SYNOPSYS®

DesignWare® DW_axi_x2h

Databook

DW_axi_x2h – *Product Code*

2.04a
March 2020

Copyright Notice and Proprietary Information

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.03c onward.

Date	Release	Description
2.04a	March 2020	■ Revision version change for 2020.03a release ■ Added "User/Sideband Signals" on page 33 ■ Updated synthesis results in "Performance" on page 81 ■ Updated "Verification" on page 71 ■ Updated "Reset Input Usage" on page 38 ■ "Signal Descriptions" on page 47, "Parameter Descriptions" on page 39. "Internal Parameter Descriptions" on page 69auto-extracted from the RTL
2.03a	February 2018	■ Revision version change for 2018.02a release ■ Updated synthesis results in "Performance" on page 81 ■ 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 47, "Parameter Descriptions" on page 39. "Internal Parameter Descriptions" on page 69auto-extracted from the RTL with change bars
2.02a	March 2016	■ Revision version change for 2016.03a release ■ Added "Running SpyGlass® Lint and SpyGlass® CDC" section ■ Added "Running Spyglass on Generated Code with coreAssembler" section ■ "Signal Descriptions" on page 47 and "Parameter Descriptions" on page 39 auto-extracted from the RTL ■ Added chapter "Internal Parameter Descriptions" on page 69 ■ Added Appendix A, "Basic Core Module (BCM) Library" ■ Updated area and power numbers in "Performance" on page 81
2.01a	October 2014	■ Version change for 2014.10a release. ■ Updated the "Integration Considerations" chapter.
2.00a	June 2013	Added information about new and modified signals and parameters to support the AMBA 4 AXI and ACE-Lite specification.
1.07a	May 2013	Version change for 2013.05a release. Updated the template.

Date	Release	Description
1.06b	Oct 2012	Added the product code on the cover and in Table 1-1.
1.06b	Oct 2011	Version change for 2011.10a release.
1.06a	Jun 2011	Updated system diagram in Figure 1-1; enhanced "Related Documents" section in Preface.
1.06a	Jan 2011	Version change for 2010.12a release.
1.05a	Sep 2010	Corrected names of include files and vcs command used for simulation
1.04a	Dec 2009	Edited mhgrant and mhbusreq signal descriptions; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.04a	May 2009	Removed references to QuickStarts, as they are no longer supported.
1.04a	Oct 2008	Updated "Two Synchronous Clocks" section.
1.03d	Jun 2008	Version change for 2008.06a release.
1.03c	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information that you need to interface the DW_axi_x2h component to the Synopsys DesignWare Synthesizable Components for AMBA environment. This component conforms to the AMBA 3 AXI and AMBA 4 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_x2h.

- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_x2h.

- Chapter 4, "Signal Descriptions" provides a list and description of the DW_axi_x2h 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_x2h.

- Chapter 7, "Integration Considerations" includes information you need to integrate the configured DW_axi_x2h into your design.

- Appendix A, "Basic Core Module (BCM) Library" documents the synchronizer methods (blocks of synchronizer functionality) used in DW_axi_x2h 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 2 and AMBA 3 AXI/AMBA 4 AXI, see 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 multilayer 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.

Component Name	Description
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 slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI or AMBA 4 AXI subsystems
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_x2h is a configurable bridge between the ARM AMBA 3 AXI/AMBA 4 AXI protocol bus and the ARM AMBA 2.0 protocol AHB bus. The DW_axi_x2h component has both an AMBA3 AXI/AMBA 4 AXI-compliant slave interface and an AMBA 2.0 AHB-compliant master interface. It is part of the family of DesignWare Synthesizable Components for AMBA. The DW_axi_x2h AXI slave interface connects to an AXI master interface—for example, the DW_axi AXI interconnect or the DW_axi_gm AXI master gasket. The DW_axi_x2h AHB master interface connects to an AHB bus — for example, the DW_ahb AHB interconnect fabric.

> ☞ **Note**   You must have a DWC-AMBA-Fabric-Source license to enable the AXI4 interface.

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 (Advanced eXtensible Interface) 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_x2h in a Complete System
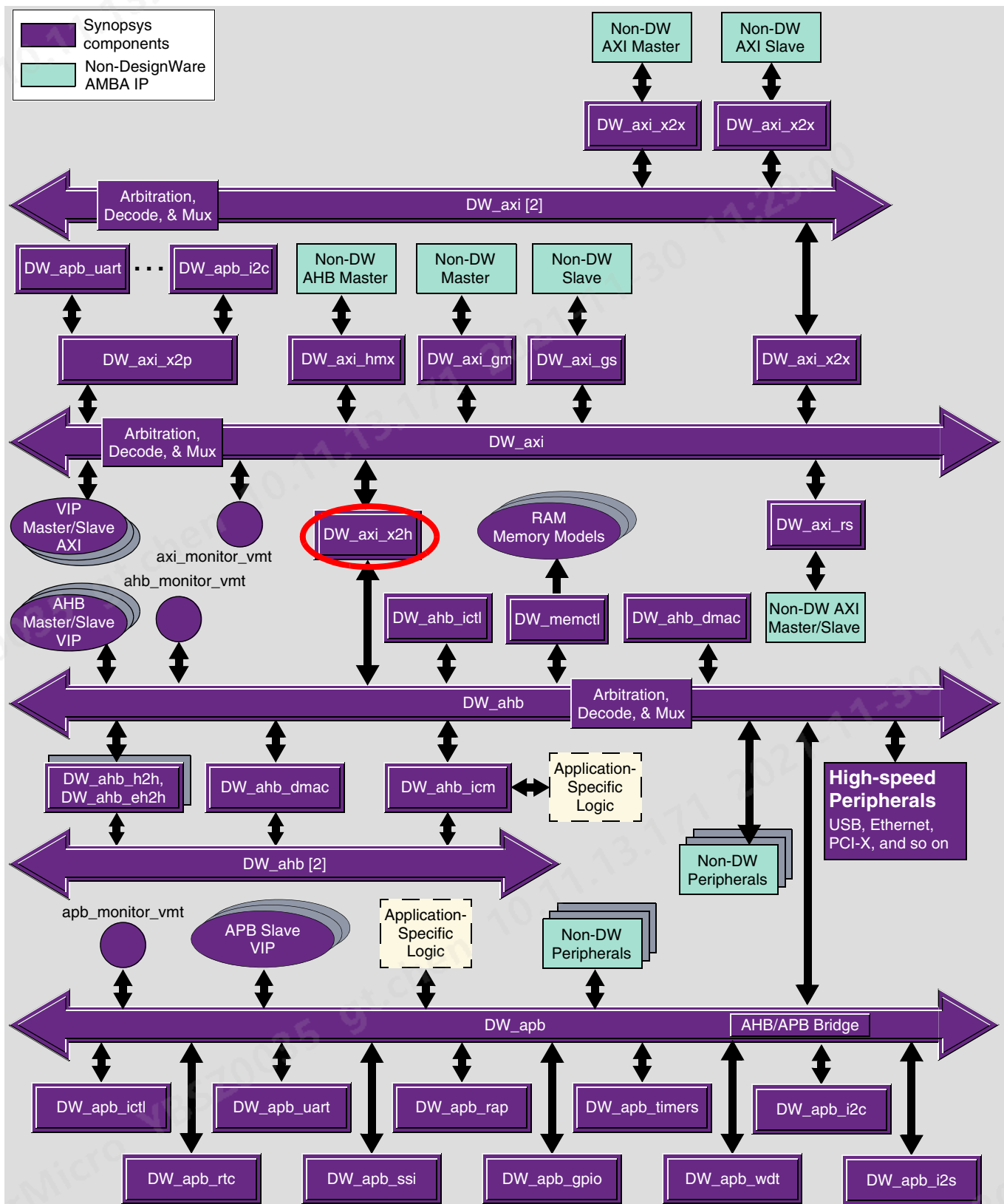
1.2 General Product Description

The DesignWare DW_axi_x2h AXI-to-AHB Module provides a method to transfer AXI-generated transactions to an AHB bus. It provides the subsystem designer with an easy way to reuse existing AHB components or full AHB/APB subsystems in new designs that use the higher-performance AXI bus.

Functionally, the DW_axi_x2h has an AXI slave interface that receives AXI transactions. Internal to the DW_axi_x2h, the AXI transactions are translated into AHB transactions. The DW_axi_x2h has an AHB master interface, which initiates the translated AHB transactions on an AHB bus.

The DW_axi_x2h has the ability to pipeline multiple AXI read and write transactions. Internal to the DW_axi_x2h are user-configurable command, data, and response buffers, which control how many outstanding transactions are possible.

1.2.1 DW_axi_x2h Block Diagram

Figure 1-2 illustrates the block diagram of the DW_axi_x2h.

Figure 1-2   DW_axi_x2h Block Diagram
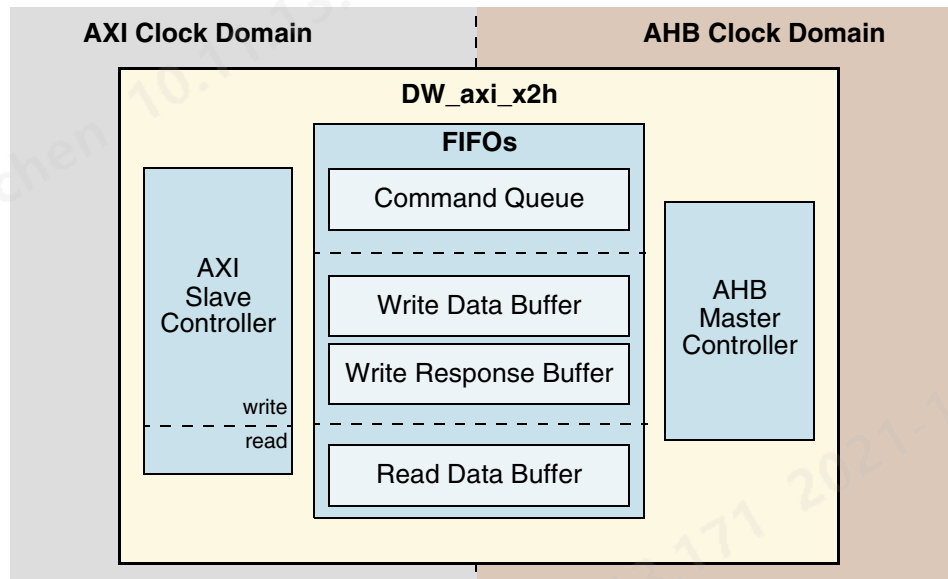
1.3 Features

The DW_axi_x2h has the following features:

- Bridge from AXI to AHB bus, allowing for easy integration of legacy AHB designs with newer AXI designs

- Complies with the following specifications:
  - AMBA 2.0 (AHB)
  - AMBA 3 AXI
  - AMBA 4 AXI

- Configurable AXI slave interface

- Configurable AHB master interface; includes AHB Lite support

- Configurable depths on command, data, and response queues

- All transactions passed through industry-standard DesignWare FIFOs

- Supports transfer downsizing; AHB data width can be the same as AXI or narrower

- User/sideband signal support on AXI read and write address channels

- Configurable synchronous or asynchronous AXI/AHB clock operations with any clock ratio

- Configuration of DesignWare AHB Lite system

Figure 1-3 illustrates how the DW_axi_x2h processes transactions as an AXI slave and an AHB master.

Figure 1-3   DW_axi_x2h Transaction Flow Diagram
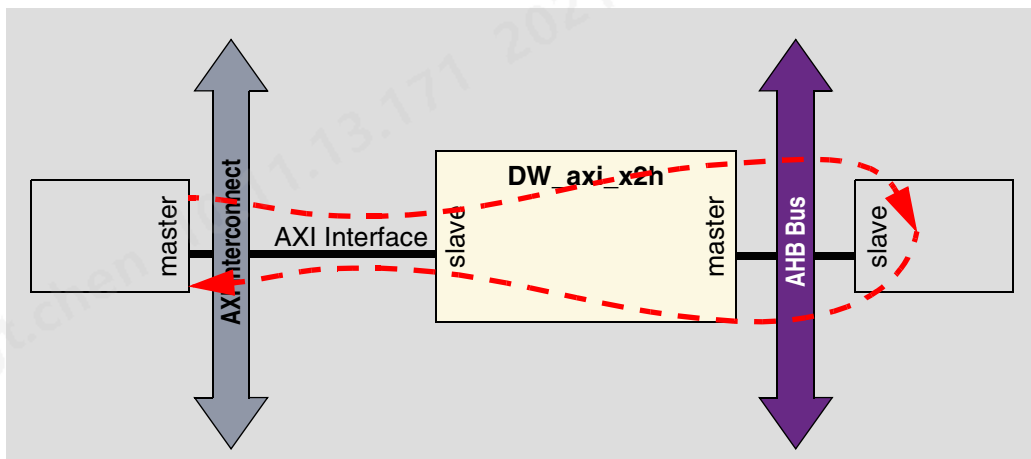
The DW_axi_x2h AXI slave interface has the following features:

- Complies with the following specifications:

  - AMBA 3 AXI

  - AMBA 4 AXI

- Supports multiple outstanding AXI transactions (configurable)

- Configurable AXI address, data, burst length, and transaction ID widths

- Supports INCR, WRAP, and FIXED AXI transfers of all legal lengths and sizes

- Converts AXI reads/writes to sequential AHB accesses

  - No write or read data re-ordering

  - No write or read data interleaving

- AXI low-power interface support

  - Transaction activity status sent to an external low-power controller (LPC) for enabling or disabling the clock.

  - Configurable maximum number of clock cycles the system must remain idle for, before entering the low-power mode.

The DW_axi_x2h AHB master interface has the following features:

- AHB 2.0 (AHB) compliant
- User-configurable as an AHB-Lite master interface
- Configurable AHB addr/data widths
- Uses defined-length bursts for AHB reads
- Uses SINGLE and unspecified-length INCR burst length types for AHB writes — unspecified length burst usage can be disabled
- Early burst termination support
- Supports split/retry responses
- Can be configured to HLOCK AHB when processing Locked Accesses from AXI

The DW_axi_x2h has the following known limitations:

- No support for Big Endian mode data addressing
- No support for data reordering — data reordering depth of 1
- No support for data interleaving — data interleaving depth of 1
- No support for exclusive accesses — external support possible
- No support for automatic connections in DesignWare Connect (can be manually connected)
- No support for subsystem simulation in DesignWare Connect

When the AXI4 interface is enabled, the following features are not applicable, as these signals are not present on the AHB bus:

- QoS
- Multiple region
- User-defined signaling

1.4   Standards Compliance

The DW_axi_x2h 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_x2h 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_x2h performs under the provided test conditions.

1.6   Where To Go From Here

At this point, you may want to get started working with the DW_axi_x2h 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, see *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_axi_x2h component, see *"Overview of the coreConsultant Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 3 AXI, and AMBA 4 AXI User Guide.*

For more information about implementing your DW_axi_x2h component within a DesignWare subsystem using coreAssembler, see *"Overview of the coreAssembler Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 3 AXI, and AMBA 4 AXI User Guide.*

2

Functional Description

This chapter describes the DW_axi_x2h.

2.1   Overview

The DW_axi_x2h maps the AXI protocol to the AHB protocol. On one side, the DW_axi_x2h acts as an AXI slave and is connected to an AXI interconnect or directly to an AXI master. On the other side, the DW_axi_x2h acts as an AHB master and is connected to an AHB interconnect. An interface between the two sides is comprised of the following:

- Common command (CMD) queue
- Write data buffer
- Read data buffer
- Write response buffer

The depths of these buffers can be configured through four separate parameters, and their widths automatically adjust according to the data and address widths configured for the AXI and AHB interfaces. For information on DW_axi_x2h parameters, see "Parameter Descriptions" on page 39.

There is an option to configure AHB Lite, which is the DesignWare implementation of AMBA 2.0 AHB-Lite. The DesignWare AHB Lite configuration does not include the following features:

- Requesting/granting protocols to the arbiter and split/retry responses from the slaves; all slaves are made non-split capable
- No arbiter as the signals associated with the component are not used: hbusreq and hgrant
- No write data, address, or control multiplexers
- Pause mode not enabled
- Default master number changed to 1
- Number of masters is changed to 1

For more information about AHB Lite, see the *DesignWare DW_ahb Databook*.

Figure 2-1 illustrates the block diagram of the DW_axi_x2h.

