SYNOPSYS®

DesignWare® DW_axi_hmx

Databook

DW_axi_hmx – Product Code

2.03a
March 2020

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com

Contents

Revision History

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

Version	Date	Description
2.03a	March 2020	■ Revision version change for 2020.03a release ■ Added Appendix 8, "Basic Core Module (BCM) Library" ■ Updated synthesis results in "Performance" on page 66 ■ Updated "Verification" on page 61 ■ "Signal Descriptions" on page 39, "Parameter Descriptions" on page 31, "Internal Parameter Descriptions" on page 59 auto-extracted from the RTL
2.02a	February 2018	■ Revision version change for 2018.02a release ■ Updated synthesis results in "Performance" on page 66 ■ 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 39, "Parameter Descriptions" on page 31, "Internal Parameter Descriptions" on page 59 auto-extracted from the RTL with change bars
2.01a	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 39 and "Parameter Descriptions" on page 31 auto-extracted from the RTL ■ Added chapter "Internal Parameter Descriptions" on page 59 ■ Updated area and power numbers in "Performance" on page 66
2.00a	October 2014	■ Version change for 2014.10a release. ■ Added "AHB INCR Writes" section in "Functional Description" chapter. ■ Added AMBA 4 AXI support. ■ Updated "Performance" section in the "Integration Considerations" chapter. ■ Removed the HMX_AXI_DATA_WIDTH parameter as it was a duplicate of the HMX_DATA_WIATDH parameter, and replaced all occurrences of HMX_AXI_DATA_WITDH with HMX_DATA_WIDTH in Parameters chapter page 32.

Version	Date	Description
1.08b	Jun 2013	Version change for 2013.06a release.
1.08a	May 2013	Version change for 2013.05a release. Updated the document template.
1.07b	Oct 2012	Added the part number on the cover and in Table 1-1.
1.07b	Oct 2011	Version change for 2011.10a release.
1.07a	Jun 2011	Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface.
1.07a	Sep 2010	Added material about limitations with respect to defined length burst support; corrected names of include files and vcs command used for simulation
1.04a	Dec 2009	Correct waveforms in Quasi-Synchronous HCLK illustration; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.04a	Aug 2009	Updated definition of wstrb signal and HMX_AXI_DATA_WIDTH parameter name.
1.04a	May 2009	Removed references to Quickstarts, as they are no longer supported.
1.04a	Apr 2009	Corrected first bullet for AHB-to-AXI Bridge Solution
1.04a	Jan 2009	Corrected description for HMX_BLOCK_WRITE in “Transaction Blocking”
1.03a	Oct 2008	Version change for 2008.10a release.
1.02b	Sep 2008	Enhanced description of Burst Length Translation; corrected list of requirements for up- or downsizing the data bus width
1.02b	Jun 2008	Version change for 2008.06a release.
1.02a	Oct 2007	Fixed HMX_BLOCK_WRITE default value; fixed red circle in system diagram;
1.02a	Sep 2007	Enhanced Write Responses (bresp) description.
1.02a	Jul 2007	HMX_BLOCK_WRITE parameter description changed for enhancement; “Block Writes” section changed to ““Transaction Blocking”,” with description of new functionality.
1.01b	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information about the DesignWare Advanced eXtensible Interface HMX (DW_axi_hmx) gasket. This component conforms to the AMBA 3 AXI and AMBA 4 AXI specification defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

The information in this databook includes a functional description, signal and parameter descriptions, as well as information on how you can configure, create RTL for, simulate, and synthesize the component using Synopsys coreAssembler or coreConsultant. This manual also provides 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_hmx.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_hmx.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_axi_hmx 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_hmx.
- Chapter 7, "Integration Considerations" includes information you need to integrate the configured DW_axi_hmx into your design.
- Appendix A, "Glossary" provides a glossary of general terms.

Related Documentation

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

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

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

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

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

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

1 Product Overview

The DW_axi_hmx module connects a single AHB (Advanced High-performance Bus) master to the AXI (Advanced eXtensible Interface) bus.

