaction resizing occurs as follows:

- Downsizing – Occurs when the size of X2X_MP_DW parameter is greater than size of the X2X_SP_DW parameter.
- Upsizing – Occurs when *both* of the following are true:
  - Size of X2X_MP_DW parameter is less than size of X2X_SP_DW parameter
  - When X2X_HAS_TX_UPSIZE parameter is set to True (1)

2.2.2.1.2 Extended Burst Mode

You can configure the burst length width of the master and slave ports of DW_axi_x2x to between 4 and 8. However, you cannot configure different burst length widths on the master ports and slave ports. You can configure only X2X_MP_BLW (master port burst length width) in coreConsultant/coreAssembler; the X2X_SP_BLW parameters automatically inherits its value based on X2X_MP_BLW.

Note

- For all values of X2X_MP_BLW, DW_axi_x2x does not support transactions that cross 4KB boundaries. This means that transfers of total size greater than 4KB are also unsupported.
- As defined by the AMBA AXI protocol, transactions of burst type WRAP must not use a burst length greater than 16; this applies for all values of X2X_MP_BLW.

2.2.2.1.3 Transaction Downsizing

The DW_axi_x2x can change the attributes of a transaction that passes through so that the transaction fits it on a smaller bus. Examples of when this can happen are:

- When the Master Port receives a transaction with a burst size greater than the data width of the Slave Port.
- When the Master Port receives a transaction requesting an amount of data that the Slave Port cannot satisfy with a single transaction.

The following describe rules that determine DW_axi_x2x transaction downsizing:

- Transactions with unaligned addresses can change the burst length of downsized transactions issued by the DW_axi_x2x. In this case, the DW_axi_x2x reduces the burst length of transactions it issues, depending on how many bytes the unaligned address determines should be ignored. For example, consider this scenario:
  - Downsizing a write from a 32-bit to 16-bit data bus
  - Burst size = 32 bits
  - Burst length = 1
  - Start address = 'h2

The result of the downsizing is a burst size of 16 and a burst length of 1, not a burst length of 2. This occurs because the start address of 'h2 determines that the lower 16 bits should be ignored.

In the case where the number of bytes ignored due to an unaligned address is greater than the maximum number of bytes in a Slave Port transaction, there is an additional latency before the DW_axi_x2x issues the transaction. The extra latency in clock cycles in this case is the floor value of the number of bytes ignored due to unalignment divided by the maximum number of bytes in a Slave Port transaction.

- The DW_axi_x2x can increase or decrease the burst length attribute of transactions that pass through it. The DW_axi_x2x always uses the maximum possible burst length on Slave Port transactions in order to complete a transaction from the Master Port with the fewest possible transactions.
- If the DW_axi_x2x cannot satisfy a transaction request to the Master Port with a single transaction on the Slave Port, it breaks up the transaction into multiple smaller transactions.

2.2.2.1.4 Transaction Upsizing

You can control transaction upsizing using the X2X_HAS_TX_UPSIZE parameter, but only when the value of the X2X_MP_DW parameter is less than the X2X_SP_DW value. By default the DW_axi_x2x attempts to upsize a transaction from the master of any A[R|W]SIZE value. However, you can set the X2X_UPSIZE_ANY_ASIZE parameter to false in order to force the DW_axi_x2x to attempt to upsize only those transactions with a maximum A[R|W]SIZE value for the primary bus data width.

Transaction upsizing takes transactions from a smaller data width primary bus and alters their burst size to the maximum allowable on the larger data width secondary bus. The burst length of the transaction becomes smaller as a result of this operation.

In order to ensure that the DW_axi_x2x can create only upsized beats of the maximum Slave Port data width burst size, transaction upsizing is restricted to the following rules; if the rules are not met to perform upsizing, then the transaction is passed as is.

- Transaction must be completed in an integer number of beats of the Slave Port data width size, which is determined by:

  $$\frac{BurstSize \times BurstLength}{X2X\_SP\_DW}$$

  For example, consider the following transaction:

  - X2X_MP_DW = 32
  - X2X_SP_DW = 128
  - Burst size = 32
  - Burst length = 4

  The calculation becomes:

  $$\frac{32 \times 4}{128} = 1$$

  Because the number of beats is an integer, the transaction can be upsized to a transaction of burst size = 128 and burst length = 1.

  However, the following transaction cannot be upsized:

  - X2X_MP_DW = 32
  - X2X_SP_DW = 128

- ❑ Burst size = 32
- ❑ Burst length = 6

The calculation becomes:

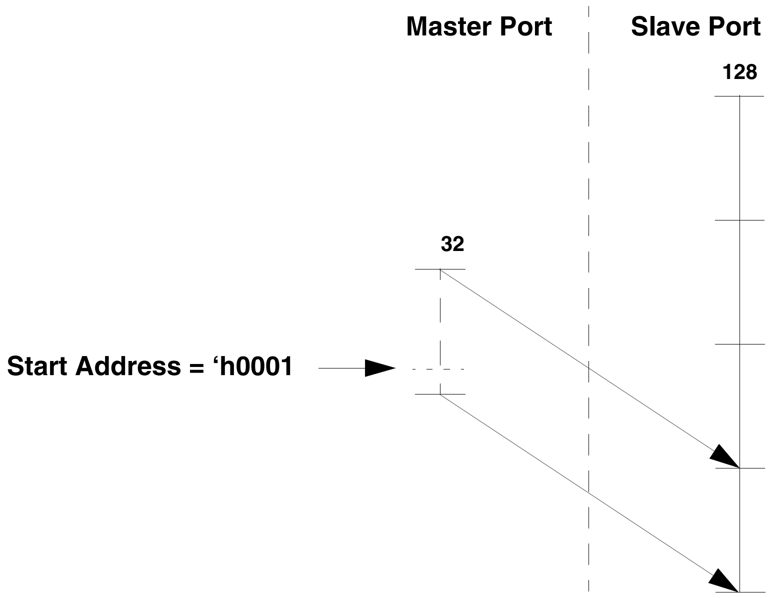
$$\frac{32 \times 6}{128} = 1.5$$

Because the result is not an integer, the DW_axi_x2x cannot upsize the transaction.

- The DW_axi_x2x upsizes only those transactions where the Master Port A[R|W]SIZE is aligned to the Slave Port data width. Because of this, the address from the Master may be unaligned with respect to A[R|W]SIZE, but the first Master Port beat must align to the start of the Slave Port data bus.

  Figure 2-3 illustrates an example of an upsized alignment. The external master sends a start address of 'h0001, but the DW_axi_x2x can still upsize. This start address is not aligned to 128, but it maps the first beat of 32-bit Master Port data to the start of the Slave Port 128-bit data bus.

Figure 2-3 Upsizing Alignment Rule



2.2.2.1.5 Transaction Types

The DW_axi_x2x takes several types of action for transactions of each burst type where a transaction size exceeds the DW_axi_x2x Slave Port data bus.

- INCR – The DW_axi_x2x decodes the pending transaction, uses the length and size to determine the total number of beats required to satisfy the request, and then initiates the appropriate transaction or transactions to satisfy the request.

  The DW_axi_x2x does not support INCR transactions that cross a pre-defined address boundary, for which the default is 4k. If a transaction of a total size greater than 4k is possible at the Master Port in a given configuration, then the DW_axi_x2x supports an address boundary defined by the total transaction size.

- WRAP – For configurations with data width or burst length width adaptation, the DW_axi_x2x breaks all WRAP transactions into a series of INCR transactions controlled to observe the address sequencing dictated by the wrap; otherwise the WRAP transaction is passed unaltered.

  Logic to handle translation of WRAP transactions in data-width-altering configurations may be removed by setting the parameter X2X_HAS_WRAP_BURST = 0; removing this logic will significantly improve the operating frequency of the DW_axi_x2x. If X2X_HAS_WRAP_BURST = 0, then wrapping bursts must not be sent to the DW_axi_x2x.

- FIXED – Transactions of this type are passed unaltered when burst size values are less than or equal to the Slave Port maximum burst size.

  - If the burst size is larger than the maximum Slave Port burst size, then the DW_axi_x2x issues one or more FIXED transactions of the maximum burst size of the Slave Port with the same address. This results in different behavior than a FIXED burst of the original burst size because the DW_axi_x2x assumes that the user will not issue FIXED bursts of a burst size larger than the data widths on a particular system. This allows the master on a larger data width system to build up beats for a FIXED burst of the Slave Port data width size in its larger data bus, which the DW_axi_x2x then issues with the smaller Slave Port size.

  - FIXED transactions can be upsized. If X2X_HAS_TX_UPSIZE is true and the transaction obeys all the upsizing rules, then the DW_axi_x2x issues an upsized FIXED burst. This results in different behavior than if the FIXED burst was issued with the original smaller A[R|W]SIZE.

    Take the following configuration as an example:

    - X2X_MP_DW=16
    - X2X_SP_DW=32
    - X2X_HAS_TX_UPSIZE is true

    If a write transaction occurs with the following:

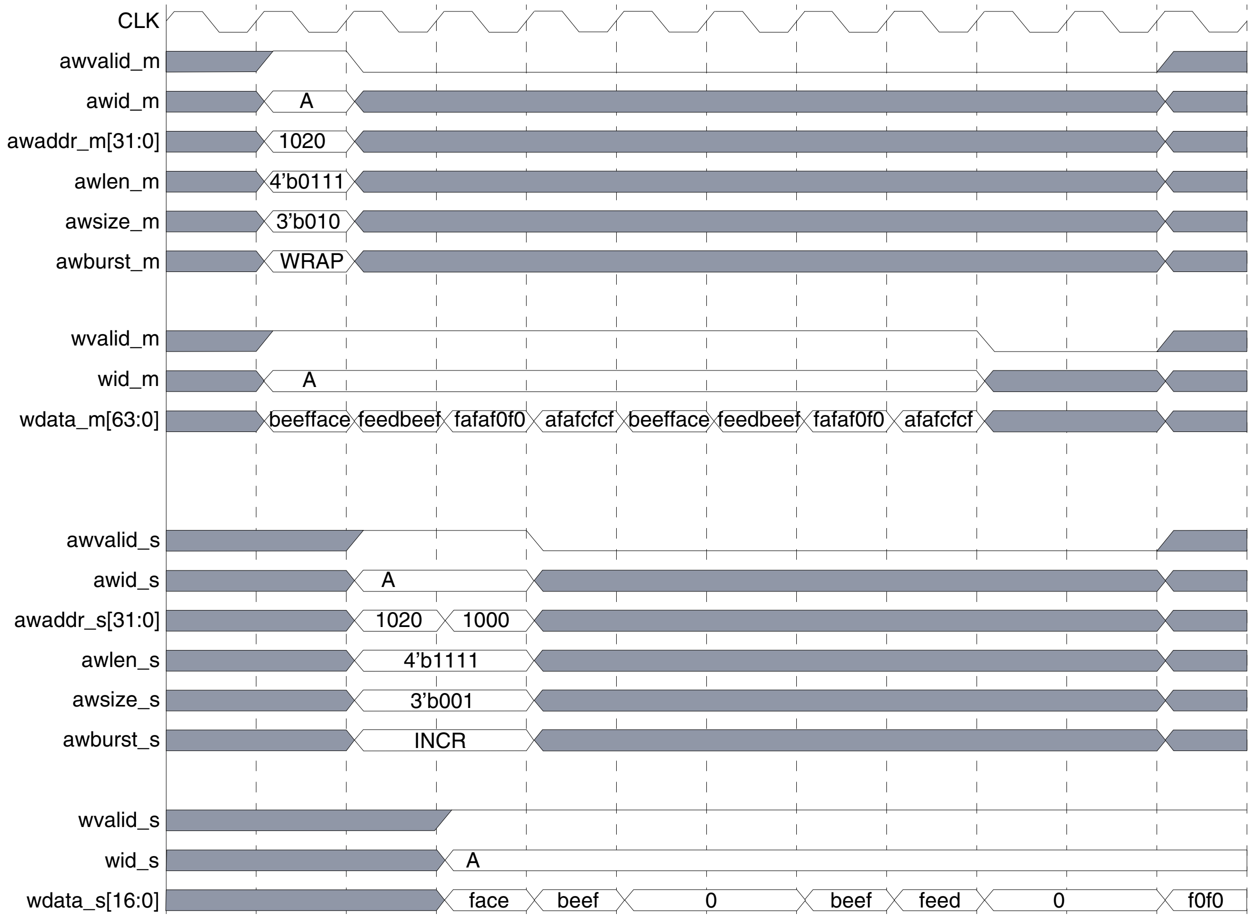
    - awsize=1 (16 bits)
    - awburst=1
    - awlen=4

    Assuming other upsizing rules are met, the DW_axi_x2x issues this transaction with awsize=2 (32 bits), awburst=1, awlen=2. In this case, two beats from the Master Port make up a single beat to the Slave Port. Because of FIXED burst addressing, the two beats from the Master Port have addresses 0x0 and 0x0. However, when the same two beats are packed into a 32-bit bus and issued as a FIXED transaction of awsize=2, the two 16-bit chunks of data from the Master Port the map to the addresses 0x0 and 0x2, respectively.

  - FIXED transactions that are broken into multiple transactions have the same address value.

Figure 2-4 illustrates an example of a write WRAP transaction downsized to multiple INCR transactions.

Figure 2-4 Write WRAP Transaction Downsized to Multiple INCR Transactions



The WRAP write transaction has a burst size of 4 bytes and burst length of 16 bytes, which are downsized for a 16-bit Slave Port data bus. With a maximum length of 16 and a 16-bit data bus on the Slave Port, the DW_axi_x2x cannot satisfy the request with a single transaction; it has to separate the transaction into multiple INCR transactions. The wrap boundary for this transaction is aligned to (SIZE * LEN) = 64 bytes or 'h40 and because the start address for the transaction is 'h1020 the DW_axi_x2x has to break the transaction into two INCR transactions of LEN=16 and SIZE=2 bytes with start addresses of 'h1020 and 'h1000 respectively, to satisfy the WRAP addressing. Note that for the purposes of this example, the upper 32 bits of the master port write data beats are 0, which is reflected in the downsized beats issued from the slave port.

Note In all timing diagrams in this document, all written values on signals are assumed to be in hex format unless otherwise stated.

2.2.2.1.6 Exclusive Accesses

An exclusive access that is downsized to multiple smaller transactions cannot function as intended by the AXI protocol specification.

Exclusive accesses that do not need to be broken into multiple transactions are allowed to pass through the DW_axi_x2x and therefore function as expected.

Note Exclusive writes are not supported when the Master port data width is not equal to the Slave port data width. This is true despite the fact that the transaction not being downsized.

2.2.2.1.7 Locking Sequences

Currently the DW_axi_x2x does not support translation of locking sequences for data-width-altering configurations.

2.2.3 Data Channels

Data bus widths are configurable to 8, 16, 32, 64, 128, 256, and 512 bits. When a configuration does not alter transactions, the DW_axi_x2x does the following:

- Forwards data channel transfers (read/write) unaltered
- Forwards early write data—that is, data that arrives before the corresponding command—from the Slave Port to the external slave

2.2.3.1 Write Data Unpacking/Packing

For write data transactions where the burst size is greater than the data bus size on the Slave Port, the DW_axi_x2x address channels downsize the transaction. Thus on the write data channel, the DW_axi_x2x unpacks incoming data into smaller transfers of the downsized transactions burst size.

Figure 2-4 illustrates write data transactions being unpacked. The Master Port receives write data transactions that have a burst size of 8 bytes, and the DW_axi_x2x has to unpack these write data transfers in order to match the burst size of the downsized write transactions that the DW_axi_x2x decodes and issues.