Figure 2-1   DW_axi_x2h Block Diagram
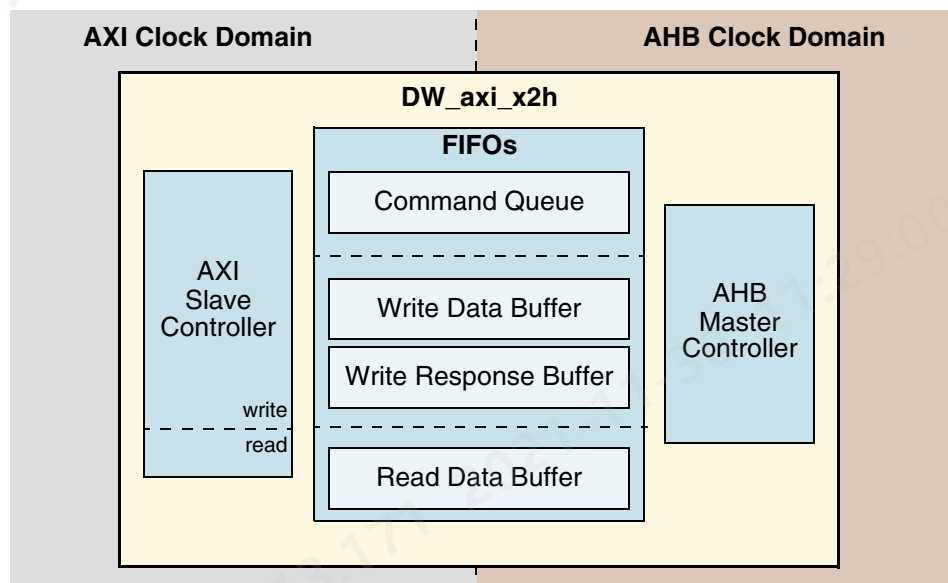
Figure 2-2 illustrates the DW_axi_x2h within the larger context between an AXI bus and an AHB bus.

Figure 2-2   AXI-to-AHB Bridge Overview
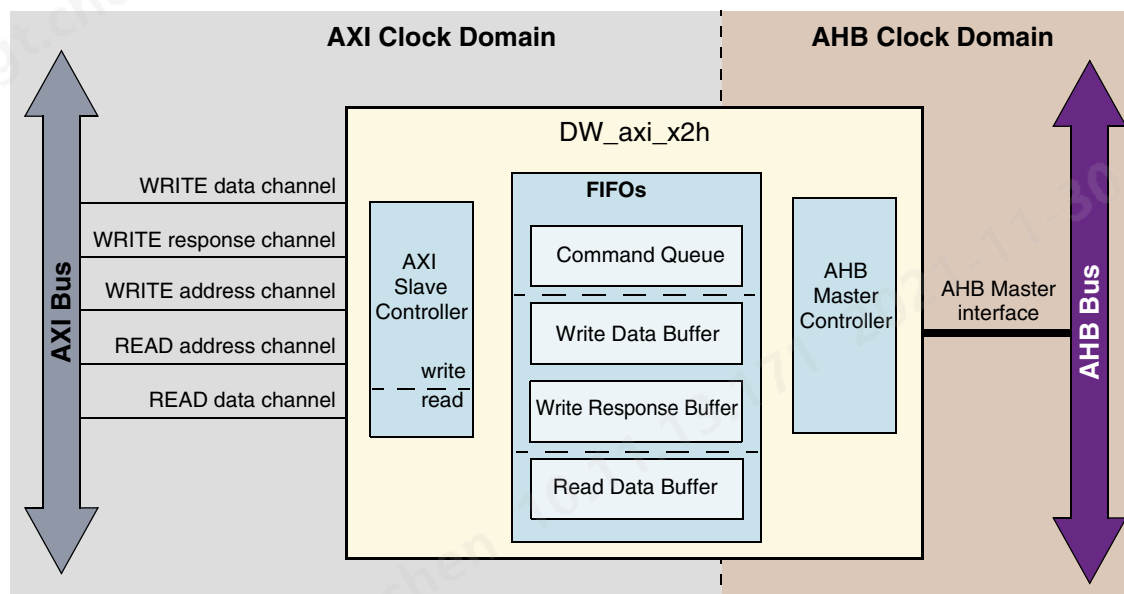
The AXI slave controller and the AHB master controller are not directly connected. All AXI/AHB bus transaction communication between the two sides passes through the DW_axi_x2h CMD queue and data buffers.

2.2 AXI to AHB Common Command (CMD) Queue

Figure 2-3 illustrates a block diagram of the CMD queue.

Figure 2-3 CMD Queue
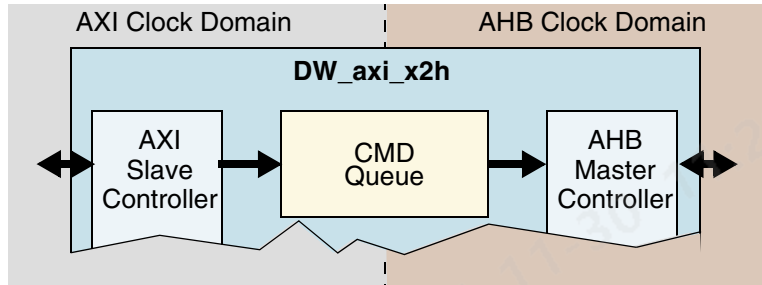
The DW_axi_x2h uses a single CMD queue for both AXI write transfers and AXI read transfers in order to pass address and control information from the AXI slave controller to the AHB master controller.

Typically the AXI slave controller pushes AXI commands into the CMD queue when they come in from the AXI read and write address channels. However, if the CMD queue completely fills up, the AXI slave controller inserts wait states on the AXI address channels by de-asserting the awready and arready signals until space becomes available to push more commands into the CMD queue.

Write and read commands may arrive from the AXI at the same time; that is, the awvalid and arvalid signals can be asserted simultaneously. When this situation arises, the AXI slave controller puts the write into the CMD queue first while stalling the read command by de-asserting the arready signal. Then the AXI slave controller puts in the read command while stalling any following write command by de-asserting the awready signal. This toggling between accepting one write and then one read continues as long as the awvalid and arvalid signals both remain asserted.

2.3 Read Transfers from AXI to AHB

The CMD queue and Read data buffer provide communication for read transactions between the AXI slave controller and the AHB master controller. All data and control communications reside in these buffers. This communication allows operations that are unique to each protocol to run without conflict. Each of the controllers sees and interfaces with only the buffers while they become full or empty.

Figure 2-4 illustrates a block diagram of the FIFOs related to the AXI-to-AHB read transactions.

Figure 2-4 Block Diagram of AXI-to-AHB Read Data Buffer
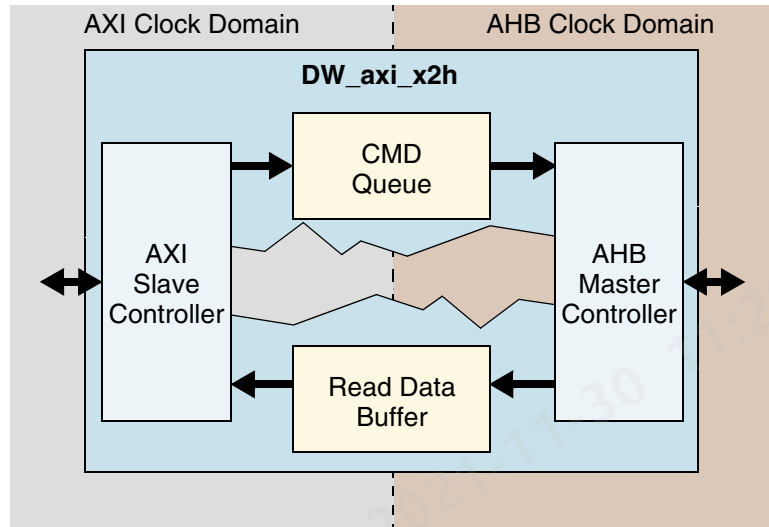
When a read command comes in from the AXI read address channel, the AXI slave controller pushes it into the CMD queue.

At some point after that, the AHB master controller pops the command out of the CMD queue, reads the data requested from AHB, and pushes the data into the read data buffer. As it does so, the AHB master controller is also required to push in an appropriate AXI response code and an appropriate value of rlast. It also echoes the AXI ID field of the AXI read command.

The AXI slave controller constantly monitors the read data buffer. Whenever it is non-empty, the AXI slave controller pops the buffer and puts the contents on the AXI read data channel.

The DW_axi_x2h architecture processes AXI read commands in order with no data interleaving. Thus, the DW_axi_x2h never returns out-of-order read data.

The following shows a typical progression of events in an AXI-to-AHB read transfer:

1. A read command comes in from the AXI read address channel.

2. When space is available in the CMD queue, the AXI slave controller pushes the address, control, and ID of the read command into the CMD queue.

3. The AHB master controller does the following:

   a. Checks the CMD queue and determines that it is not empty.

   b. Pops the read command information.

   c. Initiates the AHB arbitration and read transfer.

4. The AHB master controller pushes the data, ID, and error status into the read data buffer.

5. The AXI slave controller constantly monitors the read data buffer and, as long as it is not empty, the AXI slave controller continues to pop the read data buffer and pass its contents to the AXI read data channel.

6. As long as the read data buffer is not full, the AHB master controller continues to access data on the AHB and push the read data buffer.

7. The AHB signals the last transfer by including an RLAST flag along with the data, ID, and error status in the read data buffer.

2.4   Write Transfers from AXI to AHB

The CMD queue, Write Data buffer, and Write Response buffer provide communication for write transactions between the AXI slave controller and the AHB master controller. As with read transfers, all data and control communications for write transfers reside in these buffers.

Figure 2-5 illustrates a block diagram of the FIFOs related to the AXI-to-AHB write transactions.

Figure 2-5   Block Diagram of AXI-to-AHB Write FIFOs

When a write command comes in from the AXI write address channel, the AXI slave controller pushes it into the CMD queue.

Independent of that, whenever write data comes in from the AXI write data channel, unless the Write Data buffer is full, the AXI slave controller pushes it in. The AXI wstrb and wlast signals are also pushed in parallel with the write data. The AXI slave controller does not need to match up the write data with the associated AXI write command, and write data can be accepted before the AXI write command arrives.

The AXI slave controller always pushes into the Write Data buffer once for each AXI data beat. In cases where an AXI write does not use the full width of the AXI data bus, the AXI slave controller never compresses two or more AXI data beats into a single push into the Write Data buffer. There is always one Write Data buffer push for each AXI data beat.

Write processing begins on the AHB side when the AHB master controller pops a write command out of the CMD queue. At this point, the AHB master controller begins looking for data available from the Write Data buffer.

The AXI protocol supports "sparse" writes in which some byte lanes are written but others are not. The write data channel has a wstrb (write strobe) signal with one bit for each byte of write data width. However, because the AHB does not support sparse writes, the AHB master controller must monitor the wstrb signal coming in from the AXI through the write data buffer. Bytes are written to the AHB only when the wstrb signal is high. For those bytes with a valid wstrb signal that are non-adjacent, the AHB master controller creates multiple AHB bus transactions to complete the entire write transaction. For example, if the AXI does

a 1-beat, 32-bit write to address 0 with wstrb[3:0] = 1011, the AHB master controller would do a 2-byte write to bytes 0 and 1, skip over byte 2, and then write to byte 3.

The AHB master controller uses AHB INCR and/or SINGLE write transfers to pop and process as many AXI beats of data and WSTRB out of the Write Data buffer as are specified for the AXI write command. If the Write Data buffer runs empty, or if a stretch of sparse data with all-zero WSTRBs is encountered, the AHB master controller releases the AHB rather than holding it with BUSY cycles. As soon as more non-sparse data is available, the AHB master controller again starts writing to the AHB. The AHB master controller does not wait for a minimum threshold amount of data before initiating a write to the AHB bus.

The AHB master controller pushes an AXI response code into the Write Response buffer, along with the ID of the AXI write command after the following occur:

- Removing from the Write Data buffer the specified number of AXI beats

- All AHB writes are completed with no SPLITs or RETRYs pending

- As soon as there is space in the Write Response buffer

The AXI slave controller continually monitors the Write Response buffer. Whenever it is not empty, the AXI slave controller pops it and puts the response and ID — that is, the bresp and bid signals — onto the AXI write response channel.

The DW_axi_x2h architecture processes AXI write commands in order with no data interleaving. There can be many write commands in the CMD queue, but the AHB master controller processes one write command at a time. It requires that all data for that write command come before any data for any subsequent write command; that is, write data interleaving is not supported.

The following shows a typical progression of events in an AXI-to-AHB write transaction:
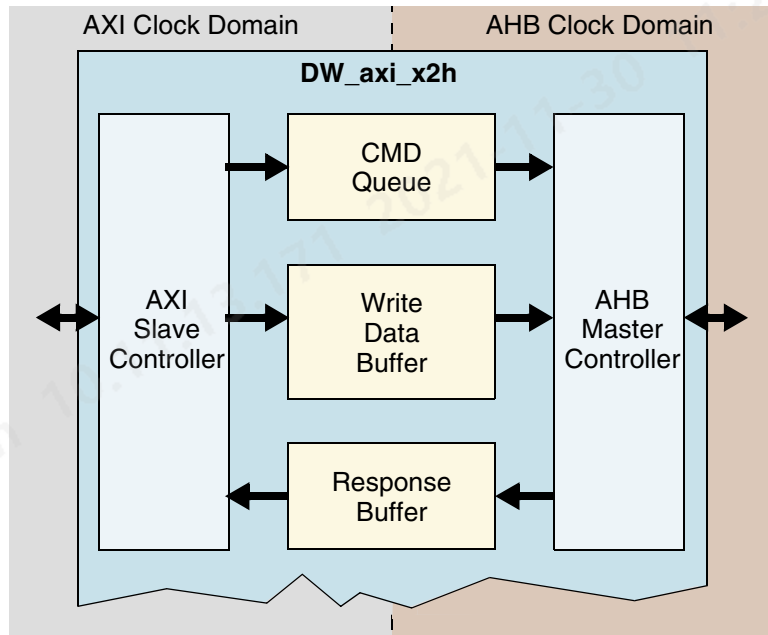
1. A write command comes in from the AXI write address channel.

2. When space is available in the CMD queue, the AXI slave controller pushes the address, control, and ID of the write command into the CMD queue.

3. As write data comes in from AXI write data channel, and as long as the Write Data buffer is not full, the AXI slave controller pushes the write data, plus the wstrb and wlast signals, into the Write Data buffer.

4. The AHB master does the following:

   ❑ Checks the CMD queue and determines that it is not empty.

   ❑ Pops the write command information.

5. At this point, the AHB master controller begins monitoring the Write Data buffer. As data becomes available, the AHB master controller pops it out and evaluates the WSTRB field. All bytes with WSTRB = 1 are written to the AHB; all with WSTRB = 0 are discarded.

6. The AHB master controller repeats step 5 until the amount of write data scheduled for the current AXI write command has been popped out of the Write Data buffer and either written to the AHB or discarded. Through all of this, the AHB master controller keeps track of whether it received any ERROR response while writing to the AHB, and also whether any WLAST field value popped out of the Write Data buffer is not as expected according to the scheduled transfer count.

7. Upon completing the scheduled count, as soon as space is available in the Write Response buffer, the AHB master controller pushes an appropriate AXI response code into it. It also echoes the AXI ID field of the AXI write command.

8. The AXI slave controller continually monitors the Write Response buffer. Whenever the buffer is not empty, the AXI slave controller pops it and puts the response and ID it obtained — bresp and bid signals — onto the AXI write response channel.

2.5   AHB Command Usage

The following discusses how read and write commands are used on the AHB.

2.5.1   Read Command Usage

The AHB master controller can issue SINGLE reads on the AHB. Depending on the depth of the Read Data buffer, the AHB master controller can issue INCR4, INCR8, and INCR16 bursts on the AHB. The AHB master controller can also issue INCR reads of undefined length if the X2H_USE_DEFINED_ONLY parameter is not set.

👉 **Note**   In general, there is not a one-to-one correspondence between read transactions from the AXI and read commands that the AHB master controller issues. Often the AHB master controller must use multiple AHB read commands to get all the data for a single AXI read transaction. The following scenarios illustrate some conditions under which this can happen.

- **Example 1** – An AXI transaction occurs for an INCR read, 12 beats long, with an AXI/AHB data width of 32 bits. There is no 12-beat read command on AHB. In this situation, the AHB master controller would likely choose an INCR8 and an INCR4, or 3 INCR4s. Alternatively, the AHB master controller can use an undefined-length INCR or 12 SINGLE reads.