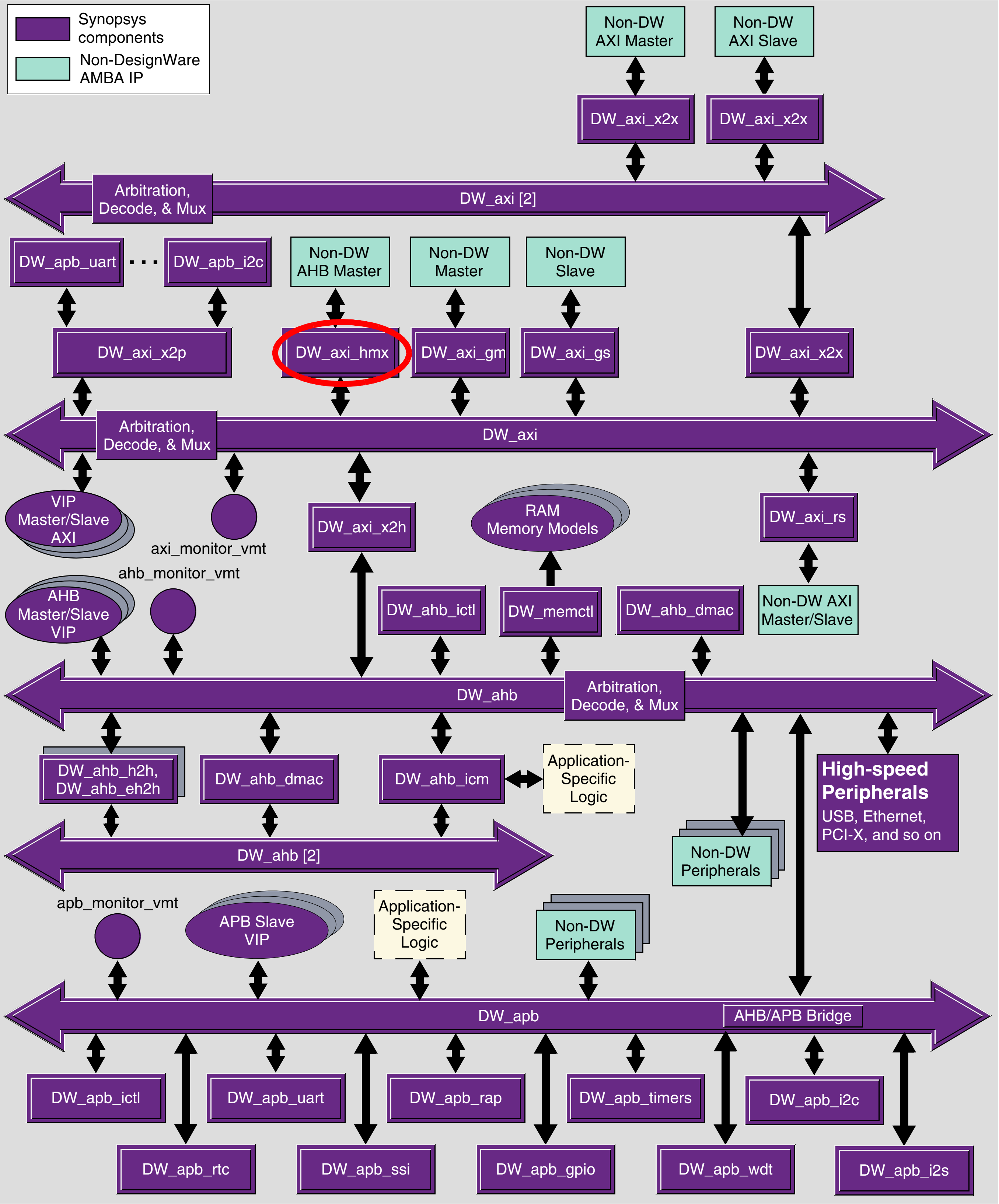
1.1 DesignWare Synthesizable Components for AMBA System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB 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_hmx in a Complete System



Note When the DW_axi_hmx is used with the DW_ahb_eh2h to bridge from AHB to AXI—that is, to allow masters on an AHB bus to access slaves on an AXI bus—there is a performance issue relating to write transfers. The DW_ahb_eh2h converts all write transactions to undefined length INCR writes, and the DW_axi_hmx converts these undefined length INCR writes to multiple AXI writes of length 1.

Although this does not affect the data throughput rate from the DW_axi_hmx, it does result in some inefficiencies on the AXI bus:

- Since every beat of write data is associated with a different address transfer, each address transfer must win arbitration on the AXI bus before the associated data beat is allowed to reach the slave.
- For slaves—for example, memory controllers—that are optimized for transfers of a particular burst length, there can be a reduction in throughput.

This issue can be resolved by communicating the exact size of the AHB-INCR transfer to the component in advance. The HMX_ADD_INCR_LEN_PORT parameter when enabled, adds an AHB non-protocol signal hburst_len to the interface, which intimates the length of the INCR transfer to the component. DW_axi_hmx can then use this value to initiate higher length INCR transfers rather than SINGLES.

You can connect, configure, synthesize, and verify the DW_axi_hmx within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_axi_hmx component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

The DW_axi_hmx connects a single AHB master to the AXI interconnect fabric (DW_axi) or any AXI slave. Read and write transactions through the DW_axi_hmx are received on the AHB master port, translated to the AXI protocol and sent out from the DW_axi_hmx AXI master interface.