For upsizing DW_axi_x2x configurations, the address channel may have generated a transaction with a burst size larger than that issued by the external master. In these situations, the DW_axi_x2x write data channel has to pack data up to the larger size before issuing from the Slave Port. This requires a packing register equal in width to the value of the X2X_SP_DW parameter for each interleaving depth supported, indicated by the X2X_WID parameter.

Write data transfers that need to be unpacked do not incur any cycle throughput or latency penalty. Write data for an upsized transaction has additional latency, depending on the upsize ratio; that is, the DW_axi_x2x has to wait to pack smaller beats up to the larger beat size.

2.2.3.2 Read Data Unpacking/Packing

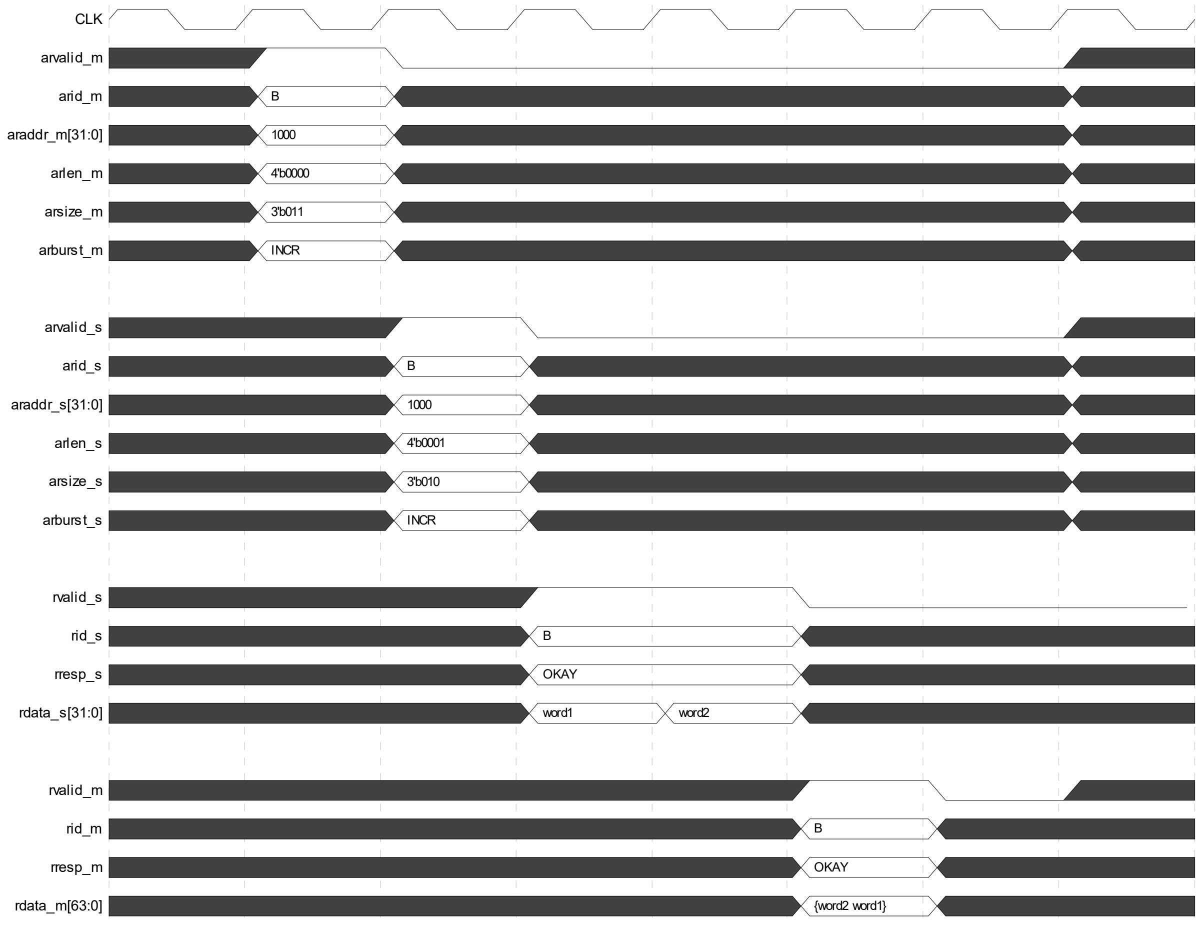
When the DW_axi_x2x downsizes read transactions to fit on the secondary bus, it packs the downsized transactions up to the burst size requested by the original transaction before pushing the requested data into the read data buffer. This makes the downsizing invisible to the external master.

When the DW_axi_x2x packs read data from a number of downsized read beats up to the requested burst size, if the response field of any of the downsized read beats indicates an error condition, the read beat that is issued to the external master contains an error condition. When collecting and merging the responses of downsized read beats, the order of priority of response types goes from high to low—that is, DECERR, SLVERR, OKAY, EXOKAY. For example, if three of the response fields of four downsized read beats indicate SLVERR and the other field indicates DECERR, the read data transfer issued to the external master indicates DECERR.

Note An ERROR response decoded from multiple downsized transfers applies to only the single beat of the requested burst to which those downsized transfers relate,; that is, none of the other beats of the burst.

Figure 2-5 illustrates an example of read data packing.

Figure 2-5 Read Data Packing



This shows a 64 bit single read transaction which has been downsized by the DW_axi_x2x to a transaction for a 32-bit data bus with a burst length of 2 bytes and a burst size of 4 bytes. The DW_axi_x2x packs the two read data transfers from the external slave back up to the original requested burst size before sending the transaction to the external master.

When DW_axi_x2x configurations are upsized, the address channel may have generated a transaction of a burst size larger than that issued by the external master. In these cases, the DW_axi_x2x read data channel must unpack transactions of the larger burst size before issuing from the Master Port. While this incurs no additional latency for when the first beat of read data reaches the external master, there is some stalling of the read data channel at the Slave Port if the read data beats arrive within a certain time period of each other.

To mitigate this effect, there is a one-stage buffer at the DW_axi_x2x Slave Port Read Data channel. This buffer can accept one data beat of the maximum Slave Port size in one cycle, but subsequent beats of that SIZE have to wait for a number of cycles while the previous beat is pushed into the channel FIFO. In other words, once an upsized data beat reaches the DW_axi_x2x Slave Port, this channel cannot accept another beat of read data for ((X2X_SP_DW/X2X_MP_DW)–1) clock cycles—where -1 accounts for the clock cycle saved by the single-stage buffer.

2.2.3.3 Read Data Interleaving and Ordering

The DW_axi_x2x allows an external slave to interleave and reorder read data. The X2X_MAX_URIDA parameter sets the number of active unique read IDs within the DW_axi_x2x; this also sets the read interleaving and read reordering depths.

For configurations where downsizing or endian mapping are required, the DW_axi_x2x must maintain transaction information and a read data packing register for each outstanding transaction up to the limits specified by the X2X_MAX_URIDA and X2X_MAX_RCA_ID parameters.

2.2.3.4 Write Data Interleaving

The DW_axi_x2x allows an external master to interleave its write data to a specified depth. The supported write interleaving depth is set by the X2X_WID parameter.

Note If the DW_axi_x2x is configured with X2X_WID > 1, you should ensure that the slave connected to the Slave Port has a burst response re-order depth greater than or equal to X2X_WID. Breaking this rule could result in deadlock conditions between the slave device and the X2X.

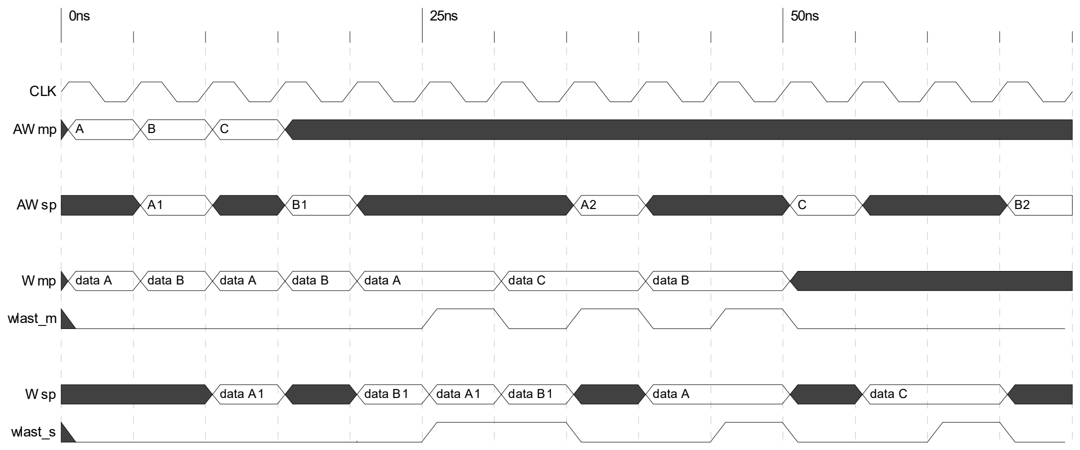
In configurations where it is possible for the DW_axi_x2x to issue multiple secondary transactions from a single primary transaction and where X2X_WID is greater than 1, the DW_axi_x2x may have to stall transactions on the write address. This is due to the fact that an external master follows the write ordering rules with respect to transactions it issues, but the DW_axi_x2x may need to issue more than one transaction from each of those transactions and must ensure that the ordering rules are followed with respect to the new transactions.

The following describes transaction stalling:

1. When the DW_axi_x2x receives a transaction, it checks for stalled transactions pending; if there are none, the DW_axi_x2x forwards the first transaction resulting from that transaction. Subsequent transactions are not forwarded until the DW_axi_x2x receives a beat of data for that transaction. The Write Data channel drives the address channels so that the write ordering rules can be maintained.
2. The operation adds one cycle of latency whenever a write data beat has to inform the DW_axi_x2x to issue a write transaction.
3. Because transactions with the same ID cannot be interleaved, if the DW_axi_x2x receives a transaction and all stalled transactions have the same ID, the stalled transactions are immediately issued, and the new transaction is stalled.

Figure 2-6 illustrates an example of the situation described below with X2X_WID equal to 2.

Figure 2-6 Write Interleaving/Primary Transactions to Multiple Secondary Transactions



The following events are illustrated in Figure 2-6:

1. The master issues three transactions: A, B, and C.
2. With no stalled transactions, A1 is immediately forwarded.
3. A2 is stored and must wait; transaction B must also wait.
4. At some point, transaction B data is then received, which causes the DW_axi_x2x to issue B1.
5. B2 is stored at this point.
6. At some later point when all the data for A1 has been issued, A2 data is received, which causes the DW_axi_x2x to issue the command for A2.
7. Transaction C data is received, which is forced to wait due to the other stalled transactions.
8. The DW_axi_x2x issues the command for C.

The following describe specific configuration situations:

- Where downsizing or endian mapping is required in DW_axi_x2x configurations, the DW_axi_x2x keeps transaction information for the number of transactions specified by X2X_WID. The DW_axi_x2x can never forward write data to an external slave before receiving the corresponding transaction for the write data, since it needs the transaction information to know how to perform the write data beats.

  In configurations where downsizing and endian mapping are not required, then the DW_axi_x2x can forward write data from the Slave Port before forwarding the corresponding transaction.

- For upsizing configurations—where X2X_HAS_TX_UPSIZE is equal to 1 and X2X_WID is greater than 1—if the DW_axi_x2x generates an upsized transaction, it does not issue the transaction until it has the first upsized beat ready to issue from the Slave Port write data channel.

  For example, the following conditions could exist:

  - X2X_MP_DW is equal to 8
  - X2X_SP_DW is equal to 32
  - A write transaction with awsize=0 is upsized to a transaction with awsize=2

In this case, the DW_axi_x2x does not issue the upsized transaction until it has received at least four beats of awsize=0, in order to pack the first 32-bit beat. If the DW_axi_x2x is already issuing a different command on the write address channel at this time, then the write data channel waits until the address channel is free before issuing the command and first beat of data for the upsized transaction at the same time.

2.2.3.5 Endian Transformation

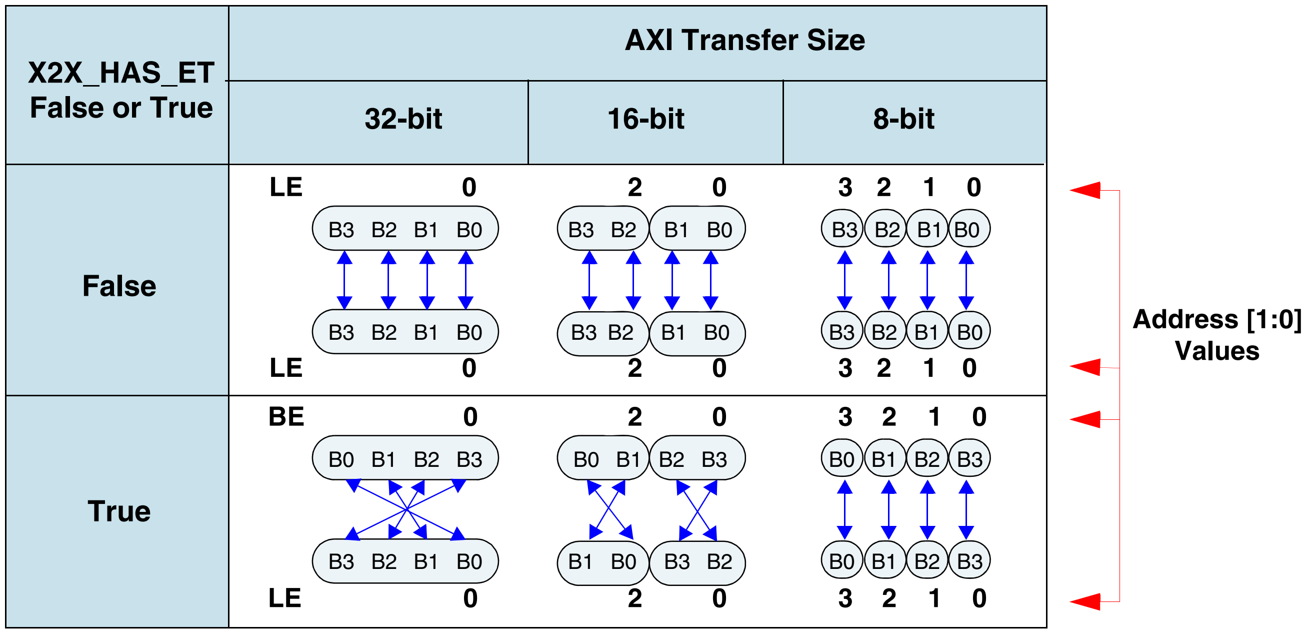
If the X2X_HAS_ET parameter is set to true, the DW_axi_x2x performs a byte-invariant endianness transformation on all data and strobes that pass through it.

Endian mapping in the DW_axi_x2x is performed with respect to the A[R|W]SIZE received at the Master Port; that is, if the DW_axi_x2x is downsizing a 64-bit beat to two 32-bit beats, it endian maps the 64-bit beat with respect to A[R|W]SIZE=8 bytes and then splits the mapped data into two beats.

Note Unaligned start addresses do not work through a byte-invariant endian transformation. Because of this, the DW_axi_x2x does not support transactions with unaligned start addresses when X2X_HAS_ET is set to true.

Figure 2-7 shows an LE--LE and a BE-LE transform. In general, you should set X2X_HAS_ET to true only if there is a change in endianness on either side of the DW_axi_x2x; that is, if both sides are big-endian, X2X_HAS_ET should be set to 0. A byte-invariant endian transform is the same from LE to BE as from BE to LE.