- **Example 2** – An AXI transaction occurs for an INCR read, 16 beats long, with an AXI/AHB data width of 32 bits, but the Read Data buffer is configured to be only 8 deep. In this case, the AHB would never issue an INCR16 read, since there would never be enough space in the Read Data buffer for all 16 data beats. Instead, the AHB master controller can perform two INCR8 reads, four INCR4 reads, an undefined-length INCR, or 16 SINGLE reads.

- **Example 3** – The AXI data bus is 128 bits wide, and an AXI transaction occurs for an INCR read, 8 beats long. However, the AHB data bus is only 32 bits, so this transfer must be downsized so that there are 4 beats on the AHB for each beat of the AXI transaction, for a total of 32 beats. To perform this read, the AHB master controller might use two INCR16 reads, four INCR8 reads, eight INCR4 reads, an undefined-length INCR, or 32 SINGLE reads.

- **Example 4** – An AXI transaction occurs for a 1-beat read with a data width of 32 bits, but the AXI address (araddr) ends with araddr[1:0] = 2'b01; that is, it is not aligned to a 32-bit boundary. "Unaligned" transactions like this are allowed on the AXI, but not on the AHB. The AHB master controller must perform a 1-byte read of the 2'b01 address, and then a 2-byte read of an address incremented to 2'b10.

- **Example 5** – An AXI transaction occurs for an INCR read, 16 beats long, starting at an address which causes it to cross a 1K boundary; AXI transactions can cross 1K boundaries, but not 4K boundaries. The AHB master controller must perform a combination of reads in order to get to the boundary, and then do more reads to complete the AXI transaction, making sure the 1K boundary is left at the break between two AHB read commands.

- **Example 6** – An AXI transaction occurs for an INCR read, 128 beats long, with an AXI/AHB data width of 32 bits in AXI4 mode. In this case, the AHB would never issue an INCR128 read as there is no 128-beat read command on AHB. The AHB master controller would choose eight INCR16 reads, 16 INCR8 reads, 32 INCR4 reads, an undefined-length INCR or 128 SINGLE reads.

2.5.1.1 Wrapping Burst Reads

The AHB master Controller does not issue "wrap" transfers on the AHB. If an AXI transaction occurs for a WRAP read, the AHB master controller issues a set of AHB transfers up to the wrap boundary, and then another set of AHB transfers starting at the wrapped address to complete the transfer.

2.5.1.2 Fixed Burst Reads

It is possible for the AXI to send a FIXED read command, which specifies that the same location should be read several times in a row — helpful when reading from a FIFO. The AHB master controller repeats the read — or sequence of reads for each AXI beat, if required — on the AHB as many times as specified by the AXI FIXED command; if downsizing or an unaligned address is involved, each AXI beat would require a sequence of AHB reads.

2.5.1.3 Read Command Downsizing

Except when unaligned addresses are involved, the transfer size used on the AHB is the smaller of:

- Size of the AXI transfer

- AHB data bus width

Thus, an AXI transfer is downsized if it is too wide for the AHB, but the reverse is not true if the AXI transfer is narrower than the AHB data width; that is, upsizing does not occur.

2.5.2 Write Command Usage

The AHB master controller can issue SINGLE writes on the AHB. If the X2H_USE_DEFINED_ONLY parameter is not set, the AHB master controller can issue INCR writes of undefined length on AHB. When writing to the AHB, the AHB master controller never uses defined-length AHB bursts such as INCR4, INCR8, or INCR16. This enables the AHB master controller to check the WSTRB field bits dynamically to determine whether each byte of AXI write data should be written to the AHB, or whether it is sparse data to be discarded and thus cannot commit up-front to write a stated length of data.

The AHB master controller performs writes on the AHB under the following conditions:

- Valid AXI write command in the CMD Queue

- AXI write data in the Write Data buffer

- WSTRB field of the data in the Write Data buffer indicates that it is not "sparse" (if there is data available in the Write Data Buffer, but it is sparse, the AHB master controller discards it)

When processing an AXI write transaction of several beats, there is no minimum threshold amount of data in the Write Data buffer for the AHB master controller to start writing. The write starts as soon as data is available.

If the Write Data buffer empties while the AHB master controller is processing an AXI write transaction of several beats, or if there is a stretch of "sparse" data, the AHB master controller releases the AHB bus until more data is available.

2.5.2.1 Wrapping Burst Writes

Note that the AHB master Controller does not issue "wrap" transfers on the AHB. If an AXI transaction occurs for a WRAP write, the AHB master controller issues a set of AHB transfers up to the wrap boundary, and then another set of AHB transfers starting at the wrapped address to complete the transfer.

2.5.2.2 Fixed Burst Writes

It is possible for the AXI to send a FIXED write command, which specifies that the same location should be written several times in a row — helpful when writing to a FIFO. The AHB master controller repeats the write — or sequence of writes for each AXI beat, if required — on the AHB as many times as specified by the AXI FIXED command.

2.6 Data Width Adaptation

The AXI read and write data bus widths must always be configured to be equal to or greater than the AHB data bus width. This section discusses how data width is adapted between the AHB bus and the AXI data buses.

2.6.1 Read Data Width Adaptation

The Read Data buffer carries read data and control fields from the AHB domain back to the AXI domain, with the widest field being the read data field itself. The Read Data buffer is always configured so that the width of its read data field matches the width of the AXI read data bus.

The AHB master controller pushes data into the Read Data buffer in such a way that the AXI slave controller can simply pop data out of the Read Data buffer and put it directly onto the AXI read data bus, one Read Data buffer pop per AXI beat. When the AHB master controller needs to perform multiple AHB reads to obtain the data for a single AXI beat due to downsizing or an unaligned address, the AHB master controller concatenates data from multiple AHB beats before pushing it into the Read Data buffer.

Figure 2-6 illustrates the relationships between the AHB data bus, AHB master controller, Read Data buffer, and AXI read data bus.

Figure 2-6   Example of Read Data Width Adaptation
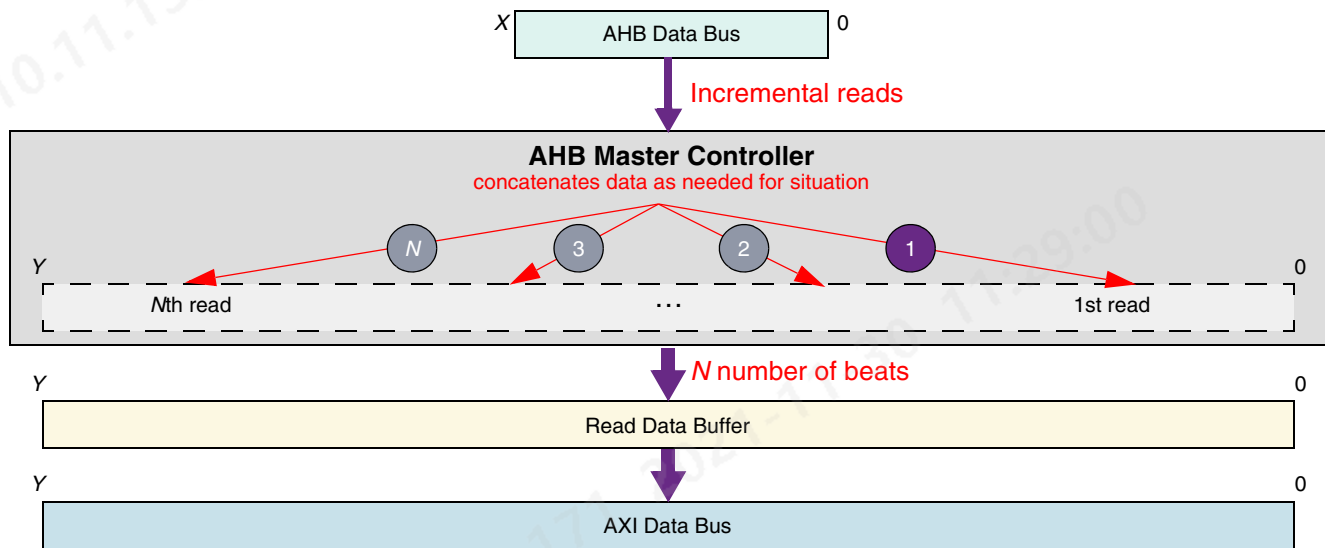
The following example illustrates read data width adaptation. For this example, not only is the AHB data bus less wide than the AXI read data bus, but the AXI transaction is for data transfers that are not as wide as the AXI read data bus. If the AXI read transaction is not as wide as the full width of the AXI read data bus, the AHB master controller aligns the data into the correct byte lanes when pushing read data into the Read Data buffer. Suppose the following:

- AXI read data bus (and therefore the data field of the Read Data buffer) is 128 bits wide, and the AHB data bus is 32 bits wide.

- AXI INCR read comes in to read four beats of 64 bits per beat; note that this is both bigger than the AHB bus width and smaller than the AXI bus width. This helps to further illustrate the data width adaptation capability.

- Starting address is 0.

To obtain 4x64 bits, the AHB master controller must downsize by performing eight 32-bit reads in order to accommodate the AHB. As long as the Read Data buffer has enough available buffer space for it, the AHB master controller can accomplish this through a 32-bit INCR8 read on the AHB.

Using the addresses from which they are read, the eight 32-bit data obtained from the AHB can be referred to as AHB-0, AHB-4, AHB-8, AHB-C, AHB-10, AHB-14, AHB-18, and AHB-1C. The AHB master controller performs four pushes into the Read Data buffer, since the AXI transaction is comprised of four 64-bit beats. These pushes are illustrated in Figure 2-7.

Note When a transfer takes place on an AXI bus which has a data bus width that is larger than the AXI transaction data width, either the high bits or the low bits of any given beat of AXI data is unused.

Figure 2-7   Results of Pushes for 128-bit-wide Data Width Adaptation
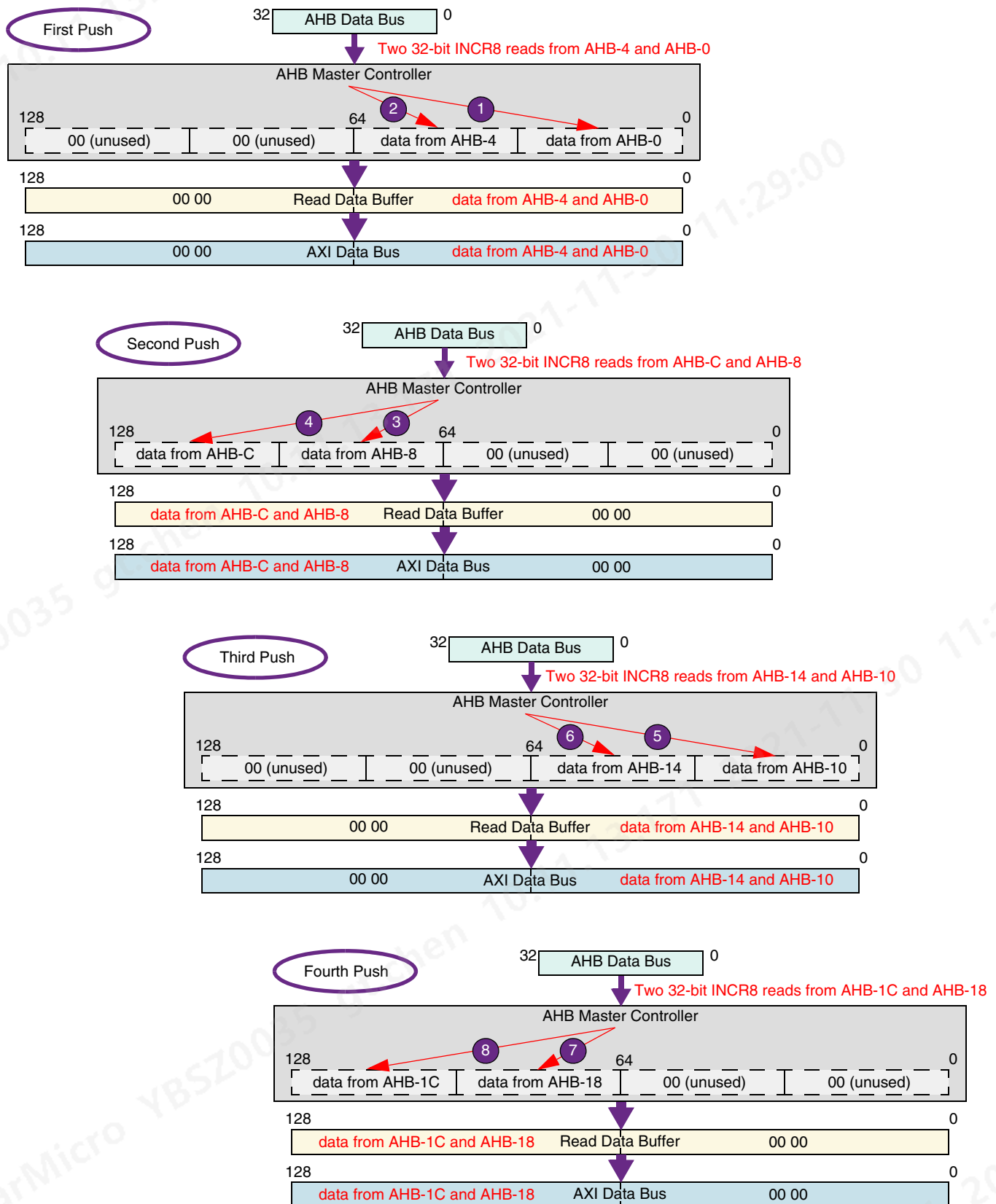
2.6.2 Write Data Width Adaptation

The Write Data buffer carries the write data, strobe, and last fields from the AXI domain to the AHB domain. The Write Data buffer is always configured so that the width of its write data and strobe fields match those of the AXI write data bus; that is, even if the AXI write data bus is configured to a different data width than the AHB bus, the Write Data buffer is the same data width as the AXI write data bus.

If necessary, the AHB master controller splits each AXI beat of write data into multiple AHB writes if downsizing is necessary or if not all of the AXI data can be written due to sparse data or an unaligned start address. If the AXI write transaction transfer size is less than the AXI write data bus width, the AHB master controller only uses data from the appropriate byte lanes and ignores the unused byte lanes.

Write data width adaptation is analogous to the issues described in "Read Data Width Adaptation" on page 27, except the direction of data flow is reversed.

2.7 Response and Error Handling

The following discusses how the DW_axi_x2h handles read and write responses.

2.7.1 Read Response and Error Handling

Reads that the AHB master controller performs on the AHB bus are handled differently, depending on the following responses:

- OKAY – Successful transaction; read terminated

- ERROR – Unsuccessful transaction; read terminated

- SPLIT – Split transaction necessary; read attempted repeatedly until OKAY or ERROR received

- RETRY – Transaction must be retried; read attempted repeatedly until OKAY or ERROR received

As the AHB master controller pushes data into the Read Data buffer, it also includes an AXI response with each beat. If the data comes from the AHB with an OKAY response, then the AHB master controller transfers the OKAY response with the data. It is possible that the data being pushed into the Read Data buffer came from more than one AHB read due to downsizing or an unaligned address. In this case, if any data from the AHB is accompanied with an ERROR, then an AXI response of SLVERR is sent to the Read Data buffer with the data.

If an ERROR response occurs during a burst, the AHB master controller continues reading from the AHB and transferring the data.

If a read transaction is comprised of multiple beats on the AXI and AHB, a single ERROR response for any of the AHB beats means that only the associated AXI beat of data is marked with SLVERR; that is, the error condition does not affect more than one AXI beat, and none of the remaining AXI beats of data are marked with SLVERR.

2.7.2 Write Response and Error Handling

After completing an AXI write transaction, the AHB master controller pushes an AXI write response into the Write Response buffer. The AXI write transaction is complete when the AHB master controller has done both of the following:

1. Popped all data for the write transaction out of the Write Data buffer

2. Completed all AHB writes; that is, all writes have been issued and terminated on the AHB by an OKAY or ERROR response

As with read responses, the AHB master controller handles writes differently with the following responses:

■ OKAY – Successful transaction; write terminated

■ ERROR – Unsuccessful transaction; write terminated

■ SPLIT – Split transaction necessary; write attempted repeatedly until OKAY or ERROR received

■ RETRY – Transaction must be retried; write attempted repeatedly until OKAY or ERROR received

When the AHB master controller pushes data into the Write Response buffer, if any of the writes on the AHB are terminated with an ERROR, then the AHB master controller provides an SLVERR response.