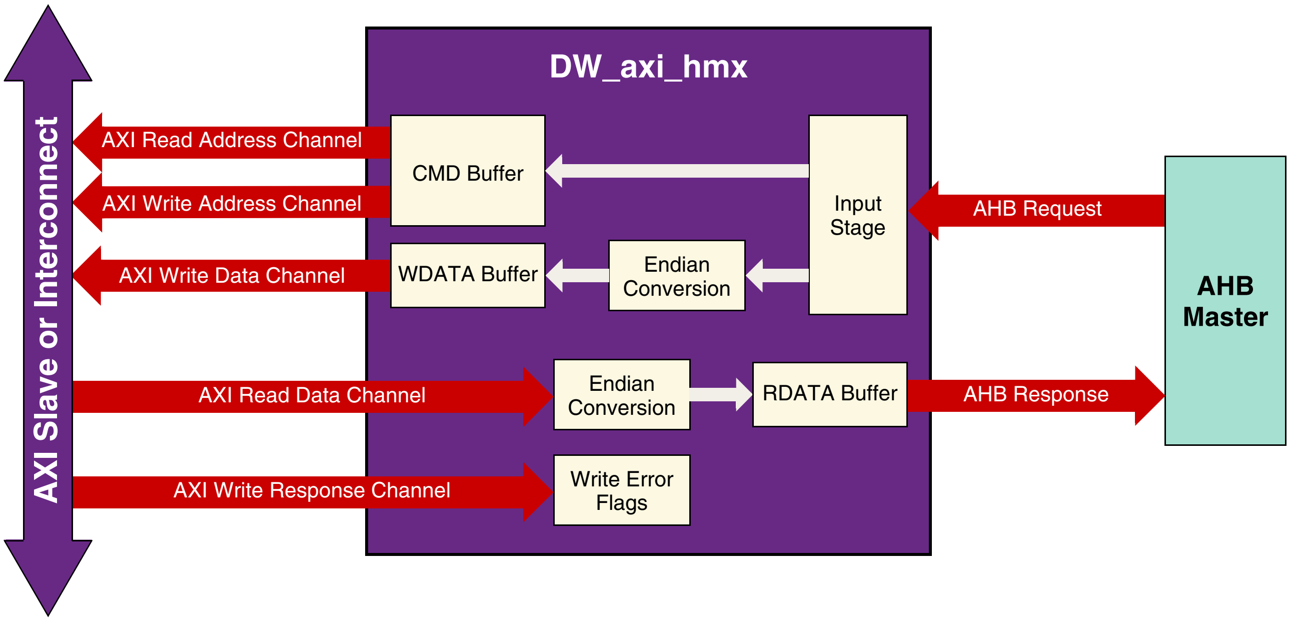
An AHB master is attached to the AHB input side of DW_axi_hmx. Note that this is a direct connection to an AHB master, not a connection to the DW_ahb interconnect fabric. The AHB side of the DW_axi_hmx is therefore referred to as an AHB master port, since it attaches to an AHB master.

The AXI side of the DW_axi_hmx is an AXI master interface. Most applications connects the DW_axi_hmx AXI master interface to the DW_axi interconnect fabric for use in an AXI subsystem. It is possible to connect the DW_axi_hmx directly to any single AXI slave, as well.

1.2.1 DW_axi_hmx Block Diagram

Figure 1-2 illustrates the DW_axi_hmx block diagram.

Figure 1-2 DW_axi_hmx Block Diagram



1.3 Features

The DW_axi_hmx supports the following features:

- Connects a single AHB master to an AXI interconnect or a single AXI slave
- Complies with the AMBA 3 AXI or AMBA 4 AXI protocol specification
- Single clock design; quasi-synchronous hclk available for slower AHB interfaces
- Buffered write transactions; AHB writes transaction completes before AXI write response
- Buffered read data (optional)
- Support for AXI error responses
- Activity status signal for low-power support
- Data endian conversion
- AHB locked transaction support to AXI 3 slave (including automated AXI unlocking command)
- Support for all AHB burst types
- Timing mode options to ease timing closure
- Option to do AHB write blocking; AXI write response required before next AHB write can start
    - Automated AHB hprot to AXI ar/awcache and ar/awprot translation
    - Configurable ar/awprot secure bit

The following are feature restrictions of DW_axi_hmx:

- Connection of an AHB subsystem to AXI; requires DW_ahb_eh2h
- Up- or downsizing of the data bus width

  - Requires DW_axi_x2x for upsizing
  - Requires DW_axi_x2x or DW_axh_eh2h for downsizing
- Fully asynchronous AHB clock; requires DW_ahb_eh2h
- Out-of-order read or write transaction depth (no out-of-order AXI transaction completion allowed)
- Write data interleaving depth of 1 (no AXI write data interleaving)
- AXI exclusive access transactions not generated
- Split or retry responses on the AHB not generated
- No software interface
- Multiple outstanding AXI read transactions not supported

1.4 DW_axi_hmx Terminology

The following terms are used throughout this databook.