Figure 2-7 Endian Definitions

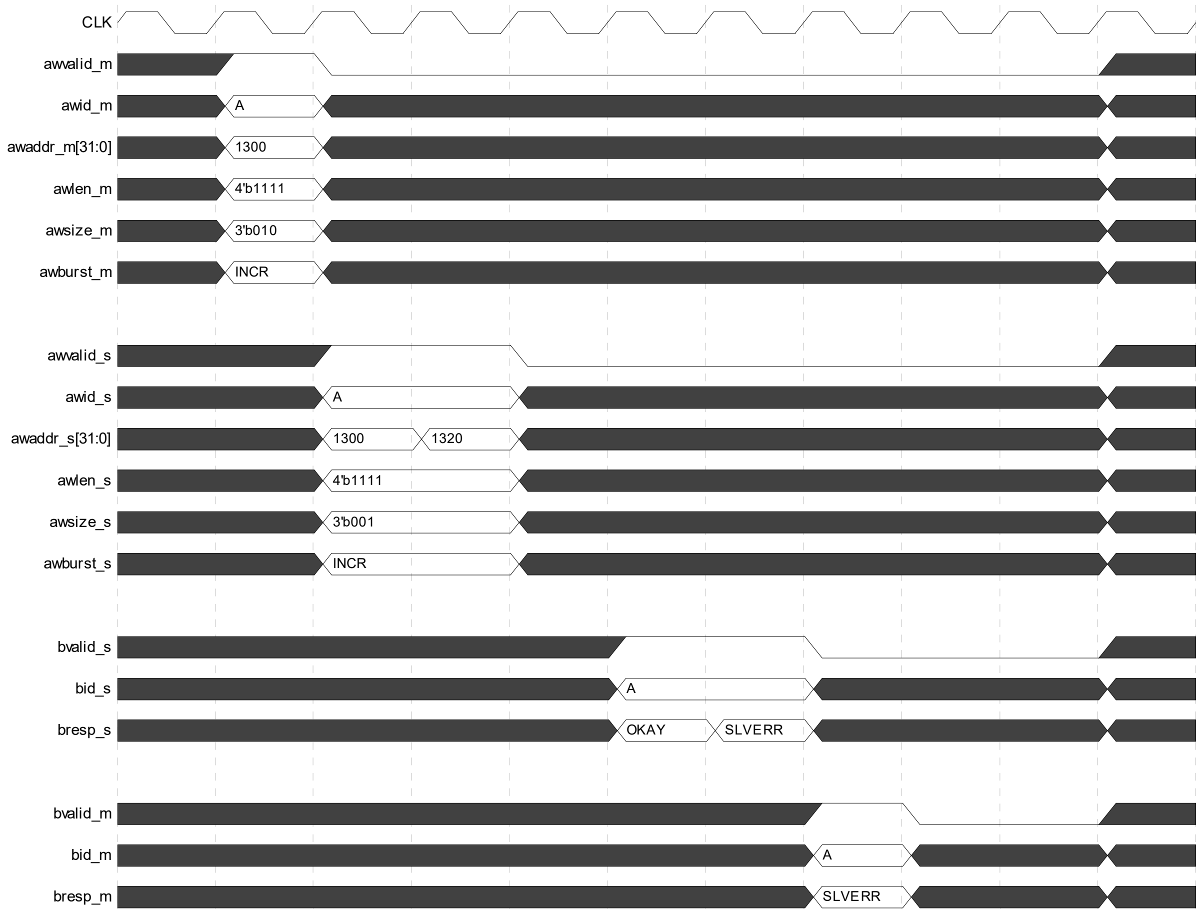


2.2.4 Burst Response Channel

For writes downsized to multiple transactions, there will be a number of burst responses. The DW_axi_x2x merges these responses to issue one burst response transfer to the burst response buffer for the original pre-downsized write. The priority of response types when merging is the same as read responses, as detailed in "Read Data Unpacking/Packing" on page 28.

There is no cycle throughput or latency difference between burst responses that do not need to be merged before issuing to the external master and those that do require merging.

Figure 2-8 Burst Response Merging Example



2.3 Non-Protocol Signals

The following sections discuss topics relating to signals.

2.3.1 Sideband Signals

The DW_axi_x2x provides sideband signals for each channel that enable users to transfer extra information outside of the AXI protocol specification. The existence and width of sideband buses are individually configurable per channel through the X2X_HAS_*SB and AXI_*_SBW parameters.

The widths of the sideband buses are the same on the Slave Port and Master Port sides, and the sideband bus contents are pushed into the channel buffers along with the rest of the channel data and control information.

For transfers downsized into multiple transfers, the downsized transfers have the same sideband signal content as the pre-downsized transfer; that is, for two beats from the Master Port with sideband contents A and F, if each one is downsized to two beats, the result is that “A A F F” is received from the Slave Port.

For transfers packed to make a single transfer, the DW_axi_x2x forwards the sideband contents of the last transfer used to create the packed transfer.

No endian mapping is performed on the sideband signals as they pass through the DW_axi_x2x.

2.3.2 Trustzone Support

The DW_axi_x2x allows an external slave to forward a trustzone support signal through the DW_axi_x2x to the primary bus side. Whether or not the signal is present is controlled by the X2X_HAS_TZ_SUPPORT parameter. For DW_axi_x2x configurations with different clock domains on the Master Port and Slave Port sides, the trustzone support signal is synchronized from Master Port to Slave Port.

2.4 Low-Power Interface

You can configure the DW_axi_x2x to include an AXI low-power handshaking interface. This interface allows the following:

- DW_axi_x2x informs the system low-power controller (LPC) when it has no outstanding transactions
- LPC requests the DW_axi_x2x to enter into a low-power state

You can include this low-power interface in your design by setting the X2X_LOWPWR_HS_IF parameter to 1.

The low power handshaking interface includes the following signals:

- csysreq input – de-asserted by the LPC to initiate a low-power state; asserted by LPC to initiate an exit from a low-power state
- csysack output – de-asserted by DW_axi_x2x to acknowledge a request to enter a low-power state; asserted by DW_axi_x2x to acknowledge a request to exit a low-power state
- cactive output – de-asserted by DW_axi_x2x when the clock can be removed after the next positive edge; csysack must also be deasserted

Note For configurations that alter transactions, there is a limit to the number of outstanding read/write transactions because of the need to store resizing information; there is no such limit for configurations that do not alter transactions.

Because of this, you should set a transaction limit that is greater than the depth of the relevant address channel FIFO.

The sequence of events for entering a low-power state is as follows:

1. The LPC requests the DW_axi_x2x to enter a low-power state by de-asserting the csysreq signal.
2. Since the DW_axi_x2x does not have a power-up or power-down sequence, it always acknowledges a csysreq signal on the next cycle by asserting or de-asserting the csysack signal.

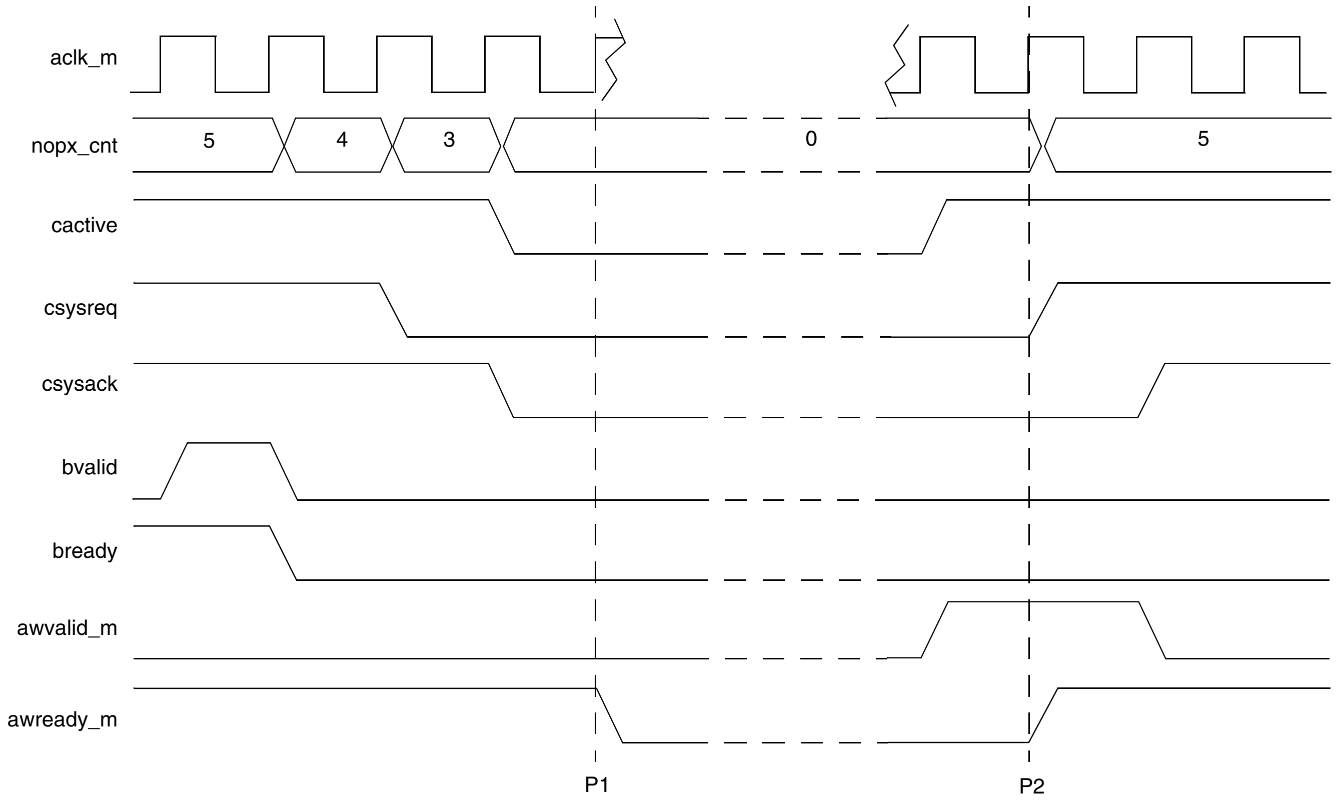
   When requested by the LPC, the low-power state depends on the value of the cactive signal at the time that the csysack signal is sampled by the LPC after the csysreq signal has been de-asserted.

3. The cactive signal de-asserts when the DW_axi_x2x has no outstanding transactions. If you set the X2X_LOWPWR_NOPX_CNT parameter to "X" in coreConsultant, the cactive signal de-asserts after *X* cycles, during which there were no outstanding transactions, have elapsed. Note that transaction completion on DW_axi_x2x will be detected 1 cycle after the last transaction completes. Thus DW_axi_x2x will start counting down just one cycle after that.
4. DW_axi_x2x enters a low-power state when *all* of the cactive, csysreq, and csysack signals are low (0) when sampled at the positive edge of aclk_m; this is illustrated at P1 in Figure 2-9.

Note Entry to a low-power state happens at any cycle in which the above condition is satisfied.

While in a low-power state, the awready_m, wready_m, and arready_m signals are held low, which prevents any new transaction from starting and thus allows the clock to be safely disabled; this is illustrated between points P1 and P2 of Figure 2-9.

Figure 2-9 Low-Power State Entry and Exit (X2X_LOWPWR_NOPX_CNT = 5)



The DW_axi_x2x exits a low-power state when the cactive signal goes high and is sampled at the positive edge of aclk_m; illustrated at P2 of Figure 2-9.

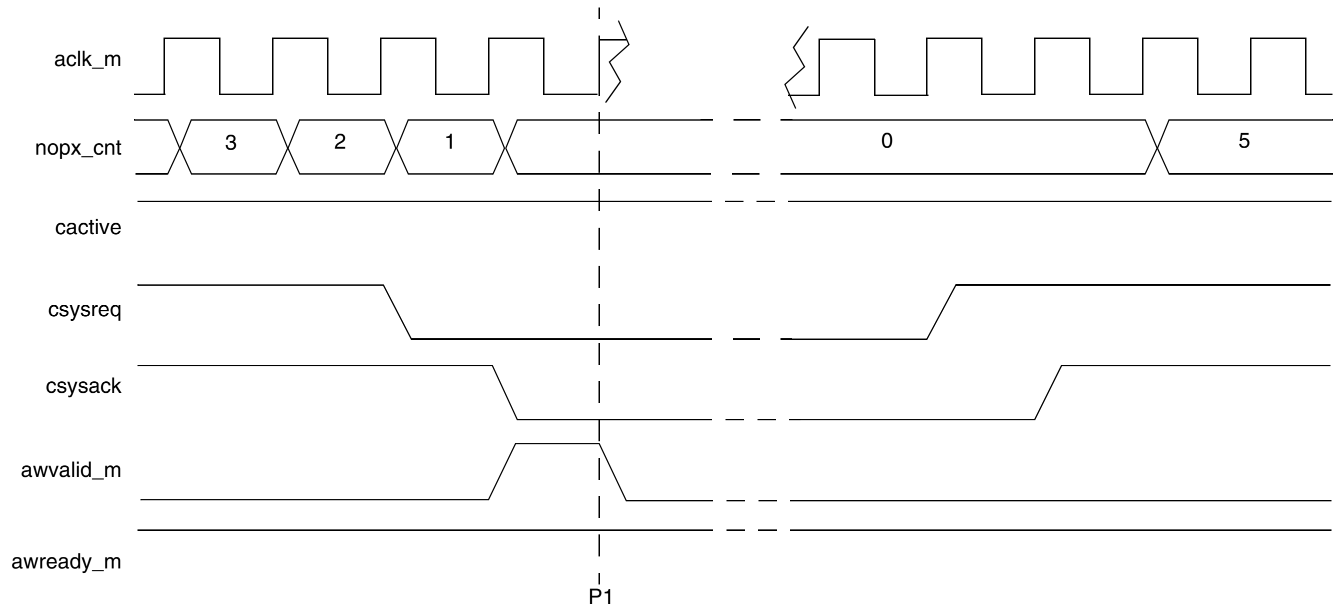
An exit from low-power can be initiated by:

- LPC asserting the csysreq signal, which causes the DW_axi_x2x to assert the cactive signal, illustrated in Figure 2-11.

- A master asserting the awvalid or arvalid signals of the DW_axi_x2x, which causes the DW_axi_x2x to assert the cactive signal, illustrated in Figure 2-9.

Figure 2-10 shows the DW_axi_x2x rejecting a request to enter a low-power state. At P1, the no-pending-transaction (nopx_cnt) counter has reached 0, but on this same cycle the awvalid_m signal asserts, which keeps the cactive signal asserted. The LPC samples at P1 that the DW_axi_x2x has rejected the entry to low-power mode, and therefore must not remove the clock.

Figure 2-10 DW_axi_x2x Low-Power Interface Rejects Low-Power Entry



2.4.1 cactive Signal De-assertion

The cactive signal de-asserts the X2X_LOWPWR_NOPX_CNT parameter *aclk_m* cycles after the last pending transaction completes. If the csysreq signal de-asserts while the component is counting down to 0 from X2X_LOWPWR_NOPX_CNT, the cactive signal will de-assert on the next cycle.

After the positive edge of the aclk_m signal where the cactive signal and the csysack signal are sampled low, the low-power controller (LPC) may remove the clock(s) from the DW_axi_x2x (P1 in Figure 2-9 on page 35). At this point the bready and awready_m signals will also be driven low.

2.4.2 cactive Signal Assertion

The cactive signal will assert combinatorially when the awvalid_m or arvalid_m signals are asserted, at which point it will then remain asserted until all outstanding transactions have completed and X2X_LOWPWR_NOPX_CNT cycles have elapsed.