If an ERROR response occurs during a burst, the AHB master controller continues writing all remaining transaction write data to the AHB.

If all writes on the AHB are terminated with OKAY, a "last check" is performed using the WLAST field from the Write Data buffer. The AXI slave controller sends the wlast signal through the Write Data buffer along with the write data and write strobe signals. As it processes the AXI write transaction, the AHB master controller keeps track of the length of the transaction, which enables it to know when the last AXI beat should be popped out of the Write Data buffer. Thus, if a data beat pops out and is accompanied by the WLAST field set, but does not match the length of the transaction, an SLVERR response is pushed into the Write Response buffer at the end of the transaction, signifying that the wrong beat is marked as the last one. If this happens, the too-early WLAST does not terminate the transaction, and a missing WLAST at the end does not extend the transaction. This error condition indicates a fundamental error in the subsystem, which the AXI slave controller cannot control.

If all AHB writes terminate with OKAY and WLAST occurs on the correct beat, then an OKAY response is pushed into the Write Response buffer. The AHB master controller then passes the ID of the AXI write command along with the response code.

2.8 Atomic Accesses

The following subsections discuss details about atomic accesses, which are exclusive accesses (for semaphore type operations) and locked accesses (for single-master access to a slave region).

2.8.1 Exclusive Accesses

Exclusive accesses are not supported in the current version of the DW_axi_x2h. If exclusive access support is required, this is possible by adding extra logic external to the DW_axi_x2h. There is a description of how this can be done in the *AMBA AXI and ACE Protocol Specification*. As defined in this specification, the DW_axi_x2h is a "single-ported slave."

2.8.2 Locked AXI-to-AHB Transactions

When a read or write transaction from the AXI to the AHB occurs with "Lock Type" set to "Locked access," the information is passed across the CMD queue, and the AHB master controller can optionally lock and hold the AHB bus using HLOCK. You can set this feature using the X2H_PASS_LOCK parameter.

> **Note** This feature is available only when the AXI 3 interface is selected (X2H_AXI_INTERFACE_TYPE = 0).

Once the AHB master controller locks the AHB, it holds it until another command comes through the CMD queue with "Lock Type" *not* set to "Locked access." This situation corresponds to the "final unlocking transaction" of the AXI locked sequence, described in the *AMBA AXI specification*, and occurs while the AHB is locked; the lock is released as soon as the transaction completes. If a "final unlocking transaction" is broken into several AHB commands, the AHB master controller releases the AHB lock only after one of the following occurs:

- Reads the final piece of data to be placed in the Read Data buffer with RLAST set

- Successfully writes the final piece of data so that the write response can be placed in the Write Response buffer

Completely sparse write transactions — that is, writes with all WSTRB = 0, which write nothing to the AHB — can be used as locking or unlocking accesses.

2.9 Cache/Prot Signal Mapping

The AXI specification defines AR/AWCACHE[3:0] and AR/AWPROT[2:0] as "additional control signals," which are an extension of the AHB HPROT[3:0] signal. For both reads and writes, the AXI signals are mapped to the AHB as shown in Table 2-1.

Table 2-1   AXI to AHB Signal Mapping

AXI Signals	AHB Signals
AR/AWCACHE[3] (AXI write allocate)	Not mapped
AR/AWCACHE[2] (AXI read allocate)	Not mapped
AR/AWCACHE[1] (AXI3 cacheable/AXI4 modifiable)	HPROT[3] (cacheable)
AR/AWCACHE[0] (AXI bufferable/non-bufferable)	HPROT[2] (bufferable)
AR/AWPROT[2] (AXI data/instruction access)	HPROT[0] (data/opcode access)
AR/AWPROT[1] (AXI secure/non-secure)	Not mapped
AR/AWPROT[0] (AXI privileged/non-privileged)	HPROT[1] (privileged/user access)

A value of 1 on the AR/AWPROT[2] bit specifies an instruction fetch, while a 0 specifies a data fetch. This bit is inverted to become HPROT[0] on the AHB, which uses 0 for an opcode fetch and 1 for data; the AHB master controller performs this inversion.

2.10 User/Sideband Signals

2.10.1 Overview of User/Sideband Signals

DW_axi_x2h provides user/sideband signals on AXI read and write address channels that enable transfer of extra information outside of the AXI protocol specification. When the interface type is selected as AXI3 (X2H_AXI_INTERFACE_TYPE=0), additional information can be sent through the awsideband and arsideband sideband signals. When AXI4 interface is selected (X2H_AXI_INTERFACE_TYPE=1), additional information can be sent through the awuser and aruser user signals.

The value of awsideband/awuser or arsideband/aruser is directly transmitted as mid signal on the AHB master interface.

The X2H_MID_WIDTH parameter specifies the width of awsideband/awuser, arsideband/aruser and mid signals. When set to 0, these signals are removed from DW_axi_x2h.

No endian mapping is performed on the sideband and user signals as they pass through the DW_axi_x2h.

Figure 2-8 shows AXI transaction mapped to AHB transaction in DW_axi_x2h with sideband signals.

Figure 2-8   Transaction in DW_axi_x2h with Sideband Signals
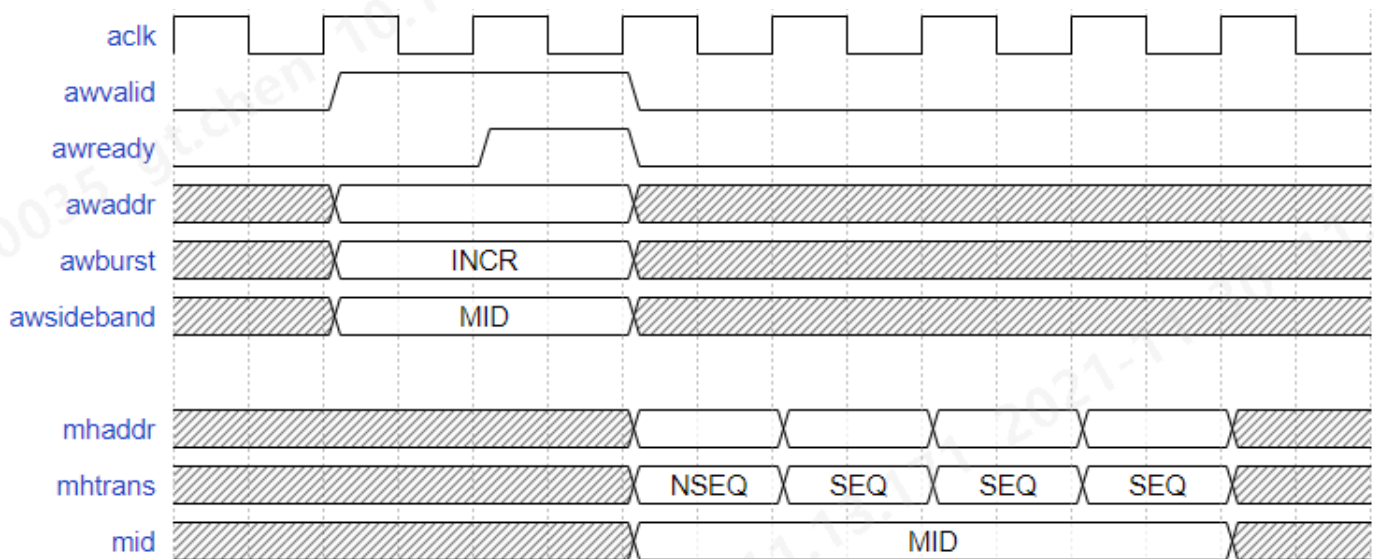
2.10.2 Setting the User/Sideband Signals

Specify the user/sideband bus width for the DW_axi_x2h, using the Specify Configuration activity in coreConsultant.

2.10.3 Signals Related to User/Sideband Signals

The following are the signals related to User/Sideband signals:

- awsideband [(X2H_MID_WIDTH-1):0]

- arsideband [(X2H _MID_WIDTH-1):0]

- awuser [(X2H _MID_WIDTH-1):0]

- aruser [(X2H _MID_WIDTH-1):0]

- mid[(X2H _MID_WIDTH-1):0]

For more information about these signals, see the Signal Descriptions chapter.

2.11 Low-Power Interface

You can configure the DW_axi_x2h to include an AXI low-power handshaking interface. This interface allows the following:

- DW_axi_x2h informs the system low-power controller (LPC) when it has no outstanding transactions

- LPC requests the DW_axi_x2h to enter into a low-power state

You can include this low-power interface in your design by setting the X2H_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_x2h to acknowledge a request to enter a low-power state; asserted by DW_axi_x2h to acknowledge a request to exit a low-power state

- cactive output – de-asserted by DW_axi_x2h when the clock can be removed after the next positive edge; csysack must also be de-asserted

👉 **Note**   When X2H_LOWPWR_HS_IF = 1, there are limits to the maximum number of outstanding read and write transactions set by the X2H_MAX_PENDTRANS_READ and X2H_MAX_PENDTRANS_WRITE parameters, respectively. This limit is applied at the DW_axi_x2h slave port by stalling the channel prior to the channel FIFO. Because of this, you should set a 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_x2h to enter a low-power state by de-asserting the csysreq signal.

2. Since the DW_axi_x2h 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_x2h has no outstanding transactions. If you set the X2H_LOWPWR_NOPX_CNT parameter to X in coreConsultant, the cactive signal de-asserts after X cycles, during which there are no outstanding transactions, have elapsed.

4. DW_axi_x2h 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; this is illustrated at P1 in Figure 2-9.

> ☞ **Note**   Entry to a low-power state happens at any cycle in which the mentioned conditions are satisfied.

While in a low-power state, the awready, wready, and arready 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 (X2H_LOWPWR_NOPX_CNT = 5)
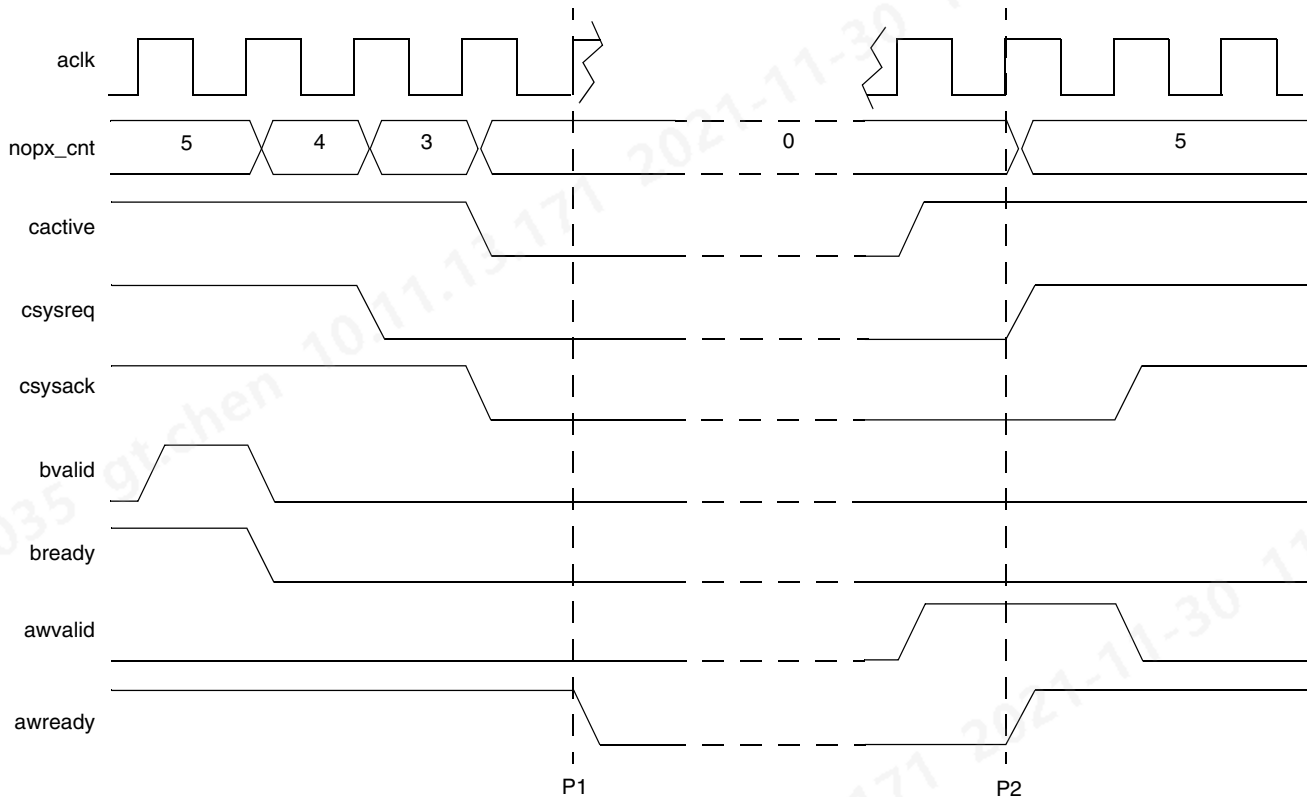
The DW_axi_x2h exits a low-power state when the cactive signal goes high and is sampled at the positive edge of aclk; illustrated at P2 of Figure 2-9.

An exit from low-power can be initiated by:

- LPC asserting the csysreq signal, which causes the DW_axi_x2h to assert the cactive signal, illustrated in Figure 2-11.

- A master asserting the awvalid or arvalid signals of the DW_axi_x2h, which causes the DW_axi_x2h to assert the cactive signal, illustrated in Figure 2-9.

Figure 2-10 shows the DW_axi_x2h 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 signal asserts, which keeps the cactive signal asserted. The LPC samples at P1 that the DW_axi_x2h has rejected the entry to low-power mode, and therefore must not remove the clock.

Figure 2-10   DW_axi_x2h Low-Power Interface Rejects Low-Power Entry
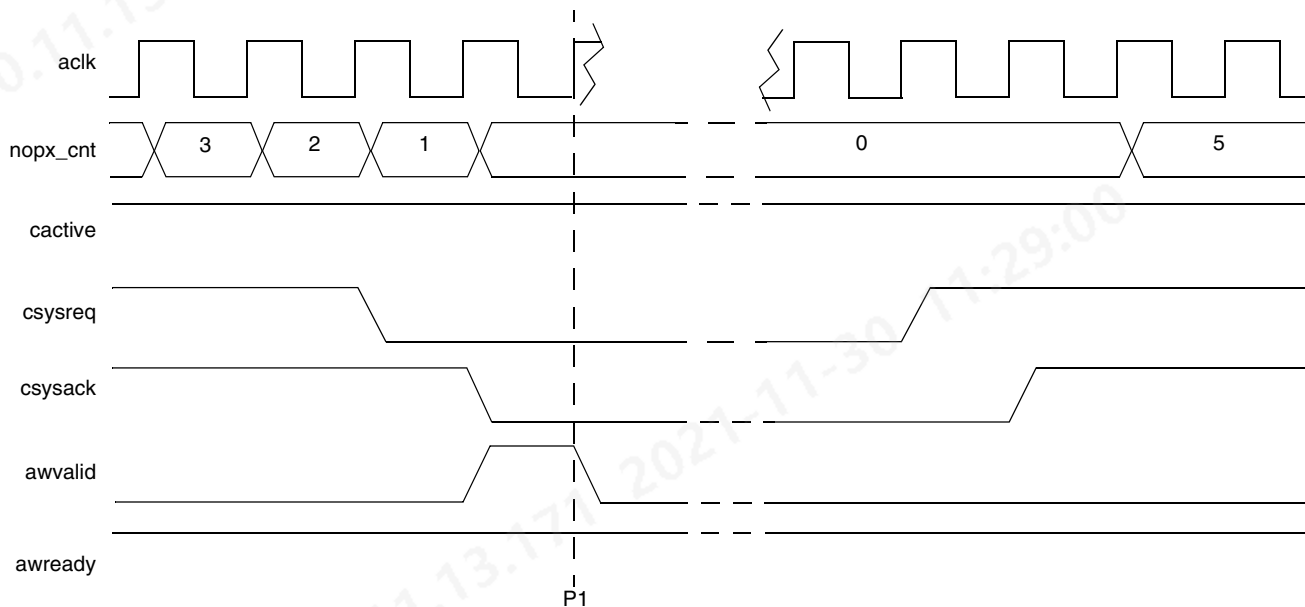
2.11.1   cactive Signal De-assertion

The cactive signal de-asserts the X2H_LOWPWR_NOPX_CNT parameter *aclk* cycles after the last pending transaction completes. If the csysreq signal de-asserts while the component is counting down to 0 from *X2H_LOWPWR_NOPX_CNT*, the cactive signal will de-assert on the next cycle.