- **Active read/write command** – A command that has been generated by an AXI master and accepted by an AXI slave but has not completed.

  A read command completes when the last data beat of the read burst completes; for instance, when the AXI slave asserts rlast and rvalid signals, and the AXI master asserts rready.

  A write command completes when the AXI slave returns the write response and it is accepted by the AXI master; for instance, when the AXI slave asserts bvalid, and the AXI master asserts bready.
- **Beat** – A single data transfer.
- **Burst** – Number of data transfers in a transaction.
- **AXI transfer** – A single sequence initiated by valid and ended by ready. A write command phase, read command phase, write data phase, read data phase, and write response phase are each, by themselves, a transfer.
- **Locked sequence** – A sequence of locked read or write commands from an AXI master that locks 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.
- **Byte reordering** – Changing the location of bytes in the data word across the Bridge (Little Endian or Big Endian).
- **Write transaction** – An AXI write transaction is composed of the following three independent phases:
  - Write command phase.
  - Write data phase.
  - Write response phase. This terminates the write transaction.
- **Read transaction** – An AXI read transaction is composed of the following two independent phases:

- ❑ Read command phase.
- ❑ Read data phase. The signalling of the last beat of read data terminates the read transaction.

- **Buffered (write) transaction** – In a buffered write transaction, the write response is provided by an intermediate instance and not by the actual data destination.
- **Blocked transaction** – A transaction is considered blocked when it is ignored by the receiver until the preceding transaction has completed.

1.5 Standards Compliance

The DW_axi_hmx component conforms to the *AMBAAXI and ACE Protocol Specification* from ARM. Readers are assumed to be familiar with these specifications.

1.6 Verification Environment Overview

The DW_axi_hmx 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.

The "Verification" on page 61 chapter discusses the testbench environment and provides an overview of the tests that are used to verify DW_axi_hmx when you run component-level simulation.

1.7 Licenses

Before you begin using the DW_axi_hmx, you must have a valid license. For more information, see *"Licenses"* section in the *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide.*

Source code for this component is available on a per-project basis as a DesignWare Core. Contact your local sales office for the details.

1.8 Where To Go From Here

At this point, you may want to get started working with the DW_axi_hmx 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_hmx component, see *"Overview of the coreConsultant Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

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

2 Functional Description

The DW_axi_hmx connects an AMBA 2.0 AHB master to the AMBA 3 AXI or AMBA 4 AXI interconnect fabric (DW_axi) or any AMBA 3 AXI or AMBA 4 AXI slave. Most applications connect the DW_axi_hmx AXI master interface to the DW_axi interconnect fabric for use in an AXI subsystem.

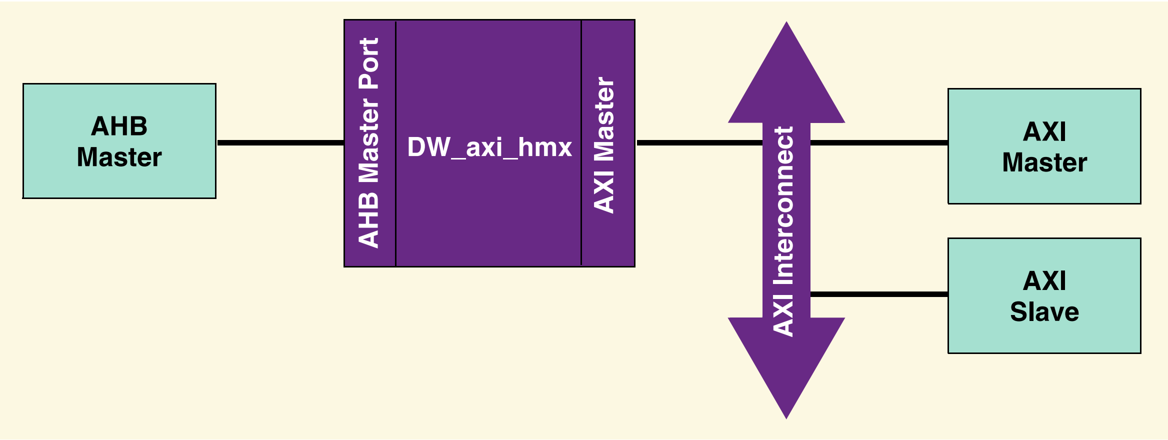
It is possible to connect the DW_axi_hmx directly to any single AXI slave, as well. The DW_axi_hmx AHB interface connects directly to an AHB master, not the DW_ahb AHB interconnect fabric. Read and write transactions through the DW_axi_hmx are received from the AHB master, translated to the AXI protocol and sent out from the DW_axi_hmx AXI master interface.

Note There cannot be more than thirty-two outstanding transactions at any one time in the DW_axi_hmx.