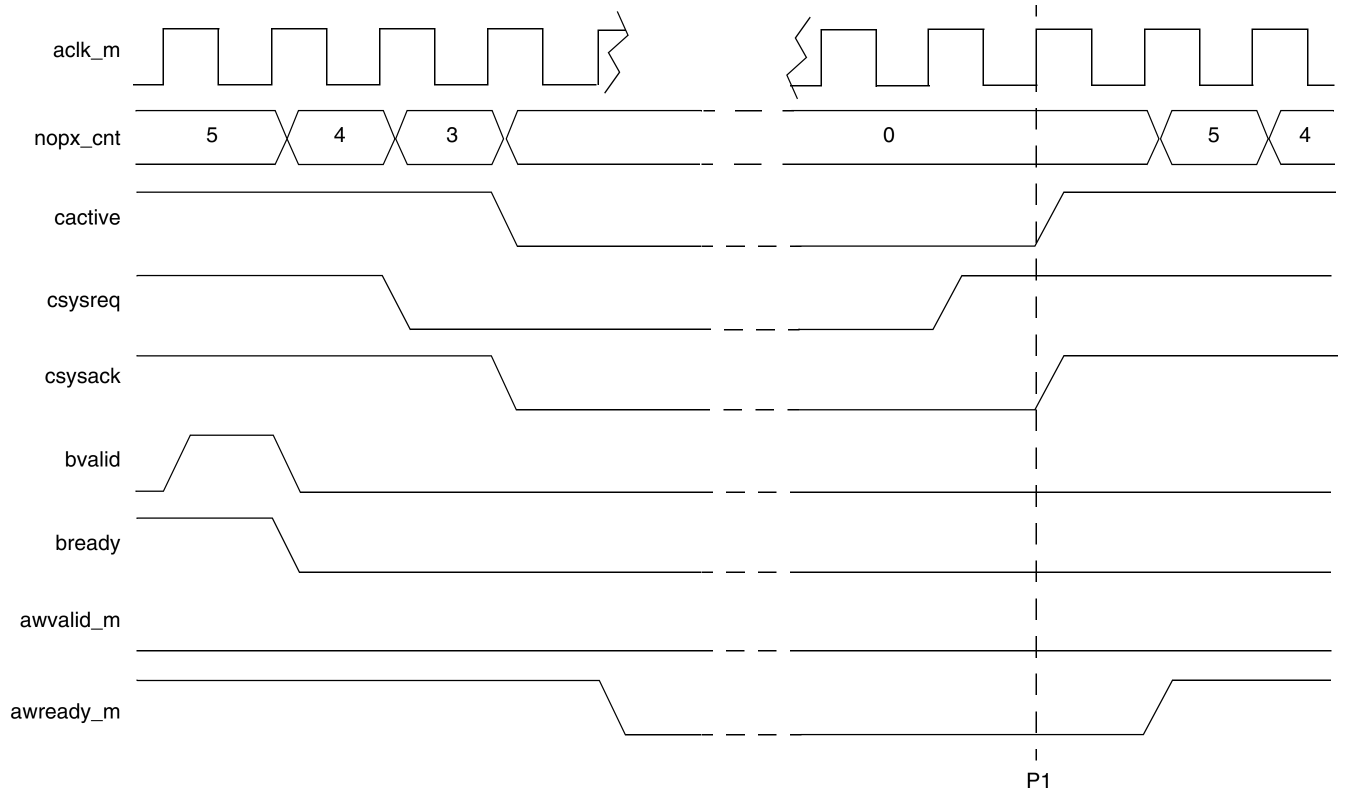
Note Early write data will not cause the cactive signal to assert or to remain asserted.

If the cactive signal is low and the csysreq signal is asserted, the DW_axi_x2x will assert the cactive signal at the same time as the csysack signal in order to complete the low-power exit handshake. After the clock edge where the cactive and csysack signals are sampled high on exit from low-power mode, the DW_axi_x2x will keep the cactive signal asserted for X2X_LOWPWR_NOPX_CNT cycles, after which it will either:

- De-assert until there are active transactions again.
- De-assert until another low-power exit handshake is completed.

This can be seen at P1 in Figure 2-11 where the cactive signal is asserted 1 clock cycle after the csysreq signal asserts, and the no-pending-transaction counter (nopx_cnt) starts to decrement again from the next cycle.

Figure 2-11 LPC Initiates Low-Power Exit



Note

When coming out of reset, if the awvalid_m and arvalid_m signals equal 0:

- cactive signal equals 0, if X2X_LOWPWR_NOPX_CNT parameter equals 0
- cactive signal equals 1, if X2X_LOWPWR_NOPX_CNT parameter is greater than 0 and will de-assert when X2X_LOWPWR_NOPX_CNT clock cycles have elapsed

2.5 Clocks and Resets

The clocking mode is configured by the X2X_CLK_MODE parameter. The following section explains the available clocking modes.

2.5.1 Synchronous Clocking

If X2X_CLK_MODE is set to be synchronous, only the Master Port clock and reset signals are present. In this configuration, there is no clock boundary-crossing logic.

2.5.2 Asynchronous Clocking

If X2X_CLK_MODE is set to asynchronous, there are separate clock and reset ports for the Master Port and Slave Port—aclk_m, aresetn_m, aclk_s, and aresetn_s. These signals can be driven by asynchronous clocks.

2.5.3 Synchronization Depth

If X2X_CLK_MODE is set to asynchronous, the user can individually configure the synchronizer depths for the Master Ports and Slave Ports with the X2X_MP_SYNC_DEPTH and X2X_SP_SYNC_DEPTH configuration parameters, which set the number of synchronizing register stages in signals that pass from the Slave Port to the Master Port, and from the Master Port to the Slave Port, respectively.

The transaction is not be driven from the Slave Port until one Master Port clock (increment write pointer) plus three Slave Port clocks later (two for synchronization plus one for status flag registering) when:

- X2X_SP_SYNC_DEPTH is set to 2
- Channel register is empty
- A transaction is pushed into the read address channel

You can choose from the following options:

- 0 – No synchronization
- 2 – Two-stage synchronization, with both stages synchronizing on the positive clock edge
- 3 – Three-stage synchronization, with all three stages synchronizing on the positive clock edge

The number of synchronization stages, or depths, directly affects the latency of the status signals of internal channel registers—full and empty—which can affect latency on valid and ready signals.

Because there is a lower risk of metastability when synchronizing signals from a faster clock domain to a slower clock domain, the user can configure different numbers of synchronization stages for the Master Port and Slave Port sides:

- Slow Master Port clock with respect to Slave Port clock yields smaller values of X2X_MP_SYNC_DEPTH and larger values of X2X_SP_SYNC_DEPTH
- Fast Master Port clock with respect to Slave Port clock yields larger values of X2X_MP_SYNC_DEPTH and smaller values of X2X_SP_SYNC_DEPTH

The desired clock frequencies also dictate how these parameters should be set, where higher clock frequencies imply more synchronization stages.

2.5.4 Quasi-Synchronous Clocking

The aclk_m and aclk_s inputs to the DW_axi_x2x are considered quasi-synchronous when they are of an integer ratio to each other and are edge-aligned. This edge alignment ensures that there is no risk of metastability, so the user can configure zero synchronization stages on both the Master Port and Slave Port side, which in turn ensures that there is no additional channel latency due to synchronization.

In quasi-synchronous mode—that is, X2X_CLK_MODE = 1 *and* X2X_SP_SYNC_DEPTH = 0 *and* X2X_MP_SYNC_DEPTH = 0—the logic paths from the internal FIFO memory registers to FIFO data out sampling registers in the pop clock domain can be constrained to be two cycle paths. Due to a register on the empty status signal from the internal dual clock FIFOs, every address location in the FIFO will be sampled a minimum of two clock cycles after it has been written.

The following synthesis constraints can be used to do this to a DW_axi_x2x instance:

```
set_multicycle_path -setup 2 -from [get_clocks aclk_m] -through
{U_AW_channel_fifo/U_dclk_fifo_U_FIFO_MEM_mem_reg* } -to [get_clocks aclk_s]
set_multicycle_path -setup 2 -from [get_clocks aclk_m] -through
{U_AR_channel_fifo/U_dclk_fifo_U_FIFO_MEM_mem_reg* } -to [get_clocks aclk_s]
set_multicycle_path -setup 2 -from [get_clocks aclk_m] -through
{U_W_channel_fifo/U_dclk_fifo0_U_FIFO_MEM_mem_reg* } -to [get_clocks aclk_s]
set_multicycle_path -setup 2 -from [get_clocks aclk_s] -through
{U_R_channel_fifo/U_dclk_fifo_U_FIFO_MEM_mem_reg* } -to [get_clocks aclk_m]
set_multicycle_path -setup 2 -from [get_clocks aclk_s] -through
{U_B_channel_fifo/U_dclk_fifo_U_FIFO_MEM_mem_reg* } -to [get_clocks aclk_m]
```

When the user has integrated the DW_axi_x2x in their system, some paths from the FIFO memory registers will not terminate within the DW_axi_x2x. In order to apply the multi-cycle path constraint to these paths, the user needs to apply their own constraints, or modify the above constraints such that the "-to" list captures registers external to the DW_axi_x2x instance.

Note The multi cycle path constraints described here are not a requirement to using quasi-synchronous mode. They can make timing easier to meet during synthesis, but are otherwise optional.

2.5.5 Reset Control

The reset signals—aresetn_m and aresetn_s should always be asserted together to avoid serious operation failures in the channel registers where either the push or pop side pointer would be cleared without clearing the other one; this would cause unreliable channel buffer status signals.

Each reset signal can be asserted, but it must be deasserted synchronously with respect to its own clock; that is, aresetn_m is deasserted synchronously with respect to aclk_m, and aresetn_s is de-asserted synchronously with respect to aclk_s.

Note DW_axi_x2x does not support asserting one reset while the other is de-asserted; doing so results in unpredictable behavior.

To avoid metastability on reset, each reset signal should be deasserted for at least three cycles of the slower clock.

No valid signal should be asserted to the DW_axi_x2x under these combined conditions:

- During assertion of aresetn_m and aresetn_s
- For one additional cycle of the relevant source channel clock
  - aclk_m for Read Address, channel, Write Address channel, and Write Data channel

- aclk_s for Read Data channel and Response channel

2.5.5.1 Reset Signal Values

While associated reset signals are asserted, output values are determined by the following:

- All valid signals driven by the DW_axi_x2x are 0 at reset
- All ready signals driven by the DW_axi_x2x are 1 during reset, except for the following:
    - bready_s and rready_s are 0 during reset for transaction-altering configurations, such as downsizing, upsizing, and so on
    - bready_s and rready_s are 1 during reset for non-transaction-altering configurations
- tz_secure_m is 0 during reset
- All other AXI signals driven by the DW_axi_x2x are 0 during reset

3 Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the actual configured state of the controller.** Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The parameter descriptions in this chapter include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of this parameter.

These tables define all of the ~~user~~ configuration options for this component.

3.1 Master Port Configuration Parameters

Table 3-1 Master Port Configuration Parameters

Label	Description
Master Port Configuration	
X2X Master Port Address Width	Width of DW_axi_x2x Master Port Addresses. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2X_MP_AW
X2X Master Port Data Width	Width of DW_axi_x2x Master Port data ports. **Values:** 8, 16, 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2X_MP_DW
X2X Master Port ID Width	Width of ID signal on the DW_axi_x2x Master Port. **Values:** 1, ..., 16 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2X_MP_IDW
X2X Master Port Burst Length Width	DW_axi_x2x master port burst length signal width. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2X_MP_BLW

3.2 Slave Port Configuration Parameters

Table 3-2 Slave Port Configuration Parameters

Label	Description
	Slave Port Configuration
X2X Slave Port Address Width	Width of DW_axi_x2x Slave Port addresses. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2X_SP_AW
X2X Slave Port Data Width	Width of DW_axi_x2x Slave Port data ports. **Values:** 8, 16, 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2X_SP_DW
X2X Slave Port ID Width	Width of ID signal on the DW_axi_x2x Slave Port. The width must be greater than or equal to Master Port ID width (X2X_MP_IDW) **Values:** 1, ..., 16 **Default Value:** X2X_MP_IDW **Enabled:** Always **Parameter Name:** X2X_SP_IDW
X2X Slave Port Burst Length Width	DW_axi_x2x slave port burst length signal width. This parameter value is controlled by the X2X_MP_BLW parameter. **Values:** 4, 5, 6, 7, 8 **Default Value:** X2X_MP_BLW **Enabled:** 0 **Parameter Name:** X2X_SP_BLW

3.3 Channel FIFO Depths Parameters

Table 3-3 Channel FIFO Depths Parameters

Label	Description
Channel FIFO Depths	
Depth of AW Channel Buffer	Depth of Write Address channel buffer. **Values:** 2, ..., 16 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2X_AW_BUF_DEPTH
Depth of W Channel Buffer	Depth of Write Data channel buffer. **Values:** 2, ..., 32 **Default Value:** 16 **Enabled:** Always **Parameter Name:** X2X_W_BUF_DEPTH
Depth of B Channel Buffer	Depth of Burst Response channel buffer. **Values:** 2, ..., 16 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2X_B_BUF_DEPTH
Depth of AR Channel Buffer	Depth of Read Address channel buffer. **Values:** 2, ..., 16 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2X_AR_BUF_DEPTH
Depth of R Channel Buffer	Depth of Read Data channel buffer. **Values:** 2, ..., 32 **Default Value:** 16 **Enabled:** Always **Parameter Name:** X2X_R_BUF_DEPTH

3.4 Clocking Parameters

Table 3-4 Clocking Parameters

Label	Description
Clock Mode	Selects whether DW_axi_x2x Slave Port clock and DW_axi_x2x Master Port clock are synchronous or asynchronous. This parameter affects the implementation of the channel buffers and the existence of the ports aclk_s and aresetn_s. **Values:** ■ Synchronous (0) ■ Asynchronous (1) **Default Value:** Synchronous **Enabled:** Always **Parameter Name:** X2X_CLK_MODE
Master Port Synchronization Depth	Number of synchronization register stages in the internal channel buffers for signals passing from DW_axi_x2x Slave Port to DW_axi_x2x Master Port. ■ 0: No synchronization stages. ■ ~~1: Two-stage synchronization; first stage is negative edge and second stage is positive edge.~~ ■ 2: Two-stage synchronization, both stages are positive edges. ■ 3: Three-stage synchronization, all stages are positive edges. If one port has a synchronization depth of 0, the other port must also be 0. This parameter is enabled only if (X2X_CLK_MODE==1). **Values:** 0, ~~1,~~ 2, 3 **Default Value:** 2 **Enabled:** X2X_CLK_MODE == 1 **Parameter Name:** X2X_MP_SYNC_DEPTH
Slave Port Synchronization Depth	Number of synchronization register stages in the internal channel buffers for signals passing from DW_axi_x2x Master Port to DW_axi_x2x Slave Port. ■ 0: No synchronization stages. ■ ~~1: Two-stage synchronization, first stage is negative edge and second stage is positive edge.~~ ■ 2: Two-stage synchronization, both stages are positive edges. ■ 3: Three-stage synchronization, all stages are positive edges. If one port has a synchronization depth of 0, the other port must also be 0. This parameter is enabled only if (X2X_CLK_MODE==1). **Values:** 0, ~~1,~~ 2, 3 **Default Value:** 2 **Enabled:** X2X_CLK_MODE == 1 **Parameter Name:** X2X_SP_SYNC_DEPTH