After the positive edge of the aclk signal where the cactive signal and the csysack signal are sampled low, the low-power controller (LPC) may remove the clocks from the DW_axi_x2h (P1 in Figure 2-9 on page 35). At this point the bready and awready signals are driven low.

2.11.2   cactive Signal Assertion

The cactive signal asserts combinatorially when the awvalid or arvalid signals are asserted, at which point it remains asserted until all outstanding transactions have completed and *X2H_LOWPWR_NOPX_CNT* cycles have elapsed.

👉 **Note**   Early write data does 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_x2h asserts 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_x2h keeps the cactive signal asserted for *X2H_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
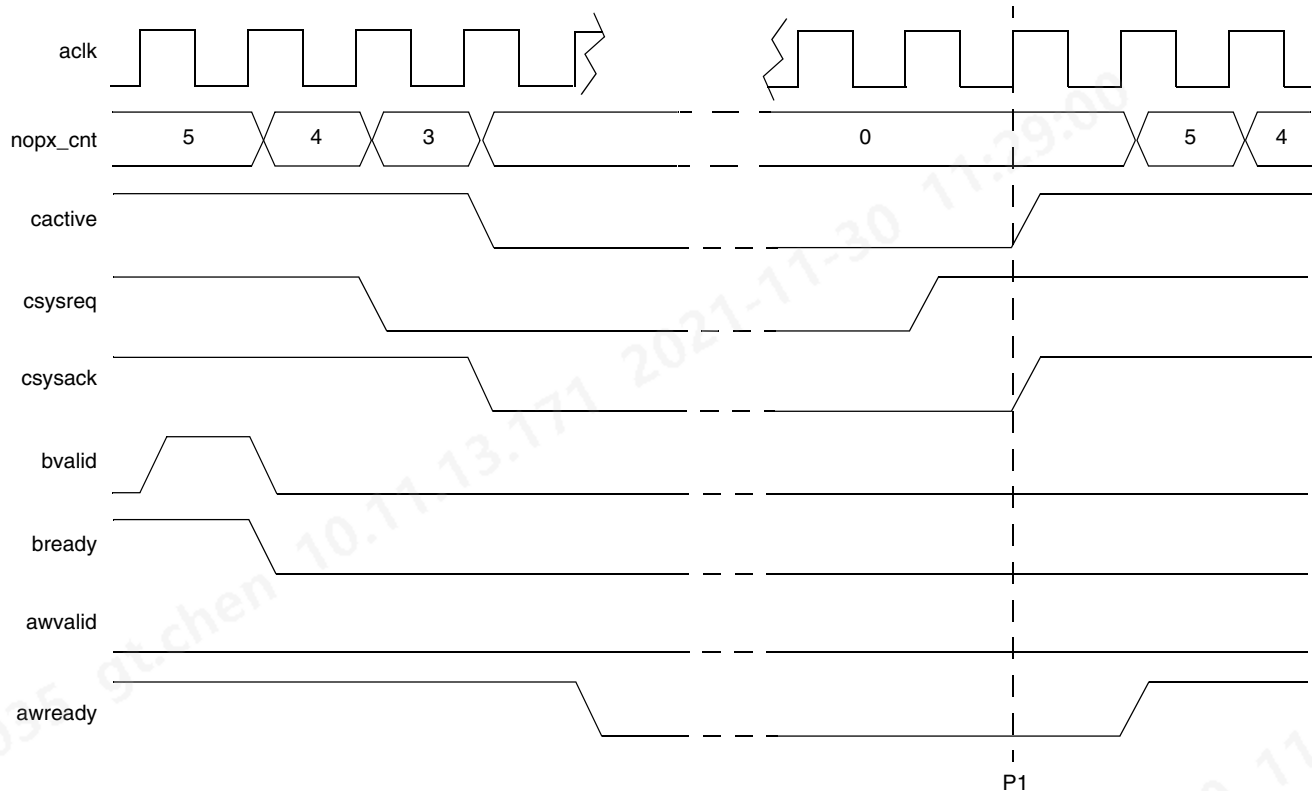
> **Note**   When coming out of reset, if the awvalid and arvalid signals equal 0:
> - cactive signal equals 0, if X2H_LOWPWR_NOPX_CNT parameter equals 0
> - cactive signal equals 1, if X2H_LOWPWR_NOPX_CNT parameter is greater than 0 and it de-asserts when *X2H_LOWPWR_NOPX_CNT* clock cycles have elapsed

2.12   Clock Adaptation

The DW_axi_x2h has two clock inputs:

- aclk – associated with the AXI bus
- mhclk – associated with the AHB bus

The design of the DW_axi_x2h allows aclk and mhclk to be fully asynchronous. However, if the two buses operate at the same frequency so that aclk and mhclk can be driven with the same clock signal, you can configure the DW_axi_x2h to take advantage of this by eliminating synchronization between the two clock domains; this saves both gates and latency.

The X2H_CLK_MODE parameter — which determines whether the AXI clock and AHB clock are asynchronous, synchronous, or the same — properly configures the DW_axi_x2h for the clocking

environment in which it operates. For more information about choosing the right clocking mode for your design, see "Clocking Configuration" on page 78.

2.13 Reset Input Usage

The DW_axi_x2h has two active-low reset inputs:

- aresetn – input from the AXI side

- mhresetn – input from the AHB side

These inputs are used to asynchronously reset registers inside the component. The aresetn resets registers that are clocked by the AXI clock (aclk), and mhresetn resets registers that are clocked by the AHB clock (mhclk).

It is assumed that both the reset inputs are asserted and de-asserted at the same time with de-assertion synchronous to the corresponding clocks. De-assertion may not be exactly at the same time as there may be delays associated with respect to the synchronization to each clock domain.

Note DW_axi_x2h does not support asserting of one reset while the other is de-asserted; doing so results in unpredictable behavior.

2.14 AHB Lite

You can use the X2H_AHB_LITE parameter to specify the DW_axi_x2h as an AHB Lite master; this causes the AHB master controller to have fewer gates. Under AHB Lite mode, the mhgrant and mhbusreq signals are not included in the design and split/retry responses, and early-burst terminations are not supported.

Only the AHB master controller is affected if you configure the DW_axi_x2h for AHB Lite; the basic, multiple-FIFO-based architecture of the DW_axi_x2h is not modified. If you use the X2H_AHB_LITE parameter, you may consider selecting shallower FIFO depths, since AHB throughput should improve if there is only one AHB master on the AHB bus. For more information on the parameters that affect FIFO depths, see "Parameter Descriptions" on page 39.

For more information on AHB Lite, see "Functional Description" chapter in the *DesignWare DW_ahb Databook*.

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 configuration options for this component.

3.1 Parameters

Table 3-1   Parameters

Label	Description
Use DesignWare Foundation Synthesis Library	The component code utilizes DesignWare Foundation parts for optimal Synthesis QoR. Customers with only a DesignWare license must use Foundation parts. Customers with only a Source license, cannot use Foundation parts. Customers with both Source and DesignWare licenses have the option of using Foundation parts. **Values:** ■ false (0) ■ true (1) **Default Value:** True if DesignWare License is available; False if DesignWare License is not available **Enabled:** Parameter is enabled if customer has both Source and DesignWare licenses **Parameter Name:** USE_FOUNDATION
Select AXI Interface Type?	Select the AXI Interface Type as AXI3 or AXI4. By default, the DW_axi_x2h supports the AXI3 interface. **Values:** ■ AXI3 (0) ■ AXI4 (1) **Default Value:** AXI3 **Enabled:** Parameter is enabled if customer has both Source and DesignWare licenses **Parameter Name:** X2H_AXI_INTERFACE_TYPE
Address Bus Width	Read and write the address bus width of the AXI system to which the bridge is attached as an AXI slave. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2H_AXI_ADDR_WIDTH
Data Bus Width	Read and write the data bus width of the AXI system to which the bridge is attached as an AXI slave. **Note**: The data bus width for the AXI slave interface must be greater than or equal to the data bu width of the AHB master interface. **Values:** 32, 64, 128, 256 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2H_AXI_DATA_WIDTH

Table 3-1   Parameters (Continued)

Label	Description
AXI ID Width	Width of read and write ID field of the AXI system to which the bridge is attached as an AXI slave. ID field enables a master to issue separate, ordered transactions. For information on how the ID field is used in read and write transfers, see "Read Transfers from AXI to AHB" and "Write Transfers from AXI to AHB" sections in the DW_axi_x2h Databook. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 **Default Value:** 16 **Enabled:** Always **Parameter Name:** X2H_AXI_ID_WIDTH
AXI Burst Length Width	Select width of the AXI AWLEN and ARLEN burst count fields. If X2H_AXI_INTERFACE_TYPE = AXI3, then the value of the X2H_AXI_BLW parameter is always 4. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** X2H_AXI_INTERFACE_TYPE==1 **Parameter Name:** X2H_AXI_BLW
Write Interleaving Depth	Write data interleaving depth. Specifies the number of addresses that can be pending on the slave interface. Write interleaving allows the slave interface to accept interleaved write data with different write address ID values. Currently set to 1 and is not user-definable in this release. **Values:** 1, 2 **Default Value:** 1 **Enabled:** 0 **Parameter Name:** X2H_WRITE_DATA_INTERLEAVING_DEPTH
Exclusive Address Depth	Number of exclusive accesses supported. For more information, see "Atomic Accesses" in the DW_axi_x2h Databook. (The current version of the design does not support any Exclusive Accesses). **Values:** 0, ..., 4 **Default Value:** 0 **Enabled:** 0 **Parameter Name:** X2H_EXCLUSIVE_ACCESS_DEPTH
Data Bus Endianness	Data bus endianness of the AXI system to which the bridge is attached as an AXI slave. (The current version only supports Little-Endian). **Values:** ■ Little-Endian (0) ■ Big-Endian (1) **Default Value:** Little-Endian **Enabled:** 0 **Parameter Name:** X2H_AXI_ENDIANNESS

Table 3-1   Parameters (Continued)

Label	Description
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:** X2H_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 transaction. **Note:** If csysreq de-asserts while waiting for X2H_LOWPWR_NOPX_CNT number of cycles, cactive de-asserts immediately. If a new transaction is initiated during the wait period, the counting will be halted, cactive does not de-assert, and the counting is reinitiated when there is no pending transactions. This parameter is available only if X2H_LOWPWR_HS_IF is true. **Values:** 0, ..., 4294967295 **Default Value:** 0 **Enabled:** X2H_LOWPWR_HS_IF==1 **Parameter Name:** X2H_LOWPWR_NOPX_CNT
Maximum Number of Outstanding Read Transactions	Maximum number of outstanding AXI read transactions; higher values allow more transactions to be active simultaneously, but also increase gate count slightly. This parameter is available only if X2H_LOWPWR_HS_IF is true. **Values:** 1, ..., 32 **Default Value:** 4 **Enabled:** (X2H_LOWPWR_HS_IF == 1) ? (X2H_LOWPWR_LEGACY_IF == 0) : 0 **Parameter Name:** X2H_MAX_PENDTRANS_READ
Maximum Number of Outstanding Write Transactions	Maximum number of AXI write transactions that may be outstanding at any time; higher values allow more transactions to be active simultaneously, but also increase gate count slightly. This parameter is available only if X2H_LOWPWR_HS_IF is true. **Values:** 1, ..., 32 **Default Value:** 4 **Enabled:** (X2H_LOWPWR_HS_IF == 1) ? (X2H_LOWPWR_LEGACY_IF == 0) : 0 **Parameter Name:** X2H_MAX_PENDTRANS_WRITE

Table 3-1   Parameters (Continued)

Label	Description
Clocking Mode	The bridge AXI slave interface is clocked by aclk. The bridge AHB master interface is clocked by mhclk. This parameter specifies the relationship between aclk and mhclk, and also determines what the bridge does to sychronize between the two domains. ■ 0: aclk and mhclk are different, and completely asynchronous ■ 1: aclk and mhclk are different, one is the multiple of the other ■ 2: Both aclk and mhclk are driven by the same clock signal For more information on clocking modes, see the "Clock Adaptation" section in the DW_axi_x2h Databook. **Values:** ■ Two Asynchronous Clocks (0) ■ Two Synchronous Clocks (1) ■ Single Clock (2) **Default Value:** Two Asynchronous Clocks **Enabled:** Always **Parameter Name:** X2H_CLK_MODE
Dual Clock Mode Synchronisation Depth	Number of synchronization register stages in the internal channel FIFOs for signals crossing clock domains between AXI and AHB. When aclk and mhclk are quasi-synchronous this parameter can be set to 0 to reduce the latency across the bridge. ■ 0: No synchronization satges ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages **Values:** 0, 2, 3, 4 **Default Value:** 2 **Enabled:** X2H_CLK_MODE != 2 **Parameter Name:** X2H_DUAL_CLK_SYNC_DEPTH
Common Command Queue Depth	Number of locations in the Common Command (CMD) queue, which transfers AXI commands from the bridge AXI slave to the bridge AHB master. For more information on the Write Data buffer, see the "AXI to AHB Common Command (CMD) Queue" and "Configuring Buffer Depths" sections in the DW_axi_x2h Databook. Set to 1 if the clocking mode is set to Single clock. **Values:** 1, 2, 4, 8, 16, 32 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2H_CMD_QUEUE_DEPTH

Table 3-1   Parameters (Continued)

Label	Description
Write Data Buffer Depth	Number of locations in the write data buffer, which transfers write data from the bridge AXI slave to the bridge AHB master. For more information on the write data buffer, see the "Write Transfers from AXI to AHB" and "Configuring Buffer Depths" sections in the DW_axi_x2h Databook. Set to 1 if the clocking mode is set to Single clock. **Values:** 1, 2, 4, 8, 16, 32, 64 **Default Value:** 16 **Enabled:** Always **Parameter Name:** X2H_WRITE_BUFFER_DEPTH
Write Response Buffer Depth	Number of locations in the write response buffer which contain responses from the AHB Master indicating the AHB completion of the AXI write transfer. For more information on the write response buffer, see the "Write Transfers from AXI to AHB" and "Configuring Buffer Depths" sections in the DW_axi_x2h Databook. Set to 1 if the clocking mode is set to Single clock. **Values:** 1, 2, 4, 8, 16 **Default Value:** 2 **Enabled:** Always **Parameter Name:** X2H_WRITE_RESP_BUFFER_DEPTH
Read Data Buffer Depth	Number of locations in the read data buffer which transfers read data from the bridge AHB master to the bridge AXI slave. For more information on the read data buffer, see the "Read Transfers from AXI to AHB" and "Configuring Buffer Depths" sections in the DW_axi_x2h Databook. **Values:** 1, 2, 4, 8, 16, 32 **Default Value:** 8 **Enabled:** Always **Parameter Name:** X2H_READ_BUFFER_DEPTH
Address Bus Width	Address bus width of the AHB system to which the bridge is attached as an AHB master. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2H_AHB_ADDR_WIDTH

Table 3-1   Parameters (Continued)

Label	Description
Data Bus Width	Read and write data bus width of the AHB system to which the bridge is attached as an AHB master. For more information on data bus widths, see the "Data Width Adaptation" section in the DW_axi_x2h Databook. **NOTE**: Data bus width for the AHB master interface must be less than or equal to that of the AXI slave interface. **Values:** 32, 64, 128, 256 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2H_AHB_DATA_WIDTH
AHB-Lite	Configure the bridge for AHB or AHB-Lite operation. ■ 0: AHB ■ 1: AHB-Lite For more information, see the "AHB Lite" section in the DW_axi_x2h Databook. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2H_AHB_LITE
Pass AXI Lock to AHB	Configures the bridge to lock the AHB when performing AXI locked access transfers; that is, to pass the LOCK signal from the AXI to the AHB. For more information, see the "Locked AXI-to-AHB Transactions" section in the DW_axi_x2h Databook. **Values:** ■ Do not Pass AXI Lock to AHB (0) ■ Pass AXI Lock to AHB (1) **Default Value:** Do not Pass AXI Lock to AHB **Enabled:** X2H_AXI_INTERFACE_TYPE==0 **Parameter Name:** X2H_PASS_LOCK
Prohibit AHB INCR bursts	As an AHB Master, allows bridge to or prohibit its use of the undefined-length INCR burst (defined-length bursts INCR4, INCR8, INCR16 are allowed in either case). For information on reads and writes of undefined length, see "Read Command Usage" and "Write Command Usage" sections in the DW_axi_x2h Databook. **Values:** ■ Allow INCR (0) ■ Prohibit INCR (1) **Default Value:** Allow INCR **Enabled:** Always **Parameter Name:** X2H_USE_DEFINED_ONLY