2.1 DW_axi_hmx Application Examples

As shown in Figure 2-1, the DW_axi_hmx has an AHB master port and an AXI master interface to connect an AHB master and AXI interconnect.

Figure 2-1 DW_axi_hmx Interface

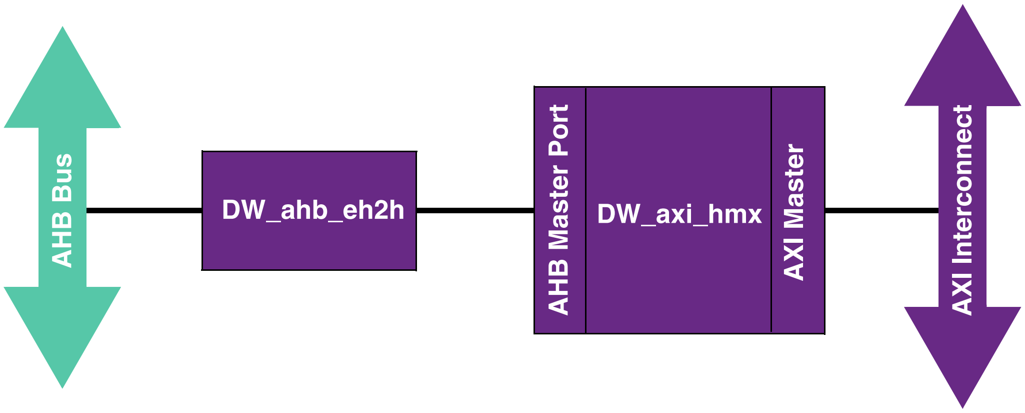


The DW_axi_hmx can be used in conjunction with the DW_ahb_eh2h bridge to form an AHB-to-AXI bridge. This AHB-to-AXI bridge solution, illustrated in Figure 2-2, allows an AHB subsystem to gain access to slaves in an AXI subsystem; for a block diagram of this AHB-to-AXI bridge solution.

For an AXI-to-AHB bridge, allowing an AXI master or subsystem access to slaves in an AHB subsystem, see the DesignWare component DW_axi_x2h. This bridge solution offers the following features:

- Downsizing of data for differing AHB/AXI data widths
- Support for endianness conversion
- Support for synchronous or asynchronous clock domains
- Off-loading of the AHB bus when a transaction on the AXI bus is ongoing

Figure 2-2 DW_axi_hmx with DW_ahb_eh2h for AHB-to-AXI Bridge Solution



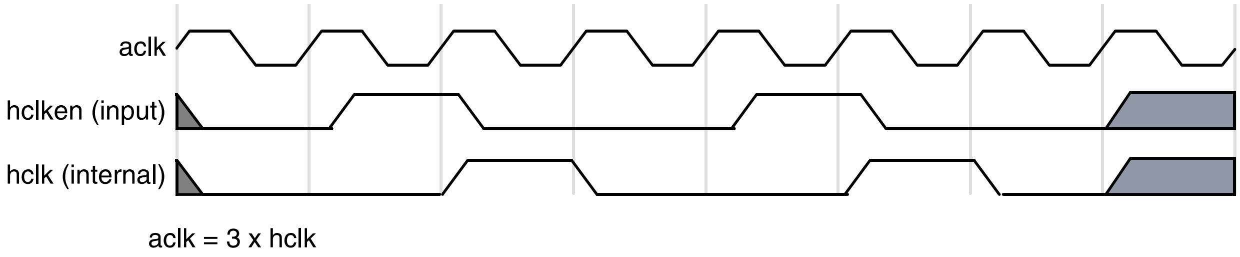
For a full list of the DW_ahb_eh2h features, see the *DesignWare DW_ahb_eh2h Databook*.

2.2 Quasi-Synchronous HCLK

The DW_axi_hmx is driven off a single clock, aclk. An enable signal—hclken—enables the internal clock on the AHB side of the DW_axi_hmx—hclk—to be slowed down. The clock on the AXI side, aclk, is therefore an integer multiple of the clock on the AHB side. Asynchronous clocks can be used when DW_axi_hmx is used in conjunction with DW_ahb_eh2h as previously shown in this section.

Figure 2-3 shows the quasi-synchronous behavior of the hclk in relation to aclk.

Figure 2-3 Quasi-Synchronous HCLK



2.3 Write Transactions

AHB write transactions are translated and split into the CMD and WDATA buffers. These buffers initiate AXI write address channel and write data channel transfers. On the AXI write data channel, write data from AHB is provided as it becomes available, with wait states inserted (wvalid deasserted) if AHB write data is not continuously supplied or if the AHB clock is slower than the AXI clock. If AXI is not able to accept the write data immediately (wready deasserted) and the WDATA buffer fills, then DW_axi_hmx de-asserts hready, stalling the AHB write data until wready is asserted on AXI and WDATA buffer space becomes available.