3.5 Low Power Handshaking Interface Parameters

Table 3-5 Low Power Handshaking Interface Parameters

Label	Description
	Low Power Handshaking Interface
Low Power Interface Enable	If True, the low-power handshaking interface (csysreq, csysack, and cactive signals) and associated control logic is implemented. If False, no support for low-power handshaking interface is provided. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_LOWPWR_HS_IF
Number of inactive clock cycles before component requests power down	Number of AXI clock cycles to wait before cactive signal de-asserts, when there are no pending transactions. Note that if csysreq de-asserts while waiting this number of cycles, cactive de-asserts immediately. If a new transaction is initiated during the wait period, the counting is halted, cactive does not de-assert, and the counting is re-initiated when there are no pending transactions. **Values:** 0, ..., 4294967295 **Default Value:** 0 **Enabled:** X2X_LOWPWR_HS_IF==1 **Parameter Name:** X2X_LOWPWR_NOPX_CNT

3.6 Active Transactions Parameters

Table 3-6 Active Transactions Parameters

Label	Description
Active Write Transactions	
Unique Write ID Accept Limit	Maximum number of unique write IDs that the DW_axi_x2x may have outstanding transactions for at any time. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** (X2X_HAS_WI_FAN_OUT == 0) && ((X2X_MP_DW != X2X_SP_DW) || (X2X_MP_BLW != X2X_SP_BLW) || X2X_HAS_ET || X2X_LOWPWR_HS_IF) **Parameter Name:** X2X_MAX_UWIDA
Write Accept Limit Per ID	Maximum number of write transactions that may be active for a particular ID value. **Values:** 1, ..., 16 **Default Value:** 4 **Enabled:** (X2X_MP_DW != X2X_SP_DW) || (X2X_MP_BLW != X2X_SP_BLW) || X2X_HAS_ET || X2X_HAS_WI_FAN_OUT || X2X_LOWPWR_HS_IF **Parameter Name:** X2X_MAX_WCA_ID
Active Read Transactions	
Unique Read ID Accept Limit	Maximum number of unique read IDs for which DW_axi_x2x may have outstanding transactions for at any time. This parameter also sets the read interleaving and read reordering depth of the DW_axi_x2x. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** (X2X_MP_DW != X2X_SP_DW) || (X2X_MP_BLW != X2X_SP_BLW) || X2X_HAS_ET || X2X_LOWPWR_HS_IF **Parameter Name:** X2X_MAX_URIDA
Read Accept Limit Per ID	Maximum number of read transactions that may be active for a particular ID value. **Values:** 1, ..., 16 **Default Value:** 4 **Enabled:** (X2X_MP_DW != X2X_SP_DW) || (X2X_MP_BLW != X2X_SP_BLW) || X2X_HAS_ET || X2X_LOWPWR_HS_IF **Parameter Name:** X2X_MAX_RCA_ID

3.7 Write Data Interleaving Parameters

Table 3-7 Write Data Interleaving Parameters

Label	Description
	Write Data Interleaving
Write Interleave Depth of X2X Master Port	Write Interleave Depth. This parameter establishes the number of write data transactions for which an external master can interleave write data. This parameter only applies to configurations with: ■ Data width altering ■ Burst length width altering ■ Endianness transformation ■ Write interleaving fan out If none of these exist in the configuration then this parameter is disabled and the DW_axi_x2x supports an infinite write interleaving depth. **Values:** 1, ..., 8 **Default Value:** 1 **Enabled:** (X2X_MP_DW != X2X_SP_DW) || (X2X_MP_BLW != X2X_SP_BLW) || X2X_HAS_WI_FAN_OUT || X2X_HAS_ET **Parameter Name:** X2X_WID

3.8 General X2X Options Parameters

Table 3-8 General X2X Options Parameters

Label	Description
	Pipelining
Add Address Channel Pipeline Stage ?	If set to true, the DW_axi_x2x includes a pipeline stage in the address channels. This allows the DW_axi_x2x to be synthesized to higher clock frequencies at the cost of one extra cycle of latency through the address channels. If set to False, the pipeline stage is removed. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** (X2X_MP_DW != X2X_SP_DW) | (X2X_HAS_ET == 1) **Parameter Name:** X2X_HAS_PIPELINE
	Transaction Upsizing
Enable Transaction Upsizing ?	Configures the DW_axi_x2x to generate transactions of a larger X2X_SP_DW asize from a smaller X2X_MP_DW, wherever possible. This parameter is enabled only if X2X_MP_DW is less than X2X_SP_DW and if the maximum number of bytes in a Master Port transaction is greater than or equal to the byte width of the Slave Port data bus. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Enabled only if X2X_MP_DW is less than X2X_SP_DW and if the maximum number of bytes in a Master Port transaction is greater than or equal to the byte width of the Slave Port data bus. **Parameter Name:** X2X_HAS_TX_UPSIZE
Upsize From Any A[R/W]SIZE Value ?	Allows the DW_axi_x2x to attempt to upsize transactions of any ARSIZE or AWSIZE value from the primary bus. If this parameter is False, the DW_axi_x2x only attempts to upsize transactions from the primary bus with a maximum ARSIZE or AWSIZE value. This parameter is enabled only if X2X_HAS_TX_UPSIZE is equal to 1. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** X2X_HAS_TX_UPSIZE == 1 **Parameter Name:** X2X_UPSIZE_ANY_ASIZE

Table 3-8 General X2X Options Parameters (Continued)

Label	Description
	Wrapping Bursts
Support Wrapping Bursts ?	When set to true, the DW_axi_x2x includes logic to handle wrapping bursts for data-width-altering configurations. If set to False, the logic is removed and the user must not drive WRAP bursts. Removing this logic significantly improves the operating frequency of DW_axi_x2x. This parameter is disabled if X2X_MP_DW is equal to X2X_SP_DW. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** (X2X_SP_DW != X2X_MP_DW) || (X2X_SP_BLW != X2X_MP_BLW) **Parameter Name:** X2X_HAS_WRAP_BURST
	Endianness Transformation
Perform Endianness Conversion ?	Configures the DW_axi_x2x for a byte-invariant endianness transformation on data and strobe signal contents. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_ET
	Trustzone Support
Pass Trustzone Signal Through the X2X ?	Controls whether the tx_secure_s signal exists on the Master Port side and the tx_secure_m signal on the Slave Port side. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_TZ_SUPPORT

3.9 Sideband Signals Parameters

Table 3-9 Sideband Signals Parameters

Label	Description
Sideband Signals	
Include Sideband Bus for Write Address Channels?	If True, then all master and slave Write Address channels have an associated sideband bus. The Write Address channel sideband bus is routed in the same way as the Write Address channel payload. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_AWSB
Include Sideband Bus for Write Data Channels?	If True, then all master and slave Write Data channels have an associated sideband bus. The Write Data channel sideband bus is routed in the same way as the Write Data channel payload. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_WSB
Include Sideband Bus for Write Response Channels?	If True, then all master and slave Write Response channels have an associated sideband bus. The Write Response channel sideband bus is routed in the same way as the Write Response channel payload. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_BSB
Include Sideband Bus for Read Address Channels?	If set to true, then all master and slave Read Address channels have an associated sideband bus. The Read Address channel sideband bus is routed in the same way as the Read Address channel payload. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_ARSB

Table 3-9 Sideband Signals Parameters (Continued)

Label	Description
Include Sideband Bus for Read Data Channels?	If set to true, then all master and slave Read Data channels have an associated sideband bus. The Read Data channel sideband bus is routed in the same way as the Read Data channel payload. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2X_HAS_RSB
Sideband Signals Width	
Width of the Write Address Channel Sideband bus	Defines width of the Write Address sideband bus. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** X2X_HAS_AWSB == 1 **Parameter Name:** X2X_AW_SBW
Width of the Write Data Channel Sideband bus	Defines width of the Write Data sideband bus. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** X2X_HAS_WSB == 1 **Parameter Name:** X2X_W_SBW
Width of the Write Response Channel Sideband bus	Defines width of the Burst Response sideband bus. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** X2X_HAS_BSB == 1 **Parameter Name:** X2X_B_SBW
Width of the Read Address Channel Sideband bus	Defines width of the Read Address sideband bus. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** X2X_HAS_ARSB == 1 **Parameter Name:** X2X_AR_SBW
Width of the Read Data Channel Sideband bus	Defines width of the Read Data sideband bus. **Values:** 1, ..., 64 **Default Value:** 4 **Enabled:** X2X_HAS_RSB == 1 **Parameter Name:** X2X_R_SBW

4 Signal Descriptions

This chapter details all possible I/O signals in the controller. For configurable IP titles, your actual configuration might not contain all of these signals.

Inputs are on the left of the signal diagrams; outputs are on the right.

Attention: For configurable IP titles, do not use this document to determine the exact I/O footprint of the controller. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the I/O signals for your actual configuration at workspace/report/IO.html or workspace/report/IO.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the I/O signals that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the widths might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

Active State: Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).

Registered: Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of No does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.

Synchronous to: Indicates which clocks in the IP sample this input (drive for an output) when considering all possible configurations. A particular configuration might not have all of the clocks listed. This clock might not be the same as the clock that your application logic should use to clock (sample/drive) this pin. For more details, consult the clock section in the databook.

Exists: Names of configuration parameters that populate this signal in your configuration.

Validated by: Assertion or de-assertion of signals that validates the signal being described.

Attributes used with Synchronous To

- Clock name - The name of the clock that samples an input or drive and output.
- None - This attribute may be used for clock inputs, hard-coded outputs, feed-through (direct or combinatorial), dangling inputs, unused inputs and asynchronous outputs.
- Asynchronous - This attribute is used for asynchronous inputs and asynchronous resets.

The I/O signals are grouped as follows:

- General Master Port Signals on page 55
- Master Port Write Address Channel on page 56
- Master Port Write Data Channel on page 59
- Master Port Write Response Channel on page 61
- Master Port Read Address Channel on page 63
- Master Port Read Data Channel on page 66
- AXI Low Power Interface on page 68
- General Slave Port Signals on page 69
- Slave Port Write Address Channel on page 70
- Slave Port Write Data Channel on page 73
- Slave Port Write Response Channel on page 75
- Slave Port Read Address Channel on page 77
- Slave Port Read Data Channel on page 80
- Debug Ports on page 82

4.1 General Master Port Signals

aclk_m
aresetn_m
tz_secure_m

Table 4-1 General Master Port Signals

Port Name	I/O	Description
aclk_m	I	Clock to the DW_axi_x2x AXI Master Port. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aresetn_m	I	Asynchronous Master Port reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of aclk_m. DW_axi_x2x does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
tz_secure_m	O	Trustzone security bit passed from DW_axi_x2x Slave Port side. Port exists only if the X2X_HAS_TZ_SUPPORT parameter is equal to 1. **Exists:** (X2X_HAS_TZ_SUPPORT == 1) **Synchronous To:** aclk_m **Registered:** ((X2X_CLK_MODE == 1) && (X2X_MP_SYNC_DEPTH >= 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Master Port Write Address Channel Signals

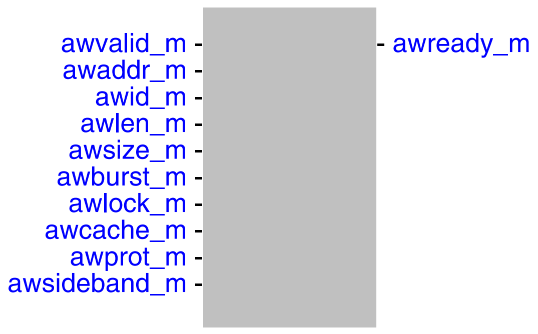


Table 4-2 Master Port Write Address Channel Signals

Port Name	I/O	Description
awvalid_m	I	Master Port write address valid. Indicates that valid write address and control information are available. Address and control information remain stable until awready signal is high. ■ 0: Address and control information not available. ■ 1: Address and control information available. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr_m[(X2X_MP_AW-1):0]	I	Master Port write address. Specifies the address of first transfer in write burst transaction. Associated control signals used to determine addresses of remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awid_m[(X2X_MP_IDW-1):0]	I	Master port write address identification. Provides identification tag for write address signals. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Master Port Write Address Channel Signals (Continued)

Port Name	I/O	Description
awlen_m[(X2X_MP_BLW-1):0]	I	Master Port write burst length. Specifies exact number of transfers in the burst and determines number of data transfers associated with the address. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize_m[2:0]	I	Master Port write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exact byte lanes to update. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awburst_m[1:0]	I	Master Port write burst. Combined with size information, shows how address for each transfer within burst is calculated. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock_m[1:0]	I	Master Port write lock. Provides additional information about the atomic characteristics of transfer. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache_m[3:0]	I	Master Port write cache. Indicates bufferable, cacheable, write-through, write-back, and allocatable attributes of transaction. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Master Port Write Address Channel Signals (Continued)

Port Name	I/O	Description
awprot_m[2:0]	I	Master Port write protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsideband_m[(X2X_AW_SBW-1):0]	I	Optional Sideband bus for write address channel. If X2X_HAS_AWSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_AWSB == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awready_m	O	Master Port write address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready. ■ 1: Slave ready. **Exists:** Always **Synchronous To:** aclk_m **Registered:** (X2X_LOWPWR_HS_IF == 0) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.3 Master Port Write Data Channel Signals

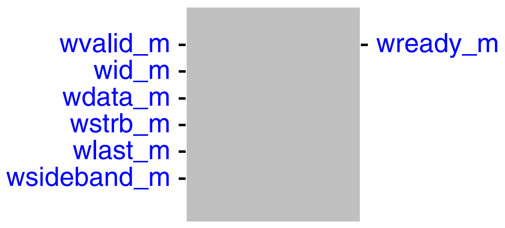


Table 4-3 Master Port Write Data Channel Signals

Port Name	I/O	Description
wvalid_m	I	Master Port write valid. Indicates that valid write data and strobes are available. ■ 0: Write data and strobes not available. ■ 1: Write data and strobes available. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wid_m[(X2X_MP_IDW-1):0]	I	Master Port write identification tag of write data transfer. This value must match the awid value of the write transaction. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wdata_m[(X2X_MP_DW-1):0]	I	Master Port write data. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-3 Master Port Write Data Channel Signals (Continued)

Port Name	I/O	Description
wstrb_m[(X2X_MP_SW-1):0]	I	Master Port write strobes. Indicates byte lanes to update in memory. One write strobe for each eight bits of write data bus. wstrb_m[n] is associated with "wdata_m[8n+7: 8n]". **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wlast_m	I	Master Port write last. Indicates last transfer in write burst. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wsideband_m[(X2X_W_SBW-1):0]	I	Optional. Master Port sideband bus for write data channel. If X2X_HAS_WSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_WSB == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wready_m	O	Master Port write ready. Indicates that slave can accept write data. ■ 0: Slave not ready. ■ 1: Slave ready. **Exists:** Always **Synchronous To:** aclk_m **Registered:** (X2X_LOWPWR_HS_IF == 0) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 Master Port Write Response Channel Signals

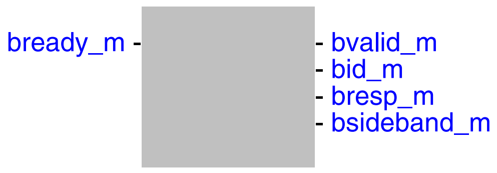


Table 4-4 Master Port Write Response Channel Signals

Port Name	I/O	Description
bvalid_m	O	Master Port write response valid. Indicates that valid write response is available. ■ 0: Write response not available. ■ 1: Write response available. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bid_m[(X2X_MP_IDW-1):0]	O	Master Port write response identification tag. This value must match awid_m value of the write transaction to which the master responds. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bresp_m[1:0]	O	Master Port write response. Indicates status of write transaction. The supported responses are OKAY, EXOKAY, SLVERR, and DECERR. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bsideband_m[(X2X_B_SBW-1):0]	O	Optional Sideband bus for write burst response channel. If X2X_HAS_BSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_BSB == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Port Write Response Channel Signals (Continued)