Table 3-1   Parameters (Continued)

Label	Description
AHB Side Pipelining Modes	Puts in or leaves out pipeline stages, which ease critical timing paths in the AHB clock domain. ■ 0: No pipelining - Omits pipelining and decreases area but increases difficulty of meeting synthesis timing constraints. ■ 1: Pipeline Interface AHB Paths - Adds pipelining to ease timing constraints between the AHB bus signals and internal FIFOs. ■ 2: Pipeline Internal AHB Paths - Adds pipelining to ease timing constraints between internal register-to-register paths. ■ 3: Pipeline both - Adds pipelining for both interface and internal AHB constraints. **Values:** ■ No Pipelining (0) ■ Pipeline Interface AHB Paths (1) ■ Pipeline Internal AHB Paths (2) ■ Pipeline Both (3) **Default Value:** Pipeline Both **Enabled:** Always **Parameter Name:** X2H_AHB_BUFFER_POP_MODE
Data Bus Endianness	Data bus endian mode of the AHB system to which the bridge is attached as an AHB master. (The current version only supports Little-Endian) **Values:** ■ Little-Endian (0) ■ Big-Endian (1) **Default Value:** Little-Endian **Enabled:** 0 **Parameter Name:** X2H_AHB_ENDIANNESS
Sideband/User Bus Width	This parameter specifies the width of awsideband/awuser, arsideband/aruser and mid signals. When set to 0, these ports are removed from the interface. **Values:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 0 **Enabled:** Always **Parameter Name:** X2H_MID_WIDTH

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:

4.1 Clock and Resets Signals
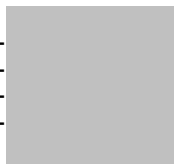
Table 4-1   Clock and Resets Signals

Port Name	I/O	Description
aclk	I	AXI slave clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aresetn	I	AXI slave reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of aclk. DW_axi_x2h does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
mhclk	I	AHB master clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhresetn	I	AHB master reset. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.2   AXI Write Address Channel Signals

Table 4-2   AXI Write Address Channel Signals

Port Name	I/O	Description
awready	O	AXI slave 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 **Registered:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==0)) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr[(X2H_AXI_ADDR_WIDTH-1):0]	I	AXI 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:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2   AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awvalid	I	AXI slave 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:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awburst[1:0]	I	AXI slave write burst. Combined with size information, shows how address for each transfer within burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock[(X2H_AXI_LTW-1):0]	I	AXI slave write lock. Provides additional information about characteristics of transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize[2:0]	I	AXI slave write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awprot[2:0]	I	AXI write protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. Bit 1 is not actually required to be connected; it is not used and therefore is not connected internal to the component. Bits 2 and 0 are required. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2   AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awid[(X2H_AXI_ID_WIDTH-1):0]	I	AXI slave write address identification. Gives identification tag for write address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsideband[(X2H_MID_WIDTH-1):0]	I	Sideband bus for AXI write address channel. **Exists:** ((X2H_MID_WIDTH!=0) && (X2H_AXI_INTERFACE_TYPE==0)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awuser[(X2H_MID_WIDTH-1):0]	I	User bus for AXI write address channel. **Exists:** ((X2H_MID_WIDTH!=0) && (X2H_AXI_INTERFACE_TYPE==1)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlen[(X2H_AXI_BLW-1):0]	I	AXI slave write burst length. Specifies exact number of transfers in burst; determines number of data transfers associated with address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache[3:0]	I	AXI write cache. Indicates bufferable, cacheable, write-through, and write-back attributes of transaction. Bits 3 and 2 are not actually required to be connected; they are not used and therefore are not connected internal to the component. Bits 1 and 0 are required. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 AXI Write Data Channel Signals

wdata - [   ] - wready
wlast - [   ]
wvalid - [   ]
wid -   [   ]
wstrb - [   ]

Table 4-3   AXI Write Data Channel Signals

Port Name	I/O	Description
wready	O	AXI slave write ready. Indicates that slave can accept write data. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** (X2H_LOWPWR_HS_IF == 0)? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdata[(X2H_AXI_DATA_WIDTH-1):0]	I	AXI slave write data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wlast	I	AXI slave write last. Indicates last transfer in write burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wvalid	I	AXI slave write valid. Indicates 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 **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3   AXI Write Data Channel Signals (Continued)

Port Name	I/O	Description
wid[(X2H_AXI_ID_WIDTH-1):0]	I	AXI slave write identification tag of write data transfer.Value must match awid value of write transaction. **Exists:** X2H_AXI_INTERFACE_TYPE==0 **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wstrb[((X2H_AXI_DATA_WIDTH/8)-1):0]	I	AXI slave write strobes. Indicates which byte lanes to update in memory. One write strobe for each eight bits of write data bus. WSTRB[n] is associated with the following byte of WDATA: WDATA[8n+7 : 8n]. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4   AXI Write Response Channel Signals

bready -   - bresp
                - bid
                - bvalid

Table 4-4   AXI Write Response Channel Signals

Port Name	I/O	Description
bresp[1:0]	O	AXI write response. Indicates status of write transaction. Allowable responses are OKAY and SLVERR. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bid[(X2H_AXI_ID_WIDTH-1):0]	O	AXI slave write response identification tag. Value must match awid value of write transaction to which slave responds. Width: X2H_AXI_ID_WIDTH configuration parameter **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bvalid	O	AXI slave write response valid. Indicates that valid write response is available. ■ 0: Write response not available ■ 1: Write response available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4   AXI Write Response Channel Signals (Continued)

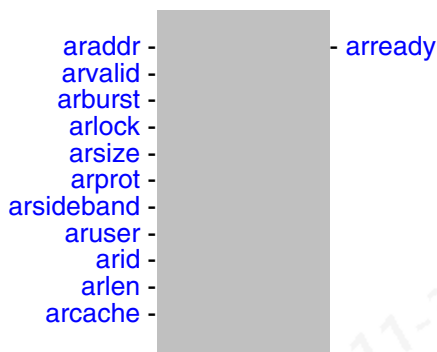| Port Name	I/O	Description
bready	I	AXI slave write response ready. Indicates master can accept response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5   AXI Read Address Channel Signals

araddr -
arvalid -
arburst -
arlock -
arsize -
arprot -
arsideband -
aruser -
arid -
arlen -
arcache -
- arready

Table 4-5   AXI Read Address Channel Signals

Port Name	I/O	Description
arready	O	AXI slave read 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 **Registered:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==0)) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
araddr[(X2H_AXI_ADDR_WIDTH-1):0]	I	AXI slave read address. Gives 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:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5   AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arvalid	I	AXI slave 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:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
arburst[1:0]	I	AXI slave read burst. Combined with size information, shows how address for each transfer within burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock[(X2H_AXI_LTW-1):0]	I	AXI slave read lock. Provides additional information about characteristics of transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize[2:0]	I	AXI slave read burst size. Indicates size of each transfer in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arprot[2:0]	I	AXI read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. Bit 1 is not actually required to be connected; it is not used and therefore is not connected internal to the component. Bits 2 and 0 are required. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5   AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arsideband[(X2H_MID_WIDTH-1):0]	I	Sideband bus for AXI read address channel. **Exists:** ((X2H_MID_WIDTH!=0) && (X2H_AXI_INTERFACE_TYPE==0)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aruser[(X2H_MID_WIDTH-1):0]	I	User bus for AXI read address channel. **Exists:** ((X2H_MID_WIDTH!=0) && (X2H_AXI_INTERFACE_TYPE==1)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arid[(X2H_AXI_ID_WIDTH-1):0]	I	AXI read address identification tag for read address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlen[(X2H_AXI_BLW-1):0]	I	AXI slave read burst length. Gives exact number of transfers in burst; determines number of data transfers associated with address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache[3:0]	I	AXI read cache. Indicates bufferable, cacheable, read-through, and read-back attributes of transaction. Bits 3 and 2 are not actually required to be connected; they are not used and therefore are not connected internal to the component. Bits 1 and 0 are required. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.6   AXI Read Data Channel Signals

rready -
- rdata
- rresp
- rid
- rlast
- rvalid

Table 4-6   AXI Read Data Channel Signals

Port Name	I/O	Description
rdata[(X2H_AXI_DATA_WIDTH-1):0]	O	AXI slave read data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp[1:0]	O	AXI slave read response. Indicates status of write transaction; allowable responses are OKAY and SLVERR. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rid[(X2H_AXI_ID_WIDTH-1):0]	O	Read identification tag. Value is generated by slave and must match arid value of read transaction to which it responds. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rlast	O	AXI slave read last. Indicates last transfer in read burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6   AXI Read Data Channel Signals (Continued)

Port Name	I/O	Description
rvalid	O	AXI slave read data valid. Indicates that required read data is available and read transfer can complete. ■ 0: Read data not available ■ 1: Read data available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rready	I	AXI slave read data ready. Indicates master can accept read data and response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.7 AHB Master Inputs Signals

mhgrant -
mhrdata -
mhready -
mhresp -

Table 4-7   AHB Master Inputs Signals

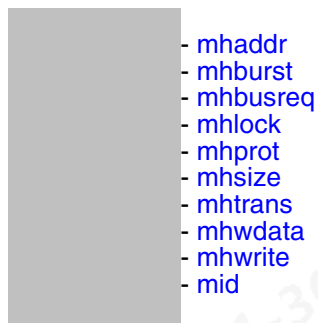| Port Name	I/O	Description
mhgrant	I	AHB bus grant. Asserted by arbiter to indicate that requesting master has won ownership of bus. **Exists:** X2H_AHB_LITE==0 **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhrdata[(X2H_AHB_DATA_WIDTH-1):0]	I	Transfer read data. Used to transfer data from bus slaves to bus master during read operations. **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhready	I	Ready response from selected slave. Indicates current transfer is complete. **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhresp[1:0]	I	Transfer response. Indicates response type from slave. **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.8 AHB Master Outputs Signals

- mhaddr
- mhburst
- mhbusreq
- mhlock
- mhprot
- mhsize
- mhtrans
- mhwdata
- mhwrite
- mid

Table 4-8   AHB Master Outputs Signals

Port Name	I/O	Description
mhaddr[(X2H_AHB_ADDR_WIDTH-1):0]	O	AHB address bus. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhburst[2:0]	O	Transfer type. Indicates if transfer constitutes part of a burst. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhbusreq	O	Bus request signal. Asserted by master to request access to bus. In AHB-Lite mode, this output is a constant, driven high. **Exists:** X2H_AHB_LITE==0 **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-8   AHB Master Outputs Signals (Continued)

Port Name	I/O	Description
mhlock	O	Bus lock. Asserted by bus master to indicate need for locked transaction. **Exists:** Always **Synchronous To:** ((X2H_AXI_INTERFACE_TYPE == 0) && (X2H_PASS_LOCK == 1)) ? "mhclk" : "None" **Registered:** ((X2H_AXI_INTERFACE_TYPE == 0) && (X2H_PASS_LOCK == 1)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhprot[3:0]	O	Protection control. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhsize[2:0]	O	Transfer size. Indicates size of transfer. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhtrans[1:0]	O	Transfer control. Indicates type of transfer being performed. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwdata[(X2H_AHB_DATA_WIDTH-1):0]	O	Transfer write data. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwrite	O	Transfer write control. When HIGH, indicates write transfer. When LOW, indicates read transfer. **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-8   AHB Master Outputs Signals (Continued)

Port Name	I/O	Description
mid[(X2H_MID_WIDTH-1):0]	O	Non-standard sideband signal output. **Exists:** X2H_MID_WIDTH!=0 **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.9 AXI Low Power Interface Signals

csysreq -   - csysack
         - cactive

Table 4-9　AXI Low Power Interface Signals

Port Name	I/O	Description
csysack	O	Low-power request acknowledgment. ■ De-asserted by DW_axi_x2h to acknowledge request to enter low-power state ■ Asserted by DW_axi_x2h to acknowledge request to exit low-power state **Exists:** X2H_LOWPWR_HS_IF==1 **Synchronous To:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==1) && (X2H_AXI_LOW_POWER==0)) ? "None" : "aclk" **Registered:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==1) && (X2H_AXI_LOW_POWER==0)) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
cactive	O	Clock active request. De-asserted by DW_axi_x2h to tell system low-power controller (LPC) that clock can be removed. ■ 1: Peripheral clock required ■ 0: Peripheral clock not required **Note**: Clock should be removed positive edge after cactive and csysack signals are sampled low (0). **Exists:** X2H_LOWPWR_HS_IF==1 **Synchronous To:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==1) && (X2H_AXI_LOW_POWER==0)) ? "None" : "aclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-9   AXI Low Power Interface Signals (Continued)

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:** X2H_LOWPWR_HS_IF==1 **Synchronous To:** ((X2H_LOWPWR_HS_IF==1) && (X2H_LOWPWR_LEGACY_IF==1) && (X2H_AXI_LOW_POWER==0)) ? "None" : "aclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

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
X2H_AXI_LOW_POWER	{X2H_LOWPWR_LEGACY_IF == 1 ? 1 : 0}
X2H_AXI_LTW	{ [::DW_axi_x2h::calc_lock_width X2H_AXI_INTERFACE_TYPE] }
X2H_LOWPWR_LEGACY_IF	0

Synopsys, Inc.

6

Verification

This chapter provides an overview of the testbench available for the DW_axi_x2h verification. After the DW_axi_x2h has been configured and the verification is setup, simulations can be run automatically. For information on running simulations for DW_axi_x2h in coreAssembler or coreConsultant, see the "Running the Simulation" section in the user guide.

> 👉 **Note**   The DW_axi_x2h 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 71
- "Testbench Directories and Files" on page 72
- "Packaged Testcases" on page 73

6.1   Verification Environment

DW_axi_x2h 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_x2h testbench:

Figure 6-1   DW_axi_x2h UVM Verification Environment
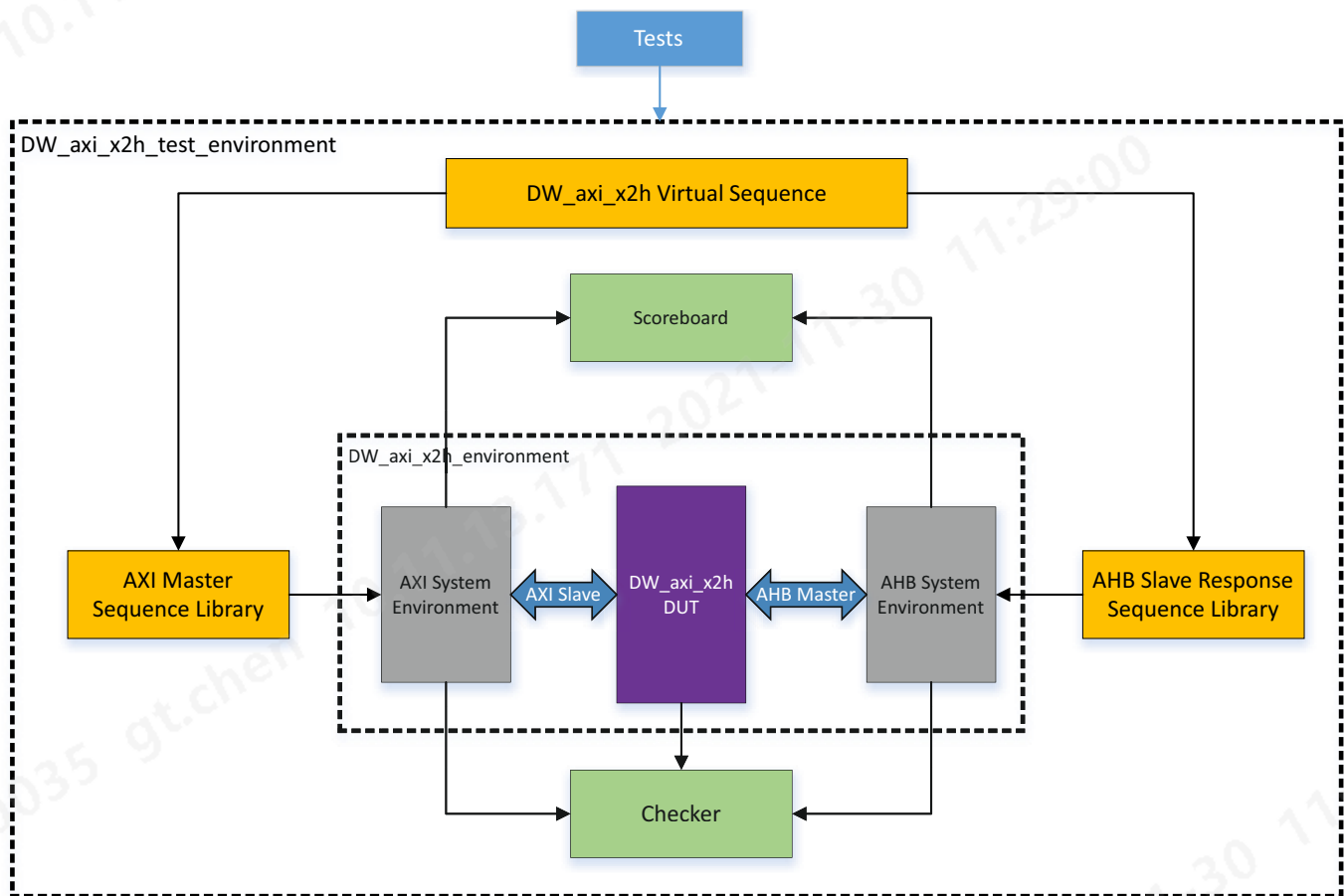
The testbench consists of the following elements:

- 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
        - AHB VIP model is used to generate response for the incoming AHB transfers

6.2   Testbench Directories and Files

The DW_axi_x2h verification environment contains the following directories and associated files.