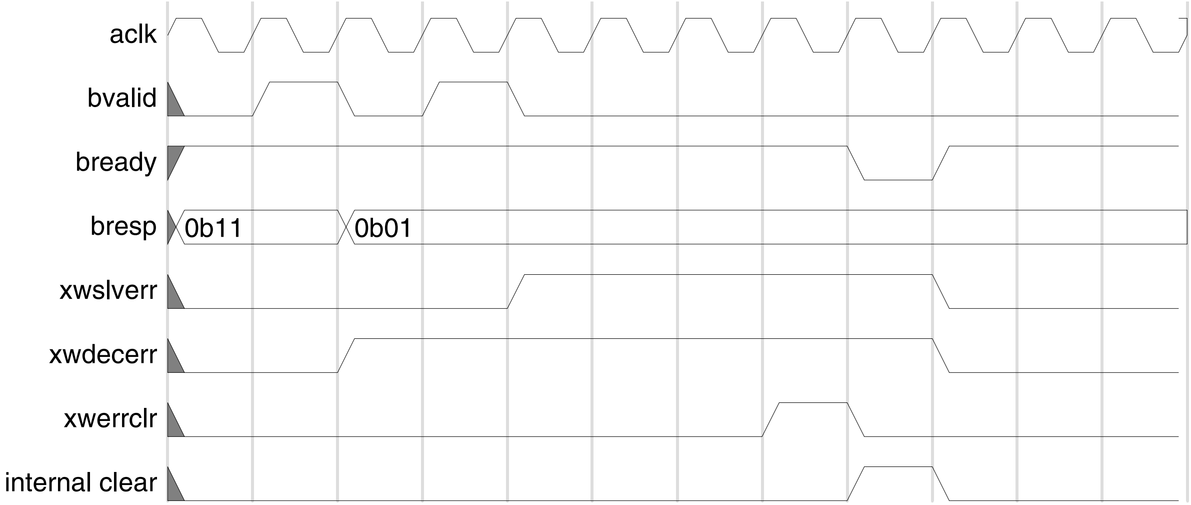
2.3.1 Write Responses (bresp)

AHB writes require a response (hresp) for each write data beat. Translating all AHB writes into single-beat transactions on AXI and waiting for an AXI write response for each beat would result in very bad write performance. Instead DW_axi_hmx responds back to the AHB master with an OKAY response for each data beat. This is referred to as write buffering. Write buffering by DW_axi_hmx anticipates a successful write when the AXI write transaction completes.

If the AXI write transaction completes with a write response of either SLVERR or DECERR, DW_axi_hmx handles these error conditions by asserting an output signal (xwslverr or xwdecerr) that can be used as an interrupt. The only way to de-assert these signals is by asserting an input signal (xwerrclr).

Both flags are cleared at the same time in the cycle following an active xwerrclr, as illustrated in Figure 2-4. During the clear, bready is driven low. If xwerrclr is asserted high for multiple cycles, bready is driven low accordingly. Cycles are measured by aclk and do not depend on hclken.

Figure 2-4 Setting and Clearing of Write Error Flags



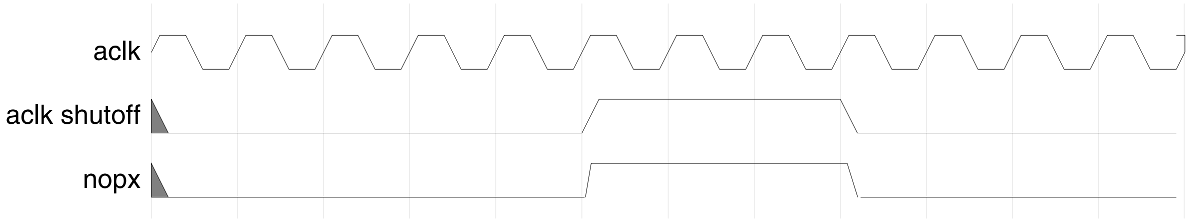
While xwerrclr is asserted, the DW_axi_hmx holds its bready output low (1'b0) in order to ensure that an assertion of xwerrclr clears only errors that occurred prior to the assertion of xwerrclr. This means that tying xwerrclr high (1'b1) breaks the operation of the DW_axi_hmx.

Note that write error responses have no effect on the operation of the DW_axi_hmx other than setting the appropriate output status signals.

2.4 Low-Power Support

The DW_axi_hmx includes a flag, nopx, that is asserted high when there are no pending commands, the WDATA and RDATA buffers are empty, and AHB hlock is not asserted. Since outstanding commands are not tracked in DW_axi_hmx, the nopx signal is asserted even when write responses for outstanding writes or read data for outstanding reads have not completed. The nopx signal can be used by the system clock controller to determine when system clocks can be shut off to go to a low-power state, as illustrated Figure 2-5.