Port Name	I/O	Description
bready_m	I	Master Port response ready. Indicates that the master can accept response information. ■ 0: Master not ready. ■ 1: Master ready. **Exists:** Always **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Master Port Read Address Channel Signals

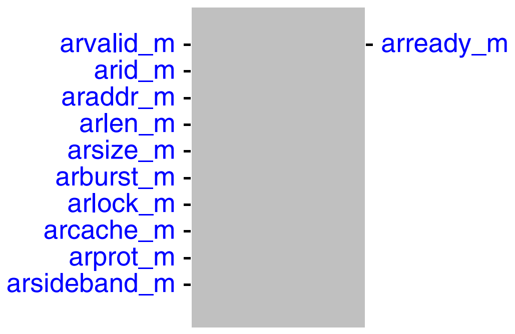


Table 4-5 Master Port Read Address Channel Signals

Port Name	I/O	Description
arvalid_m	I	Master Port read address valid. When high, this signal indicates that read address and control information are valid and remains stable until arready is high. ■ 0: Address and control information not valid. ■ 1: Address and control information valid. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
arid_m[(X2X_MP_IDW-1):0]	I	Master Port read address identification tag for read address signals. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
araddr_m[(X2X_MP_AW-1):0]	I	Master Port read address. This signal provides initial address of read burst transaction. Only start address of burst is provided, and control signals issued alongside address show how the address is calculated for remaining transfers in burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 Master Port Read Address Channel Signals (Continued)

Port Name	I/O	Description
arlen_m[(X2X_MP_BLW-1):0]	I	Master Port read burst length. This signal provides the exact number of transfers in the burst and determines the number of data transfers associated with an address. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize_m[2:0]	I	Master Port read burst size. Indicates size of each transfer in burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arburst_m[1:0]	I	Master Port read burst. Combined with size information, this signal shows how address for each transfer within burst is calculated. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock_m[1:0]	I	Master Port read lock. Provides additional information about atomic characteristics of transfer. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache_m[3:0]	I	Master Port read cache. Indicates bufferable, cacheable, write-through, write-back, and allocatable attributes of transaction. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 Master Port Read Address Channel Signals (Continued)

Port Name	I/O	Description
arprot_m[2:0]	I	Master Port read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsideband_m[(X2X_AR_SBW-1):0]	I	Optional Master port sideband bus for read address channel. If X2X_HAS_ARSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_ARSB == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arready_m	O	Master Port read address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready. ■ 1: Slave ready. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** (X2X_LOWPWR_HS_IF == 0) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.6 Master Port Read Data Channel Signals

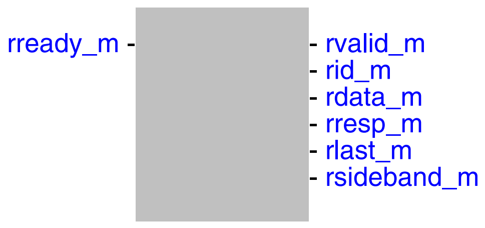


Table 4-6 Master Port Read Data Channel Signals

Port Name	I/O	Description
rvalid_m	O	Master Port read valid. Indicates that required read data is available and that the read transfer can complete. ■ 0: Read data not available. ■ 1: Read data available. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rid_m[(X2X_MP_IDW-1):0]	O	Master Port read identification tag. Value is generated by the slave and must match the arid value of read transaction to which it responds. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rdata_m[(X2X_MP_DW-1):0]	O	Master Port read data. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 Master Port Read Data Channel Signals (Continued)

Port Name	I/O	Description
rresp_m[1:0]	O	Master Port read response. Indicates status of write transaction. The supported responses are OKAY, EXOKAY, SLVERR, and DECERR. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rlast_m	O	Master Port read last. Indicates last transfer in read burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rsideband_m[(X2X_R_SBW-1):0]	O	Optional. Sideband bus for read data channel. If X2X_HAS_RSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_RSB == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rready_m	I	Master Port read ready. Indicates that the master can accept read data and response information. ■ 0: Master not ready. ■ 1: Master ready. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.7 AXI Low Power Interface Signals

csysreq -
- csysack
- cactive

Table 4-7 AXI Low Power Interface Signals

Port Name	I/O	Description
csysreq	I	System low-power request from system clock controller. ■ De-asserted by system low power controller (LPC) to initiate entry into a low-power state. ■ Asserted to initiate exit from a low-power state. **Exists:** (X2X_LOWPWR_HS_IF == 1) **Synchronous To:** aclk_m **Registered:** (X2X_LOWPWR_NOPX_CNT == 0) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
csysack	O	Low-power request acknowledgement. ■ De-asserted by DW_axi_x2x to acknowledge request to enter low-power state. ■ Asserted by DW_axi_x2x to acknowledge request to exit low-power state. **Exists:** (X2X_LOWPWR_HS_IF == 1) **Synchronous To:** aclk_m **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
cactive	O	Clock active request. De-asserted by DW_axi_x2x to inform the system low-power controller (LPC) that the clock can be removed. ■ 1: Peripheral clock required. ■ 0: Peripheral clock not required. **Note**: This clock must be removed on the positive edge after cactive and csysack signals are sampled low (0). **Exists:** (X2X_LOWPWR_HS_IF == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.8 General Slave Port Signals

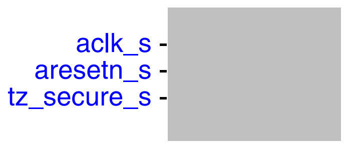


Table 4-8 General Slave Port Signals

Port Name	I/O	Description
aclk_s	I	Clock to the DW_axi_x2x AXI Slave Port. This signal is removed if the X2X_CLK_MODE parameter is equal to 0. **Exists:** (X2X_CLK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A; all signals sampled on rising edge of clock
aresetn_s	I	Asynchronous Master Port reset. This signal is removed if the X2X_CLK_MODE parameter is equal to 0. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of aclk_s. DW_axi_x2x does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (X2X_CLK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
tz_secure_s	I	Trustzone security bit passed from DW_axi_x2x Master Port side. Port exists only if the X2X_HAS_TZ_SUPPORT parameter is equal to 1. **Exists:** (X2X_HAS_TZ_SUPPORT == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((X2X_CLK_MODE == 1) && (X2X_SP_SYNC_DEPTH >= 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.9 Slave Port Write Address Channel (for x == 1) Signals

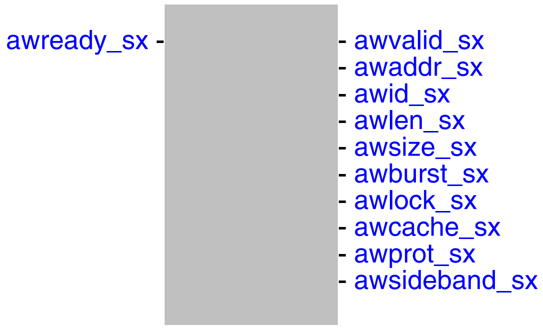


Table 4-9 Slave Port Write Address Channel (for x == 1) Signals

Port Name	I/O	Description
awvalid_sx	O	Slave Port write address valid. Indicates valid write address and control information are available. Address and control information remain stable until awready signal is high. ■ 0: Address and control information not available. ■ 1: Address and control information available. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr_sx[(X2X_SP_AW-1):0]	O	Slave write address. Specifies address of first transfer in write burst transaction. Associated control signals used to determine addresses of remaining transfers in burst. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awid_sx[(X2X_SP_IDW-1):0]	O	Slave port write address identification. Provides identification tag for write address signals. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((X2X_WID == 1) && (X2X_MP_DW != X2X_SP_DW) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-9 Slave Port Write Address Channel (for x == 1) Signals (Continued)

Port Name	I/O	Description
awlen_sx[(X2X_SP_BLW-1):0]	O	Slave Port write burst length. Specifies exact number of transfers in the burst and also determines number of data transfers associated with address. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize_sx[2:0]	O	Slave Port write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exact byte lanes to update. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awburst_sx[1:0]	O	Slave Port write burst. Combined with size information, this signal shows how address for each transfer within burst is calculated. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock_sx[1:0]	O	Slave Port write lock. Provides additional information about the atomic characteristics of transfer. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache_sx[3:0]	O	Slave Port write cache. Indicates bufferable, cacheable, write-through, write-back, and allocatable attributes of transaction. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((X2X_WID == 1) && (X2X_MP_DW != X2X_SP_DW) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-9 Slave Port Write Address Channel (for x == 1) Signals (Continued)

Port Name	I/O	Description
awprot_sx[2:0]	O	Slave Port write protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((X2X_WID == 1) && (X2X_MP_DW != X2X_SP_DW) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsideband_sx[(X2X_AW_SBW-1):0]	O	Optional. Slave port sideband bus for write address channel. If the X2X_HAS_AWSB parameter is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_AWSB == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((X2X_WID == 1) && (X2X_MP_DW != X2X_SP_DW) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awready_sx	I	Slave Port write address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready. ■ 1: Slave ready. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.10 Slave Port Write Data Channel (for x == 1) Signals

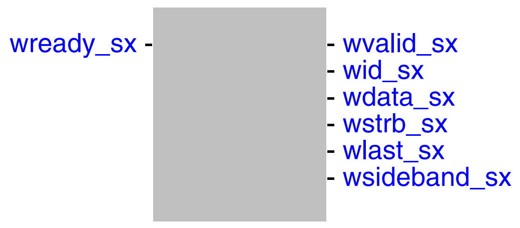


Table 4-10 Slave Port Write Data Channel (for x == 1) Signals

Port Name	I/O	Description
wvalid_sx	O	Slave Port write valid. Indicates that valid write data and strobes are available. ■ 0: Write data and strobes not available. ■ 1: Write data and strobes available. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wid_sx[(X2X_SP_IDW-1):0]	O	Slave Port write identification tag of write data transfer. This value must match awid value of the write transaction. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wdata_sx[(X2X_SP_DW-1):0]	O	Slave Port write data. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-10 Slave Port Write Data Channel (for x == 1) Signals (Continued)

Port Name	I/O	Description
wstrb_sx[(X2X_SP_SW-1):0]	O	Slave Port write strobes. Indicates byte lanes to update in memory. One write strobe for each eight bits of write data bus. wstrb_s[n] is associated with wdata_sx[8n+7: 8n]. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wlast_sx	O	Slave Port write last. Indicates last transfer in write burst. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wsideband_sx[(X2X_W_SBW-1):0]	O	Optional. Slave Port x sideband bus for write data channel. If X2X_HAS_WSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_WSB == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wready_sx	I	Slave port write ready. Indicates that slave can accept write data. ■ 0: Slave not ready ■ 1: Slave ready. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.11 Slave Port Write Response Channel (for x == 1) Signals

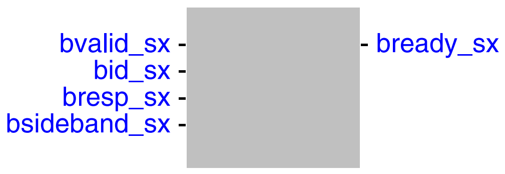


Table 4-11 Slave Port Write Response Channel (for x == 1) Signals

Port Name	I/O	Description
bvalid_sx	I	Slave port write response valid. Indicates that valid write response is available. ■ 0: Write response not available. ■ 1: Write response available. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bid_sx[(X2X_SP_IDW-1):0]	I	Slave port write response identification tag. Value must match awid_sx value of write transaction to which slave responds. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bresp_sx[1:0]	I	Slave port write response. Indicates status of write transaction. The supported responses are OKAY, EXOKAY, SLVERR, and DECERR. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bsideband_sx[(X2X_B_SBW-1):0]	I	Optional Sideband bus for write burst response channel. If the X2X_HAS_BSB parameter is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_BSB == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-11 Slave Port Write Response Channel (for x == 1) Signals (Continued)

Port Name	I/O	Description
bready_sx	O	Slave port response ready. Indicates that the slave can accept response information. ■ 0: Master not ready. ■ 1: Master ready. **Exists:** Always **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (((X2X_MP_DW == X2X_SP_DW) && (X2X_MP_BLW == X2X_SP_BLW))) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.12 Slave Port Read Address Channel Signals

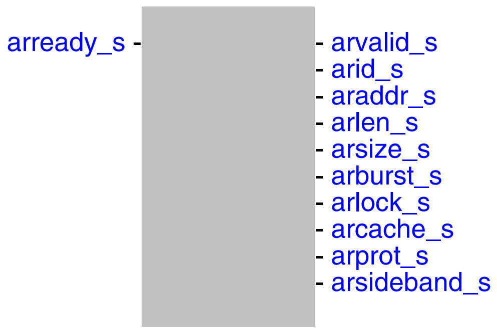


Table 4-12 Slave Port Read Address Channel Signals

Port Name	I/O	Description
arvalid_s	O	Slave Port read address valid. When high, indicates read address and control information is valid and remains stable until arready is high. ■ 0: Address and control information not valid. ■ 1: Address and control information valid. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
arid_s[(X2X_SP_IDW-1):0]	O	Slave Port read address identification tag for read address signals. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (((X2X_MP_DW) != (X2X_SP_DW)) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
araddr_s[(X2X_SP_AW-1):0]	O	Slave Port read address. This signal provides initial address of read burst transaction. Only start address of burst is provided, and control signals issued alongside address show how address is calculated for remaining transfers in burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-12 Slave Port Read Address Channel Signals (Continued)

Port Name	I/O	Description
arlen_s[(X2X_SP_BLW-1):0]	O	Slave Port read burst length. This signal provides exact number of transfers in burst; determines number of data transfers associated with address. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize_s[2:0]	O	Slave Port read burst size. Indicates size of each transfer in burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arburst_s[1:0]	O	Slave Port read burst type; INCR, FIXED, or WRAP. Combined with size information, shows how address for each transfer within burst is calculated. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock_s[1:0]	O	Slave Port read lock. Provides additional information about atomic characteristics of transfer. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache_s[3:0]	O	Slave Port read cache. Indicates bufferable, cacheable, read-through, read-back, and allocatable attributes of transaction. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (((X2X_MP_DW) != (X2X_SP_DW)) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-12 Slave Port Read Address Channel Signals (Continued)

Port Name	I/O	Description
arprot_s[2:0]	O	Slave Port read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (((X2X_MP_DW) != (X2X_SP_DW)) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsideband_s[(X2X_AR_SBW-1):0]	O	Optional Sideband bus for read address channel. If X2X_HAS_ARSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_ARSB == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (((X2X_MP_DW) != (X2X_SP_DW)) && (X2X_HAS_PIPELINE == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arready_s	I	Slave Portread address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready. ■ 1: Slave ready. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.13 Slave Port Read Data Channel Signals

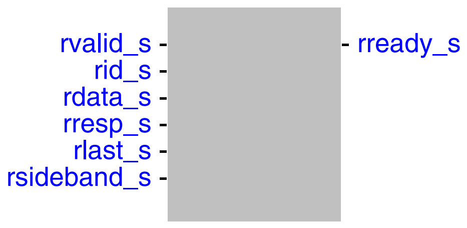


Table 4-13 Slave Port Read Data Channel Signals

Port Name	I/O	Description
rvalid_s	I	Slave Port read valid. Indicates that required read data is available and read transfer can complete. ■ 0: Read data not available. ■ 1: Read data available. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rid_s[(X2X_SP_IDW-1):0]	I	Slave Port read identification tag. Value is generated by slave and must match arid value of read transaction to which it responds. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rdata_s[(X2X_SP_DW-1):0]	I	Slave Port read data **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp_s[1:0]	I	Slave Port write response. Indicates status of write transaction. The supported responses are OKAY, EXOKAY, SLVERR, and DECERR. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-13 Slave Port Read Data Channel Signals (Continued)