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

Table 6-1   DW_axi_x2h 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, score-board, 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 description is explained in Table 6-2:

Table 6-2   DW_axi_x2h Test Description

Test Name	Test Description
test_x2h_random	This test is aimed at generating random traffic which is AXI and AHB compliant. The traffic is generated randomly based on DW_axi_x2h 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 DW_axi_x2h. 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 as per the DW_axi_x2h requirement. The Primary port is monitored for the correctness of the AXI protocol across different IP configuration. AHB Slave Sequence from the VIP generates various possible AHB responses for the requests from secondary port of the DW_axi_x2h. The response, delay, and other attributes are generated randomly. The secondary port is monitored for the correctness of the AHB protocol across different IP configuration.

7

Integration Considerations

There are many trade-offs in integrating any configurable design. The following sections help define some of these trade-offs. This should be used to help guide the IP integrator on configuring the DW_axi_x2h for successful integration into the larger subsystem.

7.1   Configuring Buffer Depths

The following sections discuss how to configure buffer depths for the CMD queue, the Write Data buffer, the Write Response buffer, and the Read Data buffer.

7.1.1   Common Command (CMD) Queue

The default depth of the CMD queue is 4; both AXI write commands and read commands go through this queue. Commands are removed from the queue as the DW_axi_x2h completes them on the AHB.

If the CMD queue fills up, the DW_axi_x2h holds both awready and arready de-asserted to temporarily refuse more AXI commands. If this occurs, you should be aware that the following might prevent AXI writes and reads from being delivered to AXI slaves:

- Architecture of AXI bus interconnect; single-address versus multiple-address

- Number of AXI masters and slaves

- Traffic patterns between various AXI masters accessing various AXI slaves

If you find that this reduction in functional bandwidth is a problem that occurs in your application, you can make the CMD queue deeper in order to reduce the probability of the CMD queue filling up. However, keep in mind that this can make it more difficult to meet register-to-register timing inside the DW_axi_x2h.

If you are concerned about the CMD queue filling up, keep in mind that the chances of this happening are influenced by the AHB subsystem. The AXI writes and reads tend to sit in the CMD queue longer, which raises the probability of the CMD queue filling up, based on one or more of the following:

- If the AHB runs on a slower clock frequency

- If the AHB data width is narrower, which requires that transfers must be downsized

- If there are a lot of other busy AHB masters, which forces the DW_axi_x2h to wait for the bus

- If the AHB slaves insert many wait states and/or send many SPLIT or RETRY responses

7.1.2 Write Data Buffer

The default depth of the Write Data buffer is 16.

If the Write Data buffer fills up, the DW_axi_x2h holds wready de-asserted in order to temporarily refuse more AXI write data until some of the data is written to the AHB.

Considerations similar to the CMD queue also apply to the Write Data buffer, except that the AXI bus interconnect may have a different architecture for data than it has for the command flow; that is, the interconnect may be a single-address versus multiple-data architecture. Again, verify if the Write Data buffer fills up, if the data flow will be stalled to other AXI slaves.

Just as with the CMD queue, if you increase the depth of the Write Data buffer, it may be more difficult to meet register-to-register timing inside the DW_axi_x2h.

7.1.3 Write Response Buffer

The default depth of the Write Response buffer is 2.

The AHB Master controller side of the DW_axi_x2h needs to push a response into the Write Response buffer after completing each AXI write command. If the Write Response buffer is full at this time, the AHB Master controller stalls until space is available (no more writes or reads are initiated on the AHB bus by the AHB Master controller until it is able to push in the response).

Be aware that, if the Write Response buffer fills up and causes the AHB Master controller to stall, this may contribute to the CMD queue and/or Write Data buffers filling up too.

The Write Response buffer could fill up if the AXI bus is slow to accept write responses. This might happen if the AXI interconnect has to hold off write responses from the DW_axi_x2h due to ordering requirements.

In contrast to the CMD queue and Write Data buffer, the Write Response buffer is less likely to fill up if the AHB subsystem is slow; that is, a slow AHB clock, a narrow AHB data bus requiring downsizing, slow AHB slaves, and so on.

If you need to increase the depth of the Write Response buffer, it should not have much effect on register-to-register timing inside the DW_axi_x2h; it makes the timing slightly worse for the bid and bresp outputs.

7.1.4   Read Data Buffer

The default depth of the Read Data buffer is 8.

The effects of the Read Data buffer filling are more complex than the CMD queue or the Write Response buffer because, when the DW_axi_x2h performs a defined-length burst read on the AHB, it does not start the burst until it has enough space in the Read Data buffer to complete it. So, for example, if the DW_axi_x2h determines that it must perform an INCR8 read, it does not start until it has space in the Read Data buffer for all eight beats of AHB data; this requires eight buffer spaces if the transfer is not downsized. In this case, the transfer stall due to the Read Data buffer appearing to be too full, even if there are seven spaces available.

☞ **Note**   If a read stalls because the Read Data buffer is deemed too full, this can contribute to the CMD queue and/or Write Data buffers also filling up.

The Read Data buffer can fill up if the AXI bus is too slow in accepting read data, which can happen for example if read data from the DW_axi_x2h is held off due to ordering requirements. Like the Write Response buffer, the Read Data buffer is more likely to fill up if the AHB side is relatively fast; that is, it has a fast AHB clock, it has a wide AHB data bus, there are fast AHB slaves, and so on.

Another thing you might consider for the Read Data buffer depth is that, due to the fact that a defined-length burst read cannot be started until all the necessary space is available, the depth of the Read Data buffer limits what kind of defined-length burst reads the DW_axi_x2h can issue on the AHB. For example, if the Read Data buffer only has enough space for 8 beats of AHB data and there is no downsizing, the DW_axi_x2h can never issue an INCR16 burst read.

Be aware that ¡°enough space for eight beats of AHB data¡± does not necessarily mean the same as eight spaces in the Read Data buffer. If the AXI-to-AHB read is downsized at 2:1¡ªsuch as a 64-bit AXI read to a 32-bit AHB¡ªthen the eight beats of AHB data can consume only four spaces in the Read Data buffer. If the AXI-to-AHB read is downsized at 4:1¡ªsuch as a 128-bit AXI read to a 32-bit AHB¡ªthen the eight beats of AHB data consume only two spaces in the Read Data buffer. If there is no downsizing¡ªthat is, the size of the AXI transfer is no greater than the AHB data width¡ªthen the eight beats of AHB data consume eight spaces in the Read Data buffer.

Keep in mind that, if the AXI data width is wider than the AHB data width, which means that many AXI-to-AHB transfers are downsized, this makes an ¡°effective Read Data buffer depth¡± of 2x, 4x, or even 8x the depth that you configure. Taking that into account, the following illustrate what happens for various "effective" depths:

- Effective Read Data buffer depth = 2

  No defined-length burst reads are possible; must use undefined-length INCR. An example configuration that would result in this ¡°effective¡± depth would be Read Data buffer depth=1, AXI data width=64, AHB data width=32. This results in a 2x increase in the ¡°effective¡± depth of the Read Data buffer, since two AHB data beats fill only one Read Data buffer space.

- Effective Read Data buffer depth = 4

  INCR4 reads are performed. The DW_axi_x2h must wait for the Read Data buffer to empty completely before starting an INCR4 read.

Note If a read stalls because the Read Data buffer is deemed too full, this can contribute to the CMD queue and/or Write Data buffers also filling up.

- Effective Read Data buffer depth = 8

  INCR4 reads are performed. The DW_axi_x2h can start a new INCR4 read when the Read Data buffer is at least half empty. The DW_axi_x2h is architected such that INCR8 reads are not done in this case because the Read Data buffer would have to empty completely between each one; it is better to use INCR4s in order to let them "stream". An example configuration that would result in this "effective" depth would be Read Data buffer depth=8, AXI data width=32, AHB data width=32. Another would be Read Data buffer depth=2, AXI data width=128, AHB data width=32.

- Effective Read Data buffer depth = 16

  INCR8 and INCR4 reads are performed. When doing INCR8 reads, the DW_axi_x2h can start one when the Read Data buffer is at least half empty. The DW_axi_x2h is architected such that INCR16 reads are not performed in this case because the Read Data buffer would have to empty completely between each one; it is better to use INCR8s in order to let them "stream".

- Effective Read Data buffer depth = 32

  INCR16, INCR8, and INCR4 reads are performed. When doing INCR16 reads, the DW_axi_x2h can start one when the Read Data buffer is at least half empty.

To summarize, these are the primary considerations for the Read Data buffer:

1. Make the Read Data buffer deep enough to allow the DW_axi_x2h to use reasonably-sized defined-length burst read commands on the AHB.

2. Keep in mind that a smaller Read Data buffer depth is needed if the AHB data width is less than the AXI data width.

3. If interruptions occur in the flow of read data on the AXI bus, you might increase the depth of the Read Data buffer in order to prevent the DW_axi_x2h from stalling.

If you increase the depth of the Read Data buffer, it should not have much effect on register-to-register timing inside the DW_axi_x2h. However, it makes timing slightly worse for the rdata, rid, and rlast outputs.

7.2 Clocking Configuration

You can configure the DW_axi_x2h in order to support different relationships between the clock on the AXI system (aclk) and the clock on the AHB system (mhclk). You do this through the "Clocking mode" pulldown in the Specify Configuration activity. You have three choices:

- Two Asynchronous Clocks
- Single Clock
- Two Synchronous Clocks

The following discuss some considerations in choosing one or another.

7.2.1 Two Asynchronous Clocks

When configured with two asynchronous clocks, the DW_axi_x2h allows any relationship¡ªincluding completely asynchronous – between aclk and mhclk. They can be fully asynchronous, and either can be

faster than the other. In this configuration, the CMD queue, Write Data buffer, Write Response buffer, and Read Data buffer are configured to synchronize the information that passes through them from one clock domain to the other.

This is the default setup. It is always safe to configure the DW_axi_x2h this way.

7.2.2   Single Clock

You can select this mode if the AXI interface and the AHB interface are to run at the same frequency. When possible, the advantage of this mode is that the CMD queue, Write Data buffer, Write Response buffer, and Read Data buffer are configured to pass information straight through with no synchronization. This saves three clocks of latency on each buffer crossing, as well as some design area.

Another possible benefit is that, when the four buffers are not used for synchronization, they can be configured to be just one entry deep. In the other clocking modes, where the buffers are used for synchronization, they must all be configured to be at least two entries deep, due to the synchronizer circuit.

⚠ **Attention**	When the DW_axi_x2h is configured for single clock operation, the aclk and mhclk inputs both drive registers. The number of registers driven by each of these inputs varies according to other configuration selections, particularly the buffer depths. It becomes a clock tree balancing issue to ensure that all these registers receive the rising clock edge at the same time with minimal skew. A common source should drive both the aclk and mhclk inputs. You must also ensure that the clock tree generation works down into the DW_axi_x2h and assures that no clock skew occurs between any of the registers, even between one driven by the aclk signal and another driven by the mhclk signal.

Only the rising clock edge must be de-skewed; the falling edge is not critical. No register in DW_axi_x2h uses the falling edge.

7.2.3   Two Synchronous Clocks

This selection should be used in only special circumstances.

As with the default Two Asynchronous Clocks setting, the buffers are configured to provide some synchronization. However, where double-level synchronization is used in the default mode, only single-level synchronization is used in this mode. This single-level synchronization is actually two-stage synchronization, where the first synchronizing register is on the negative destination clock edge and the second is on the positive destination clock edge. It is referred to as single-stage, since the synchronization can be performed in a single clock cycle.

Compared to the default Two Asynchronous Clocks setting, using this mode saves a little bit of area and one clock of latency on each buffer crossing.

This mode should be used only when aclk and mhclk are different, where you cannot use Single Clock mode, but they have enough of a relationship that makes you confident that the double-level synchronization of Two Asynchronous Clocks mode is unnecessary.

A plausible scenario for using this mode is where one clock is a multiple of the other. However, to avoid metastability problems in the single-level synchronizers, you must know something about the relationship between the rising edges of aclk and mhclk at the registers, after the clock tree is inserted. At the clock domain crossings, the first synchronizer register loads directly from the output of a register in the other

clock domain. You therefore need to know that the rising edges of aclk and mhclk are perfectly aligned, or that they are always well separated.

Because of the risk versus the benefit gained, it is recommended that you use the Two Asynchronous Clocks mode instead of this setting, even if the clocks are not completely asynchronous.

7.3   Configuring the DW_axi_x2h for High Clock Frequencies

Two settings modify the DW_axi_x2h to support higher frequencies for the AHB clock. Both settings are controlled through the "AHB side pipelining modes" pulldown in the Specify Configuration screen; by default, both settings are turned on.

- Pipeline Interface AHB Paths ¨C Configures the circuit with extra pipelining to allow late arrival of the mhgrant, mhready, and mhresp signals from the AHB. This costs some gates¡ªtypically 1000 NAND-2 equivalents when working with 32-bit AXI and AHB address and data buses¡ªbut does not increase the latency for AXI commands to reach the AHB.

Pipeline Interface AHB Paths is turned on by default in order to support late-arriving AHB inputs. You might be able to turn this off, and still have the synthesis meet timing, if some combination of the following are true:

- AHB clock speed is low. For example, if you set up the AHB clock for a 6ns period (166 MHz), the 67% arrival time of mhgrant, mhready, and mhresp still leaves 2ns setup time to the next AHB clock. This should be enough with a 0.13 micron library.

- The mhgrant, mhready, and mhresp take up less than 67% of the clock period in your application, and you provide the smaller, more accurate percentages for synthesis constraints using the Specify Port Constraints activity.

- Your silicon library has very fast gates.

- Pipeline Internal AHB Paths ¨C Configures the circuit with extra pipelining to improve AHB domain register-to-register timing inside the DW_axi_x2h. This costs some area (typically 1600 NAND-2 equivalents, when working with 32-bit AXI and AHB address and data busses), and also adds one additional AHB clock of latency for AXI commands to reach the AHB.

Pipeline Internal AHB Paths is turned on by default, since it is needed to support high AHB clock speeds. With this switch turned on, the longest path between AHB clocked registers typically has around 27 levels of logic. With this switch turned off, the longest path between AHB clocked registers goes up to typically around 40 levels of logic.

If your combination of silicon library and desired AHB clock speed allows 40 levels of logic between registers, you can try turning this off. For example, this should be OK up to at least 200MHz with a 0.13 micron library. Then, see if the synthesis can still meet timing. If so, you can save the gates and the extra AHB clock of latency.

7.4   Accessing Top-level Constraints

To get SDC constraints out of coreConsultant, you need to first complete the synthesis activity and then use the "write_sdc" command to write out the results:

1.   This cC command sets synthesis to write out scripts only, without running DC:

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

```
autocomplete_activity Synthesize
```

3. Finally, this cC command writes out SDC constraints:

```
write_sdc <filename>
```

7.5   Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_axi_x2h.