Figure 2-5 Low-Power Request Sequence



2.5 Endianness Conversion

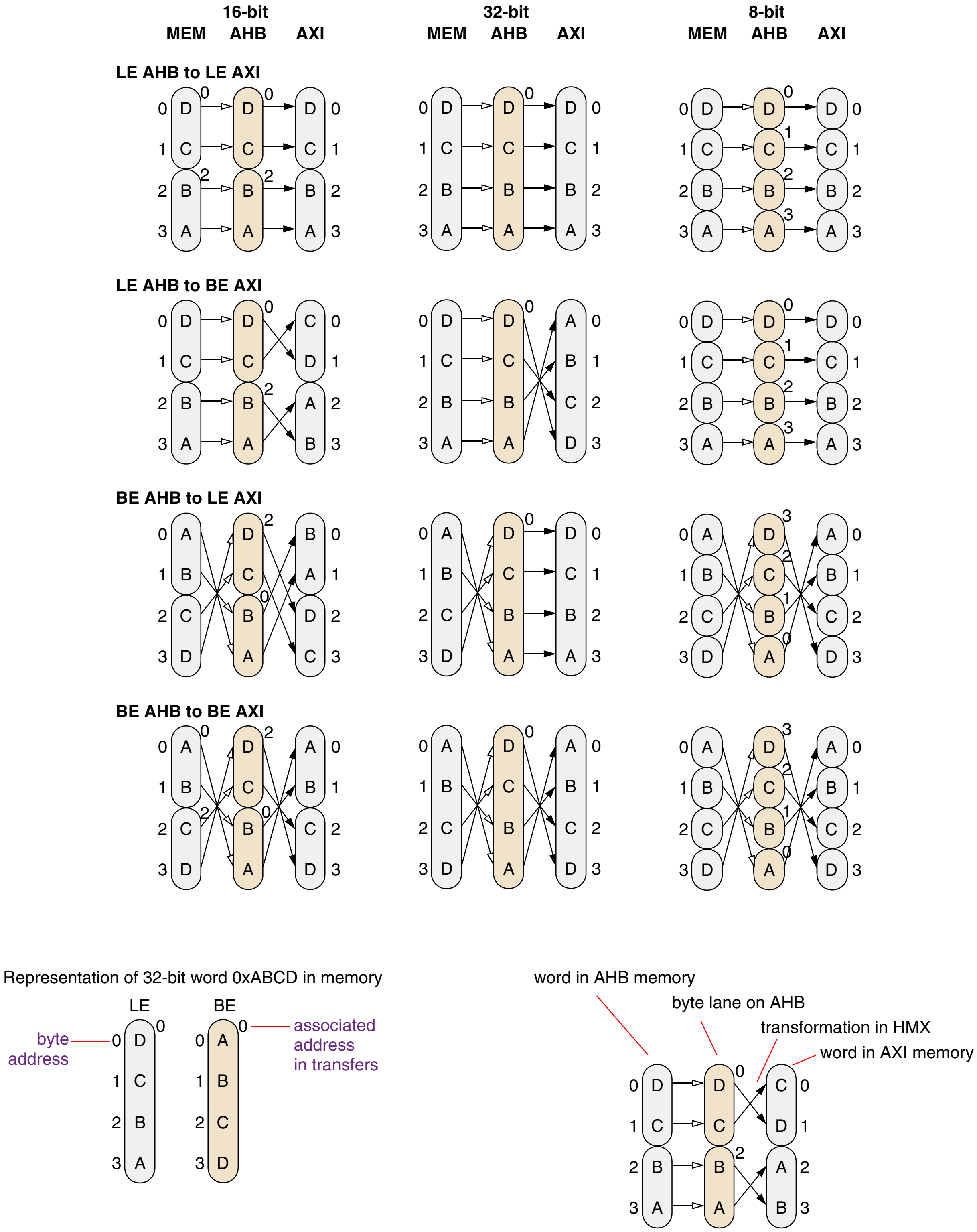
DW_axi_hmx provides support for data endianness conversion between the AHB and AXI. The endianness of both the AHB and AXI buses is configured by setting two hardware configuration parameters, HMX_ENDIAN_AHB and HMX_ENDIAN_CONVERT. Table 2-1 shows how to set the parameters for the desired endianness setting and conversion.

Table 2-1 Endianness Conversion from AHB to AXI

HMX_ENDIAN_AHB	HMX_ENDIAN_CONVERT	Incoming AHB Bus	Outgoing AXI Bus
True	True	Big Endian	Little Endian
True	False	Big Endian	Big Endian
False	True	Little Endian	Big Endian
False (default)	False (default)	Little Endian	Little Endian

Figure 2-6 also illustrates these endian conversion scenarios.