Port Name	I/O	Description
rlast_s	I	Slave Port read last. Indicates last transfer in read burst. **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rsideband_s[(X2X_R_SBW-1):0]	I	Optional Sideband bus for read data channel. If X2X_HAS_RSB is false, then this port does not appear on the I/O list. **Exists:** (X2X_HAS_RSB == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rready_s	O	Slave Port Slave Port read ready. Indicates master can accept read data and response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** (X2X_CH_SEL == 0) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** ((((X2X_MP_DW == X2X_SP_DW) && (X2X_MP_BLW == X2X_SP_BLW))) && (X2X_HAS_ET == 0)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.14 Debug Ports Signals

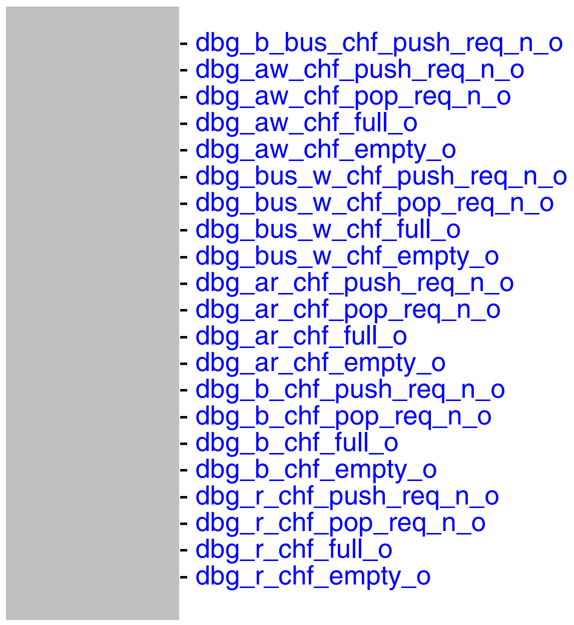


Table 4-14 Debug Ports Signals

Port Name	I/O	Description
dbg_b_bus_chf_push_req_n_o[(X2X_NUM_W_PORTS-1):0]	O	debug signal, write response bus channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_aw_chf_push_req_n_o	O	debug signal, write address channel FIFO push request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_aw_chf_pop_req_n_o	O	debug signal, write address channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Table 4-14 Debug Ports Signals (Continued)

Port Name	I/O	Description
dbg_aw_chf_full_o	O	debug signal, write address channel FIFO full status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_aw_chf_empty_o	O	debug signal, write address channel FIFO empty status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_bus_w_chf_push_req_n_o[(X2X_NUM_W_PORTS-1):0]	O	debug signal, write data channel FIFO push request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_bus_w_chf_pop_req_n_o[(X2X_NUM_W_PORTS-1):0]	O	debug signal, write data channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_bus_w_chf_full_o[(X2X_NUM_W_PORTS-1):0]	O	debug signal, write data channel FIFO full status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_bus_w_chf_empty_o[(X2X_NUM_W_PORTS-1):0]	O	debug signal, write data channel FIFO empty status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-14 Debug Ports Signals (Continued)

Port Name	I/O	Description
dbg_ar_chf_push_req_n_o	O	debug signal, read address bus channel FIFO push request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_ar_chf_pop_req_n_o	O	debug signal, read address bus channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_ar_chf_full_o	O	debug signal, read address channel FIFO full status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_ar_chf_empty_o	O	debug signal, read address channel FIFO empty status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_b_chf_push_req_n_o	O	debug signal, write response channel FIFO push request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_b_chf_pop_req_n_o	O	debug signal, write response channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Table 4-14 Debug Ports Signals (Continued)

Port Name	I/O	Description
dbg_b_chf_full_o	O	debug signal, write response channel FIFO full status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_b_chf_empty_o	O	debug signal, write response channel FIFO empty status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_r_chf_push_req_n_o	O	debug signal, read data bus channel FIFO push request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_r_chf_pop_req_n_o	O	debug signal, read data bus channel FIFO pop request. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dbg_r_chf_full_o	O	debug signal, read data channel FIFO full status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** (X2X_CLK_MODE==1) ? "aclk_s" : "aclk_m" **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dbg_r_chf_empty_o	O	debug signal, read data channel FIFO empty status. **Exists:** (X2X_HAS_INC_DEBUG_LOGIC == 1) **Synchronous To:** aclk_m **Registered:** (X2X_CLK_MODE==1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5 Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table 5-1 Internal Parameters

Parameter Name	Equals To
X2X_CH_SEL	0
X2X_HAS_INC_DEBUG_LOGIC	0
X2X_HAS_WI_FAN_OUT	0
X2X_MP_SW	(X2X_MP_DW/8)
X2X_NUM_W_PORTS	=((X2X_HAS_WI_FAN_OUT == 1) ? X2X_WID : 1)
X2X_SP_SW	(X2X_SP_DW/8)

6 Verification

This chapter provides an overview of the testbench available for the DW_axi_x2x verification. Once the DW_axi_x2x has been configured and the verification setup, simulations can be run automatically. For information on running simulations for DW_axi_x2x in coreAssembler or coreConsultant, see the "Running the Simulation" section in the user guide.

Note The DW_axi_x2x verification testbench is built using Synopsys SVT Verification IP (VIP). Ensure that you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the "Supported Versions of Tools and Libraries" section in the installation guide.

This chapter discusses the following sections:

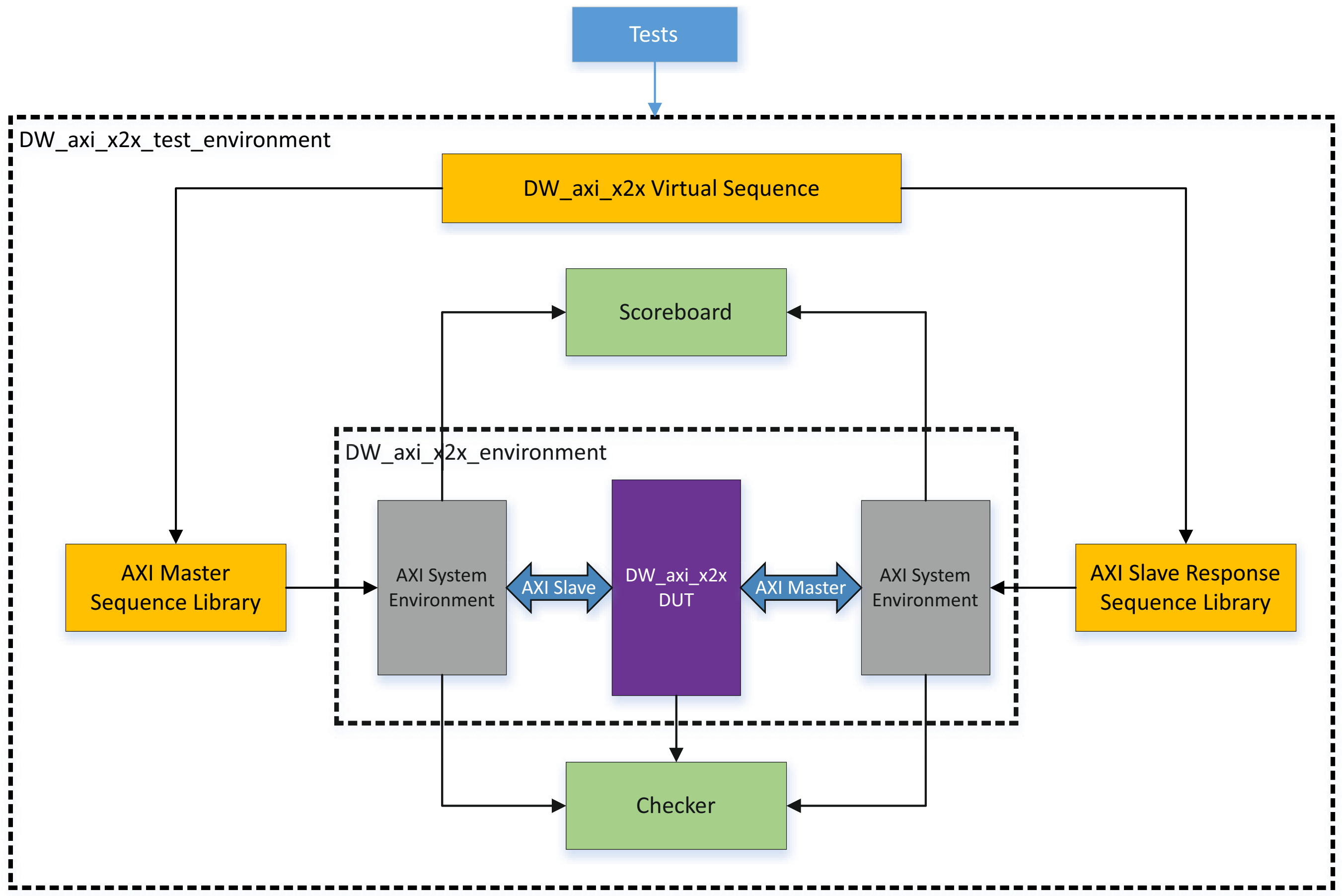
- "Verification Environment" on page 89
- "Testbench Directories and Files" on page 90
- "Packaged Testcases" on page 91

6.1 Verification Environment

DW_axi_x2x is verified using a UVM-methodology-based constrained random verification environment. The environment can generate random scenarios and the test case has hooks to control the scenarios to be generated.

Figure 6-1 shows the verification environment of the DW_axi_x2x testbench:

Figure 6-1 DW_axi_x2x UVM Verification Environment



The testbench consists of the following elements:

- Testbench makes use of the standard SVT VIP for the protocol interfaces:
  - AMBA SVT VIP
    - AXI VIP for the interface with AXI Master Data transfer interface
    - AXI Master VIP model is used to generate random sequences
    - AXI Slave VIP model is used to generate random response sequences

6.2 Testbench Directories and Files

The DW_axi_x2x verification environment contains the following directories and associated files.

Table 6-1 shows the various directories and associated files:

Table 6-1 DW_axi_x2x Testbench Directory Structure

Directory	Description
<configured workspace>/sim/testbench	Top level testbench module (test_top.sv) and the DUT to the testbench wrapper (dut_sv_wrapper.sv) exist in this folder.
<configured workspace>/sim/testbench/env	Contains testbench files. For example, scoreboard, sequences, VIP, environment, sequencers, and agents.
<configured workspace>/sim/	Primarily contains the supporting files to compile and run the simulation. After the completion of the simulation, the log files are present here.
<configured workspace>/sim/test_*	Contains individual test cases. After the completion of the simulation, the test specific log files and if applicable the waveform files are stored here.

6.3 Packaged Testcases

The simulation environment that comes as a package file includes some demonstrative tests. Some or all of the packaged demonstrative tests, depending upon their applicability to the chosen configuration are displayed in **Setup and Run Simulations** > **Testcases** in the coreConsultant GUI.

The associated shipped test cases and their descriptions are explained in Table 6-2:

Table 6-2 DW_axi_x2x Test Description

Test Name	Test Description
test_x2x_random	This test is aimed at generating random traffic which is AXI compliant. The traffic is generated randomly based on DW_axi_x2x configuration and VIP is auto-constrained to generate the traffic within the protocol limits. The traffic is generated in an outstanding fashion and sent towards the DUT. AXI Master Sequence from the VIP generates various possible AXI transfers for both Write and Read requests in parallel. The AXI transfer control attributes are randomized within the protocol and as per the DW_axi_x2x requirement. The Primary port is monitored for the correctness of the AXI protocol across different IP configuration. AXI Slave Sequence from the VIP generates various possible AXI responses for the requests from secondary port of the DW_axi_x2x. The response, delays and other attributes are generated randomly. The secondary port is monitored for the correctness of the AXI protocol across different IP configuration.

7 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment

7.1 Performance

This section discusses performance and the hardware configuration parameters, that affect the performance of the DW_axi_x2x.

7.1.1 Power Consumption, Frequency, Area, and DFT Coverage

Table 7-1 provides information about the synthesis results (power consumption, frequency, area) and DFT coverage of the DW_axi_x2x using the industry standard 16nm technology library.

Table 7-1 Synthesis Results for DW_axi_x2x

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov (%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Default Configuration	400 MHz	1921	20 nW	0.875 mW	99.94	99.86	99.2

Table 7-1 Synthesis Results for DW_axi_x2x

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov (%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Typical Configuration- 1 X2X_AR_BUF_DEPTH = 16 X2X_AW_BUF_DEPTH = 16 X2X_B_BUF_DEPTH = 16 X2X_CLK_MODE = 1 X2X_MAX_RCA_ID = 16 X2X_MAX_URIDA = 16 X2X_MAX_UWIDA = 16 X2X_MAX_WCA_ID = 16 X2X_MP_DW = 128 X2X_MP_IDW = 16 X2X_R_BUF_DEPTH = 32 X2X_SP_DW = 32 X2X_SP_IDW = 16 X2X_WID = 8 X2X_HAS_WRAP_BURST = 1 X2X_W_BUF_DEPTH = 32	400 MHz	399703	0.0003 mW	14.200 mW	99.96	99.87	99.8
Typical Configuration -2 X2X_AR_BUF_DEPTH = 16 X2X_AW_BUF_DEPTH = 16 X2X_B_BUF_DEPTH = 16 X2X_CLK_MODE = 1 X2X_HAS_ET = 1 X2X_HAS_TX_NO_ALTER = 1 X2X_MAX_RCA_ID = 16 X2X_MAX_URIDA = 16 X2X_MAX_UWIDA = 16 X2X_MAX_WCA_ID = 16 X2X_MP_DW = 32 X2X_MP_IDW = 16 X2X_R_BUF_DEPTH = 32 X2X_SP_DW = 128 X2X_SP_IDW = 16 X2X_TX_ALTER = 1 X2X_WID = 8 X2X_W_BUF_DEPTH = 32	400 MHz	276838	200 nW	9.830 mW	99.96	99.89	99.9

7.1.2 Latency

Since a synchronization depth of 1 results in two-stage synchronization—with the first stage registering on the negative edge and the second stage registering on the positive edge—this may appear as two cycles of synchronization, depending on when the edge being synchronized occurs in the synchronizing clock cycle. That is, if the edge being synchronized occurs before a consecutive negative edge and positive edge of the synchronizing clock, it appears as one synchronization cycle. However, if it appears before a positive edge, since the first stage of synchronization does not capture the signal until the next negative edge of the synchronizing clock, it appears as two synchronization cycles.

7.1.2.1 Latency Calculations

Note The values "+1*clk_s" and "+1*clk_m" in the equations below are the effect of the status flag registering in the channel buffers.

- Latency from respective Master Port to Slave Port address channel valid signals is addr_ch_latency:

  addr_ch_latency = 1*clk_m + X2X_CLK_MODE ? ((X2X_SP_SYNC_DEPTH *clk_s + 1*clk_s) + (X2X_HAS_PIPELINE ? 1*clk_s : 0)) : (X2X_HAS_PIPELINE ? 1*clk_m : 0)

  These latencies include:

  - awvalid_m to awvalid_s(*i)*
  - arvalid_m to arvalid_s
- Latency from awvalid_m to wvalid_s(*i*)— earliest time the DW_axi_x2x can forward a write beat issued in the same cycle as the corresponding command—is aw_to_w_latency:

  aw_to_w_latency = addr_ch_latency + X2X_CLK_MODE ? 1*clk_s : 1*clk_m
- Latency can accept a beat of read data from the master after the master sends the read command—is ar_to_r_latency:

  ar_to_r_latency = addr_ch_latency + (X2X_CLK_MODE ? 1*clk_s : 1*clk_m)
- Latency from wvalid_m to wvalid_s(*i*) is w_to_w_latency:

  w_to_w_latency = 1*clk_m + X2X_CLK_MODE ? (X2X_SP_SYNC_DEPTH *clk_s + 1*clk_s) : 0
- Latency from rvalid_s to rvalid_m and bvalid_s(*i*) to bvalid_m:

  r_to_r_latency = b_to_b_latency = 1*clk_s + X2X_CLK_MODE ? (X2X_MP_SYNC_DEPTH*clk_m + 1*clk_m) : 0

7.2 Hardware Considerations

The following subsections discuss considerations you should bear in mind when integrating the DW_axi_x2x in your design.