7.5.1   Power Consumption, Frequency, Area and DFT Coverage

Table 7-1 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_axi_x2h using the industry standard 16nm technology library.

Table 7-1   Synthesis Results for DW_axi_x2h

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtC ov (%)
			Dynamic	Static	StuckAt Test	Transition	
Default Configuration	aclk = 400MHz mhclk = 300MHz	20891	0.767 mW	20 nW	99.9	99.82	100

Table 7-1   Synthesis Results for DW_axi_x2h

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtC ov (%)
			Dynamic	Static	StuckAt Test	Transition	
Typical Configuration 1 X2H_AHB_ADDR_WIDTH = 64 X2H_AHB_BUFFER_POP_MODE = 1 X2H_AHB_DATA_WIDTH = 256 X2H_AHB_LITE = 0 X2H_AXI_ADDR_WIDTH = 64 X2H_AXI_BLW = 8 X2H_AXI_DATA_WIDTH = 256 X2H_AXI_ID_WIDTH = 16 X2H_AXI_INTERFACE_TYPE = 0 X2H_AXI_LOW_POWER = 1 X2H_MAX_PENDTRANS_READ = 16 X2H_MAX_PENDTRANS_WRITE = 8 X2H_CLK_MODE = 0 X2H_CMD_QUEUE_DEPTH = 32 X2H_READ_BUFFER_DEPTH = 32 X2H_USE_DEFINED_ONLY = 1 X2H_WRITE_BUFFER_DEPTH = 64 X2H_WRITE_RESP_BUFFER_DE PTH = 16	aclk = 400MHz mhclk = 300MHz	344812	12.3 mW	300 nW	99.99	99.98	100

Table 7-1   Synthesis Results for DW_axi_x2h

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtCov (%)
			Dynamic	Static	StuckAt Test	Transition	
Typical Configuration 2 X2H_AHB_ADDR_WIDTH = 64 X2H_AHB_BUFFER_POP_MODE = 3 X2H_AHB_DATA_WIDTH = 256 X2H_AHB_LITE = 0 X2H_AXI_ADDR_WIDTH = 64 X2H_AXI_BLW = 8 X2H_AXI_DATA_WIDTH = 256 X2H_AXI_ID_WIDTH = 16 X2H_AXI_INTERFACE_TYPE = 1 X2H_AXI_LOW_POWER = 1 X2H_LOWPWR_NOPX_CNT = 105 X2H_CLK_MODE = 0 X2H_CMD_QUEUE_DEPTH = 32 X2H_READ_BUFFER_DEPTH = 32 X2H_USE_DEFINED_ONLY = 1 X2H_WRITE_BUFFER_DEPTH = 64 X2H_WRITE_RESP_BUFFER_DEPTH = 16	aclk = 400MHz mhclk = 300MHz	349995	12.5 mW	300 nW	99.99	99.98	100

Synopsys, Inc.   SolvNetPlus
DesignWare   **83**

Table 7-1   Synthesis Results for DW_axi_x2h

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtCov (%)
			Dynamic	Static	StuckAt Test	Transition	
Typical Configuration 3 X2H_AHB_ADDR_WIDTH = 64 X2H_AHB_BUFFER_POP_MODE = 2 X2H_AHB_DATA_WIDTH = 64 X2H_AHB_LITE = 0 X2H_AXI_ADDR_WIDTH = 64 X2H_AXI_DATA_WIDTH = 128 X2H_AXI_ID_WIDTH = 16 X2H_AXI_INTERFACE_TYPE = 1 X2H_CLK_MODE = 2 X2H_CMD_QUEUE_DEPTH = 32 X2H_LOWPWR_HS_IF = 1 X2H_LOWPWR_LEGACY_IF = 1 X2H_READ_BUFFER_DEPTH = 32 X2H_USE_DEFINED_ONLY = 0 X2H_WRITE_BUFFER_DEPTH = 64 X2H_WRITE_RESP_BUFFER_DEPTH = 16	aclk = 300MHz mhclk = 300MHz	194591	5.7 mW	200 nW	99.99	99.92	100

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_x2h.

- "BCM Library Components" on page 85
- "Synchronizer Methods" on page 85

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_axi_x2h.

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

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_x2h_bcm05.v	FIFO controller interface for one of two clock domains instantiated in DW_axi_x2h_bcm07.v	DW_fifoctl_if Submodule of DW_fifoctl_s2_sf
DW_axi_x2h_bcm06.v	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_axi_x2h_bcm07.v	Synchronous (Dual-Clock) FIFO Controller with Static Flags	DW_fifoctl_s2_sf
DW_axi_x2h_bcm21.v	Single Clock Data Bus Synchronizer	DW_sync
DW_axi_x2h_bcm57.v	Sync. Write-Port, Asynchronize Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff

A.2 Synchronizer Methods

This section describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_axi_x2h to synchronize clock domain crossing signals.

This appendix contains the following sections:

- "Synchronizers Used in DW_axi_x2h" on page 86

- "Synchronizer 1: Simple Double Register Synchronizer (DW_axi_x2h)" on page 86
- "Synchronizer 2: Synchronous (Dual-clock) FIFO Controller with Static Flags (DW_axi_x2h)" on page 87

☞ **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.2.1 Synchronizers Used in DW_axi_x2h

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_x2h 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_x2h

Synchronizer Module File	Sub Module File	Synchronizer Type and Number
DW_axi_x2h_bcm21.v		Synchronizer 1: Simple Multiple register synchronizer
DW_axi_x2h_bcm07.v	DW_axi_x2h_bcm05.v DW_axi_x2h_bcm21.v	Synchronizer 2: Synchronous dual clock FIFO Controller with Static Flags

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_axi_x2h)

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 and mhclk are asynchronous (X2H_CLK_MODE =0) then DW_axi_x2h_bcm21 is instantiated inside the core for synchronization. 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)

Configured as: *f_sync_type* = 2, *tst_mode* = 1
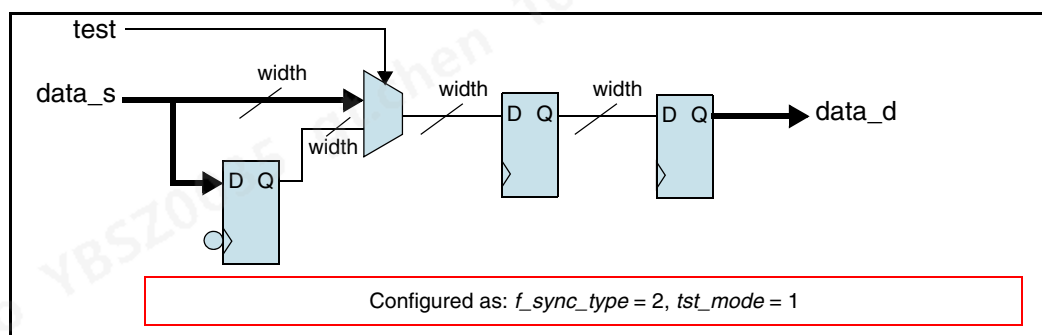
A.2.3　Synchronizer 2: Synchronous (Dual-clock) FIFO Controller with Static Flags (DW_axi_x2h)

DW_axi_x2h_bcm07 is a dual independent clock FIFO RAM controller. It is designed to interface with a dual-port synchronous RAM. The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Figure A-2 shows the block diagram of Synchronizer 2.

Figure A-2　Synchronizer 2 Block Diagram

B

Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
active AXI command	A command that has been generated by an AXI master and accepted by an AXI slave, but where the corresponding command has not completed.
	A read command completes when the last data beat of the read burst completes; that is, when the AXI slave asserts RLAST and RVALID and when the AXI master asserts RREADY.
	A write command completes when the AXI slave returns the write response and is accepted by the AXI master; that is, when the AXI slave asserts BVALID and the AXI master asserts BREADY
active DW_axi_x2h command	A command that has been generated by an AXI master on the primary bus and accepted by an AXI slave on the secondary bus but where the corresponding command has not completed.
	In terms of the DW_axi_x2h, a read command completes when the last data beat of the read burst completes on the primary bus; that is, when the DW_axi_x2h Master Port asserts RLAST and RVALID and when the external AXI master asserts RREADY.
	In terms of the DW_axi_x2h, a write command completes when the DW_axi_x2h Master Port returns the write response and it is accepted by the AXI master on the primary bus; that is, when the DW_axi_x2h Master Port asserts BVALID and the external AXI master asserts BREADY.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
A-device model	A usb_device_vmt model configured as SRP capable host only device and dual role OTG A-device.
AHB	Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (ARM Limited specification).

altered transaction	Transaction that the DW_axi_x2h has either downsized, upsized, endian-mapped, or fanned out to multiple write interleaving channels. Similarly, a non-altered transaction is a transaction that none of these operations are performed on.
AMBA	Advanced Microcontroller Bus Architecture — a trademarked name by ARM Limited that defines an on-chip communication standard for high speed microcontrollers.
APB	Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (ARM Limited specification).
APB bridge	DW_apb submodule that converts protocol between the AHB bus and APB bus.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
arbiter	AMBA bus submodule that arbitrates bus activity between masters and slaves.
AXI	Advanced eXtensible Interface Bus — a trademarked name by ARM Limited that defines an on-chip protocol for high-performance, high-frequency system designs.
B-device model	A usb_device_vmt model configured as non-OTG device (USB 2.0 standard function), SRP capable peripheral only device and dual role OTG B-device.
backward control path	For the read/write command channels and the write data channel, the backward control path is in the direction from ARREADY/AWREADY/WREADY from an external slave to ARREADY/AWREADY/WREADY to an external master. For the read data and write response channels, the backward control path is in the direction from RREADY/BREADY from an external master to RREADY/BREADY to an external slave
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.
beat	A single data transfer, usually in a burst transaction. For example, on the AHB, an INCR4 transaction has four beats and INCR8 has eight beats.
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.
blocked transaction	Transaction is considered blocked when it is not initiated before the preceding transaction has completed.
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.

burst	Number of data transfers in a transaction.
bus bridge	Logic that handles the interface and transactions between two bus standards, such as AHB and APB. See APB bridge.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command queue	First-in-first-out queue from which a model command execution engine retrieves commands; see also active command queue. VMT models support multiple command queues.
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.
core	Any configurable block of synthesizable IP that can be instantiated as a single entity (VHDL) or module (Verilog) in a design. Core is the preferred term for a big piece of IIP. Anything that requires coreConsultant for configuration, as well as anything in the DesignWare Cores library, is a core.
core developer	Person or company who creates or packages a reusable core. All the cores in the DesignWare Library are developed by Synopsys.
core integrator	Person who uses coreConsultant or coreAssembler to incorporate reusable cores into a system-level design.
coreAssembler	Synopsys product that enables automatic connection of a group of cores into a subsystem. Generates RTL and gate-level views of the entire subsystem.
coreConsultant	A Synopsys product that lets you configure a core and generate the design views and synthesis views you need to integrate the core into your design. Can also synthesize the core and run the unit-level testbench supplied with the core.
coreKit	An unconfigured core and associated files, including the core itself, a specified synthesis methodology, interfaces definitions, and optional items such as verification environment files and core-specific documentation.
cycle command	A command that executes and causes HDL simulation time to advance.
buffered write transaction	Occurs when the write response is provided by an intermediate instance and not by the actual payload sink. The actual write response is typically discarded unless it indicates an erroneous completion.
byte reordering	Changing the location of bytes in the data word across the bridge (little-endian to big-endian).

data interleaving	Refers to a data channel (read or write) source issuing data for different transactions without waiting for the data beats of previous transactions to complete.
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.
Design View	A simulation model for a core generated by coreConsultant.
design_dir	The VIP equivalent to workspace.
DesignWare Synthesizable Components for AMBA	The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs.
DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
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.
downsizing	Functional operation of the DW_axi_x2h bridge in a configuration where the primary bus data width or burst length width is larger than those of the secondary bus. Occurs where a transaction entering the DW_axi_x2h from the primary bus may have its LEN and/or SIZE attributes changed or broken into smaller transactions.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multibyte word; see also little-endian and big-endian.
forward control path	For the read/write command channels and the write data channel, the forward control path is in the direction from AWVALID/ARVALID/WVALID from an external master to AWVALID/ARVALID/WVALID to an external slave.
	For the read data and write response channels, the forward control path is in the direction from RVALID/BVALID from an external slave to RVALID/BVALID to an external master.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
gasket	A model that serves as a connection layer between the VMT models and the DP/DM interface.
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.
HNP	Host Negotiation Protocol.
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.
interleaving depth	Refers to the number of different data parts of transactions that can be active at any one time on that channel (read or write data). Interleaved transactions must have different IDs.
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.
locked sequence	A sequence of locked read or write commands from an external AXI master locking the read and write command channels of the addressed slave. The locking sequence includes both the initial locking command and the final unlocking command that is used to terminate the locked sequence.
locking command	Initial locking command of the locked sequence.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
master port	Port of the DW_axi_x2h that connects to an external master on the primary bus.
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.
non-transaction altering configuration	Configuration of the DW_axi_x2h bridge where altered transactions are not possible.
non-OTG device	The Device model configured as USB 2.0 standard function and does not contain OTG features.

OTG	On-The-Go.
payload, AXI	For the read/write command channels, the read/write address and control signals. For the read/write data channels, the read/write data and control. For the write response channel, the burst response and control. The payload is always in the direction of the forward control path and is specific to the AXI channel.
payload source	Source of the AXI channel payload. The payload source for the read/write command channels and the write data channel is an AXI master. The payload source for the read data and write response channel is an AXI slave.
payload sink	Sink of the AXI channel payload. The payload sink for the read/write command channels and the write data channel is an AXI slave. The payload sink for the read data and write response channel is an AXI master.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
posted transaction	Transaction is considered posted when it is initiated before the preceding transaction has completed.
primary and secondary	Refers to the buses with respect to a particular instantiation of the DW_axi_x2h. Primary and secondary is in reference to the direction of the initiation of transactions through the DW_axi_x2h. Primary refers to the transaction issuing bus, and secondary is the transaction responding bus.
read transaction	Composed of the following two independent phases: 1. Read command phase 2. Read data phase; signalling of last beat of read data terminates read transaction
resized transaction	Transaction that the DW_axi_x2h has either upsized or downsized.
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.
secondary and primary	Refers to the buses with respect to a particular instantiation of the DW_axi_x2h. Primary and secondary is in reference to the direction of the initiation of transactions through the DW_axi_x2h. Primary refers to the transaction issuing bus, and secondary is the transaction responding bus.
slave	Device or model that is controlled by and responds to a master.
slave port	Port of the DW_axi_x2h that connects to an external slave on the secondary bus.
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.
SRP	Session Request Protocol.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
subsystem	In relation to coreAssembler, highest level of RTL that is automatically generated.
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.
transaction	A read or write operation on the AXI or AHB that involves one or more data transfers. For example, on the AHB, an INCR8 write is considered as a single transaction; a SINGLE read or an INCR burst of unspecified length is also considered a single transaction. Any AXI read or write is considered to be a single transaction, regardless of its length.
transaction ordering	Order in which transactions are responded to. Responses to a series of transactions could be returned in an order different to the order in which they were issued; this is referred to as an out-of-order transaction. Re-ordered transactions must have different IDs.
transfer	A single sequence initiated by VALID and ended by READY. A write command phase, read command phase, write data phase, read data phase, write response phase are each, by themselves, a transfer.
upsizing	Functional operation of the DW_axi_x2h bridge in a configuration where the primary bus data width is smaller than the secondary bus data width. Occurs here a transaction entering the DW_axi_x2h from the primary bus may have its LEN and/or SIZE attributes changed to transform it into a transaction of the largest SIZE allowable on the secondary bus.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wait state	A bus cycle where the device initiating the transfer must wait for a response.
waited transfer	AXI channel is waited if the payload sink de-asserts its *READY signal when the payload source asserts its *VALID signal. Waits states are inserted into the AXI channel transfer.

workspace	A network location that contains a personal copy of a component or subsystem. After you configure the component or subsystem (using coreConsultant or coreAssembler), the workspace contains the configured component/subsystem and generated views needed for integration of the component/subsystem at the top level.
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.
write transaction	Composed of the following three independent phases: 1. Write command phase 2. Write data phase 3. Write response phase; terminates the write transaction
zero-cycle command	A command that executes without HDL simulation time advancing.

Index