7.2.1 Usage Requirements

The following are requirements for using the DW_axi_x2x in your design:

- Using DW_axi_x2x to Connect Two Interconnects – When using the DW_axi_x2x to connect two DW_axi interconnects, you should be careful to avoid sending transactions that loop indefinitely between both interconnects. For example, when configuring the two DW_axi interconnects, set the slave visibility of the DW_axi_x2x on each DW_axi so that it cannot access the other DW_axi_x2x bridge. This prevents any transactions that loop from one interconnect to the other.

7.2.2 Buffer Requirements

The following topics discuss buffer requirements in your design.

7.2.2.1 Channel Buffers

The depth of the channel buffers is set by related coreConsultant parameters. The width of the buffers is set by Master Port parameters; that is, the buffers are wide enough to accommodate the AXI signals on the Master Port. If sideband signals exist for a given channel, they are also passed through the channel FIFO, which causes the width of the channel FIFO to increase by the width of the sideband signal.

Whenever a channel FIFO becomes full, the ready to the channel source is de-asserted and the channel stalls until there is space in the buffer. Keep in mind the downsizing or upsizing ratio when choosing FIFO depths for the read and write data channels, since this affects how often the channel could stall due to a full buffer condition.

For example, if X2X_MP_DW=128 and X2X_SP_DW=16, this gives a downsizing ratio of 128÷16 = 8. Thus, if the master in this case always sends transactions of the maximum 128-bit size, then at best the Write Data channel FIFO will empty at one-eighth of the rate that it can be filled.

The same condition exists for an upsizing configuration on the Read Data channel; that is, if X2X_SP_DW=128 and X2X_MP_DW=16, then assuming that all the transactions are upsized to 128 bits, the worst case scenario occurs where the Read Data channel empties at 1one-eighth of the rate that it can be filled.

For more information on downsizing and upsizing, refer to "Address Channel Transactions" on page 23.

7.2.2.2 Inter-Channel Communication Buffers

For transactions that alter configurations, the DW_axi_x2x instantiate internal buffers that pass information between the AXI channels. The depth and width of these buffers depend on how the DW_axi_x2x is configured.

If no transaction altering is required for the configuration, then these buffers are not instantiated.

For more information on transaction altering, refer to "Address Channels" on page 22.

7.2.2.2.1 Bus Utilization

For each inter-channel communication buffers, one entry of the buffer is used for every transaction issued from the DW_axi_x2x Slave Port. Additionally, the address channel at the Slave Port stalls if there is no room in these buffers for the next transaction. Because of these two circumstances, you should carefully choose the depths of these buffers in order to balance the size of the design with bus utilization for the expected traffic profile.

For example, a worst case scenario would be a configuration with X2X_MP_DW=512 and X2X_SP_DW=8. If the master issues a maximum-sized read transaction, it would result in 1024 bytes; that is, (512÷8)*16 = 1024 bytes.

Given that the maximum number of bytes that a single transaction on the Slave Port can access is 16 bytes—that is, 1*16 = 16 bytes—this results in 1024÷16 = 64 transactions from the Master Port. Thus, if X2X_MAX_RCA_ID was set to the maximum of 16, then the DW_axi_x2x does the following:

1. Issue 16 of the 64 read transactions
2. Wait until one of the 16 transactions completes before it can send another transaction
3. Repeat until all 64 transactions complete

This means that the Read Address channel buffer empties very slowly with respect to how quickly it could fill up, so there is an increased possibility of a stalled Read Address channel—ready low—at the Master Port. Though this is a drastic scenario, you should keep in mind that this type of situation might occur in your configuration in order to decide how best to trade off FIFO sizes.

7.2.2.3 Write Address to Write Data Channel Buffer

The Write Data channel has one FIFO through which it communicates with the Write Address channel. The depth of this FIFO is (X2X_MAX_UWIDA * X2X_MAX_WCA_ID), and the width depends on the level of transaction altering—that is, different data widths, same data widths but endian translation, and so on—that the configuration must perform, coupled with the data widths on both ports.

For a typical 128-bit to 64-bit downsizing configuration with X2X_WID=1, the width of this FIFO is 16 bits. For all configurations, the width of the FIFO is controlled such that it is only as big as required for the configuration; for example, for a 64- to 128-bit upsizing configuration, the FIFO widths are 13 bits.

For more information on address and channel buffers, refer to “Address Channels” on page 22 and “Channel Buffers” on page 21.

7.2.2.4 Read Address to Read Data Channel Buffer

The Read Data channel contains X2X_MAX_URIDA FIFOs, each with a depth equal to X2X_MAX_RCA_ID. The widths of these FIFOs depend on the level of transaction altering—that is, different data widths, same data widths but endian translation, and so on—that the configuration must perform and the data widths on both ports, except that the width of the FIFOs must also include the transaction ID due to the difference in read and write ordering rules. Thus this FIFO must be X2X_MP_IDW bits wider than the Write Address to Write Data channel buffer.

For more information on address and channel buffers, refer to “Address Channels” on page 22 and “Channel Buffers” on page 21.

7.2.2.5 Write Address to Burst Response Channel Buffer

The Burst Response channel has X2X_MAX_UWIDA FIFOs, each with a depth of X2X_MAX_WCA_ID and each of a width of (X2X_MP_IDW + 2) bits.

A Basic Core Module (BCM) Library

The Basic Core Module (BCM) Library is a library of commonly used blocks for the Synopsys DesignWare IP development. These BCMs are configurable on an instance-by-instance basis and, for the majority of BCM designs, there is an equivalent (or nearly equivalent) DesignWare Building Block (DWBB) component.

This chapter provides more information about the BCMs used in DW_axi_x2x.

- "BCM Library Components" on page 99
- "Synchronizer Methods" on page 100

Note The DesignWare Building Blocks (DWBB) contains several synchronizer components with functionality similar to methods documented in this appendix. For more information about the DWBB synchronizer components go to:
https://www.synopsys.com/dw/buildingblock.php

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_axi_x2x.

Table A-1 List of BCM Library Components used in the Design

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_x2x_bcm05	FIFO controller interface for one of two clock domains instantiated in DW_axi_x2x_bcm07	DW_fifoctl_if Submodule of DW_fifoctl_s2_sf
DW_axi_x2x_bcm06	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_axi_x2x_bcm07	Synchronous (Dual-Clock) FIFO Controller with Static Flags	DW_fifoctl_s2_sf
DW_axi_x2x_bcm21	Single Clock Data Bus Synchronizer	DW_sync
DW_axi_x2x_bcm57	Synchronous Write-Port, Asynchronous Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff

Table A-1 List of BCM Library Components used in the Design

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_x2x_bcm58	Dual clock two port RAM with retiming registers (Flip-Flop Based)	
DW_axi_x2x_bcm51	Arbiter with Static Priority Scheme	DW_arb_sp
DW_axi_x2x_bcm65	Synchronous (Single Clock) FIFO with Static Flags	DW_fifo_s1_sf
DW_axi_x2x_bcm66	Synchronous (Dual Clock) FIFO with Static Flags	DW_fifo_s2_sf

A.2 Synchronizer Methods

This section also describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_axi_x2x to cross clock boundaries.

This section contains the following:

- "Synchronizers used in DW_axi_x2x" on page 100
- "Synchronizer 1: Simple Double Register Synchronizer (DW_axi_x2x)" on page 100
- "Synchronizer 2: Synchronous (Dual-clock) FIFO with static flags (DW_axi_x2x)" on page 101

A.2.1 Synchronizers used in DW_axi_x2x

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_axi_x2x are listed and cross referenced to the synchronizer type in Table A-2. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-2 Synchronizers Used in DW_axi_x2x

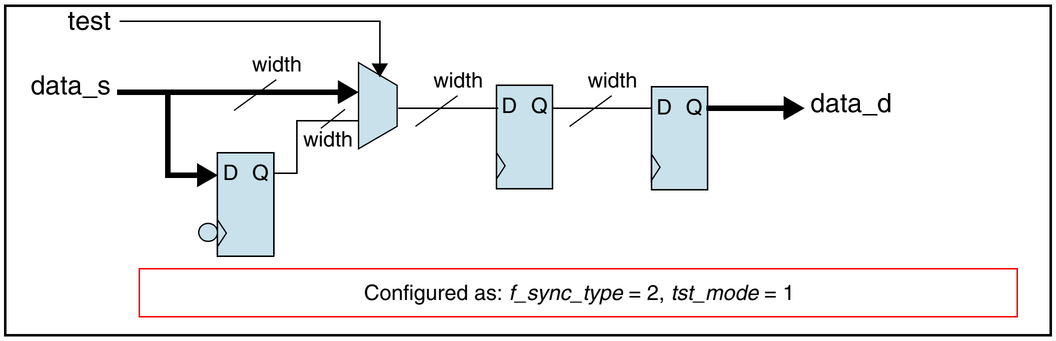
Synchronizer module file	Sub module file	Synchronizer Type and Number
DW_axi_x2x_bcm21.v		Synchronizer 1: Simple Multiple register synchronizer
DW_axi_x2x_bcm66.v	DW_axi_x2x_bcm05.v DW_axi_x2x_bcm07.v DW_axi_x2x_bcm21.v DW_axi_x2x_bcm58.v	Synchronizer 2: Synchronous dual clock FIFO with Static Flags

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_axi_x2x)

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If aclk_m and clk_s are asynchronous (X2X_CLK_MODE =1) then DW_axi_x2x_bcm21 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameters

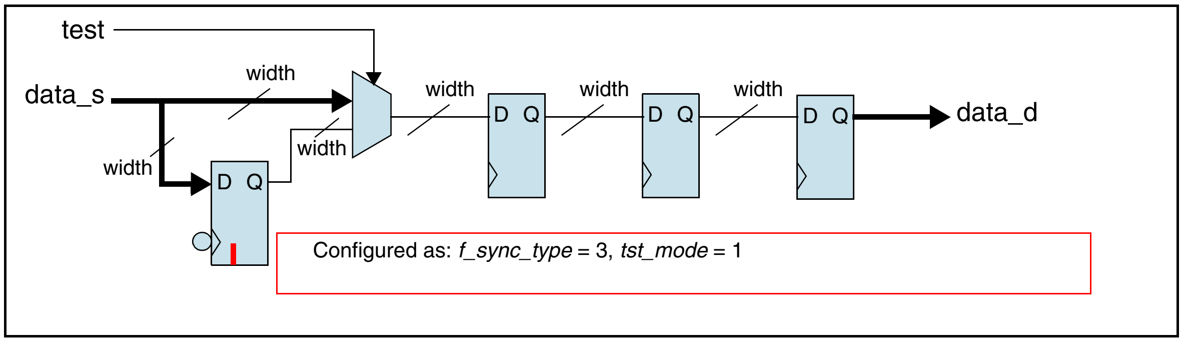
X2X_MP_SYNC_DEPTH and X2X_SP_SYNC_DEPTH. The following example shows the two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block diagram of Synchronizer 1 with two stage synchronization (both positive edge)



The following example shows the three stage synchronization process (Figure A-2) both using positive edge of clock.

Figure A-2 Block diagram of Synchronizer 1 with three stage synchronization (both positive edge)



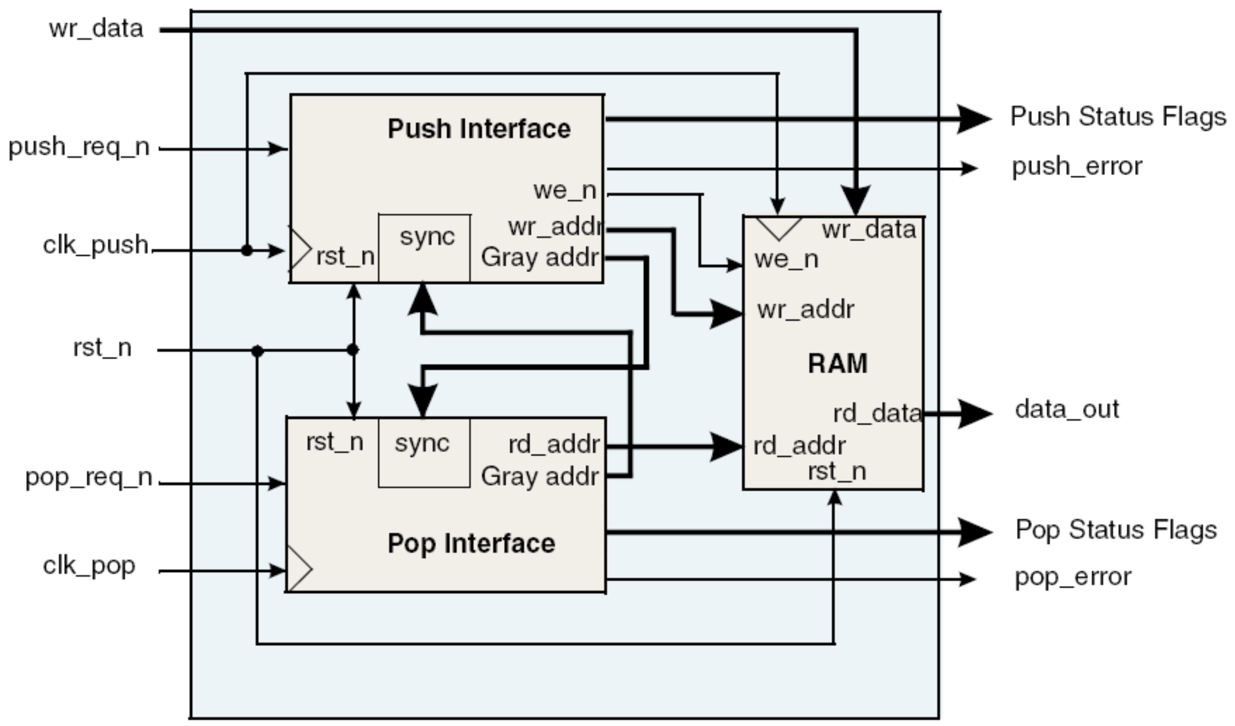
A.2.3 Synchronizer 2: Synchronous (Dual-clock) FIFO with static flags (DW_axi_x2x)

DW_axi_x2x_bcm66 is a dual independent clock FIFO. It combines the DW_axi_x2x_bcm07 FIFO controller and the DW_axi_x2x_bcm58 flip-flop based RAM DesignWare components.

The FIFO provides parameterized width and depth, and a full complement of flags (full, almost full, half full, almost empty, empty, and error) for both of the clock domains.

Figure A-3 shows the block diagram of Synchronizer 2.

Figure A-3 Synchronizer 2 Block diagram



B Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocking command	A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.
cycle command	A command that executes and causes HDL simulation time to advance.

decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWareSynthesizable Components for AMBA.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.

peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
sparse data	Data bus data which is individually byte-enabled. The AXI bus allows sparse data transfers using the WSTRB signal, each byte lane individually enabled. The AHB bus does not allow sparse data transfers.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index