Figure 2-6 DW_axi_hmx Endianness Modes



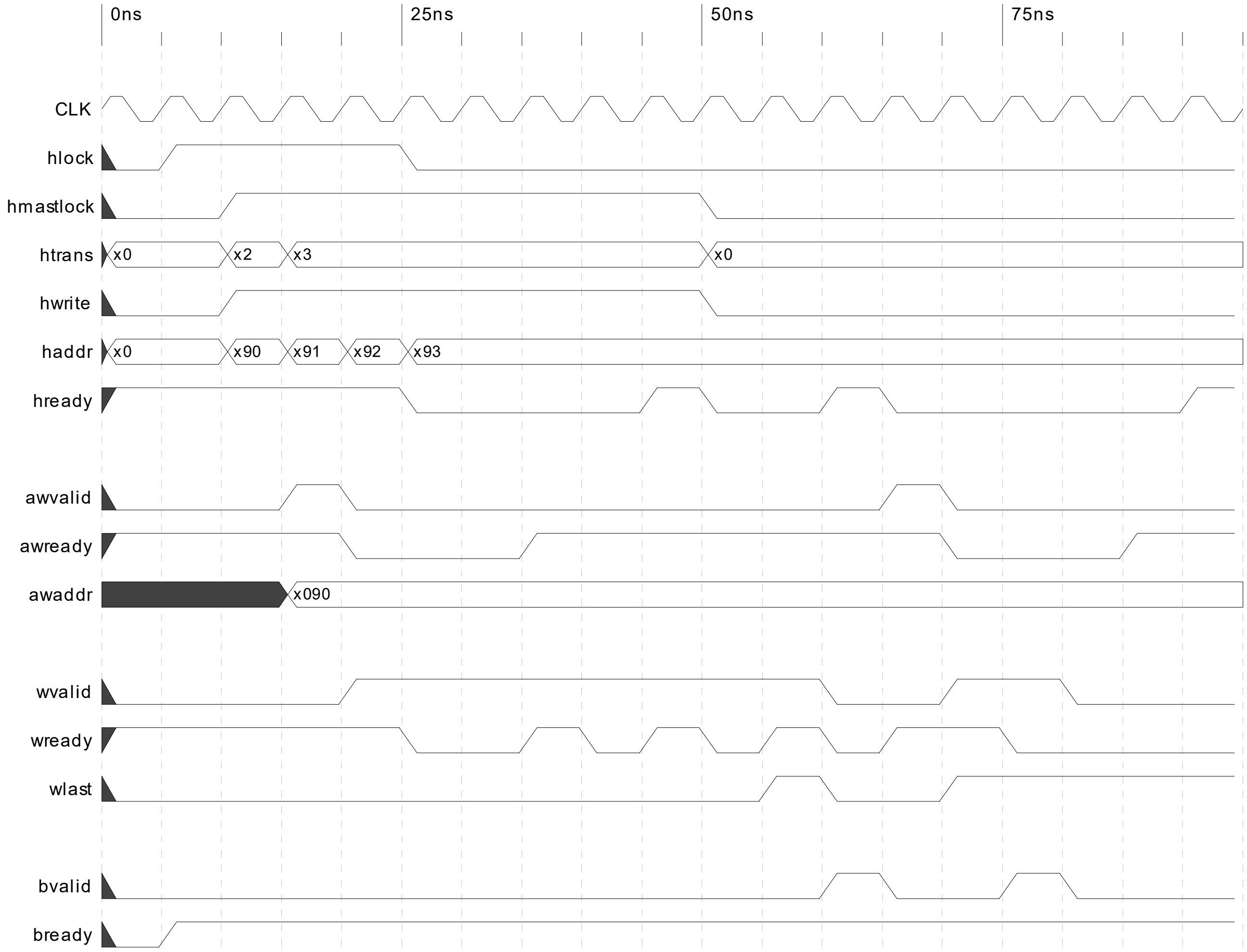
2.6 Locked Transfer Support in AXI 3 Mode

Locked transfers are supported only in an AXI 3 mode and are not supported in an AXI 4 mode. The AHB master should ensure that it does not initiate a locked transfer in AXI 4 mode as it is ignored by DW_axi_hmx.

In AXI 3 mode, a locked transfer is indicated on hlock by the AHB master. In the AXI 3 protocol, a sequence of locking transactions is completed with an unlocking transaction. This is usually the last functional transaction in a locked sequence. However, DW_axi_hmx cannot predict the last functional transaction in an AHB locked sequence.

To complete a locked sequence on AXI, DW_axi_hmx must issue an unlocking transaction to finish the sequence of locked transactions, which is illustrated in Figure 2-7.

Figure 2-7 Unlock Sequence



An unlocked transaction is generated whenever there is a falling edge of hlock. The unlocked transaction is a single INCR write with all write strobes turned off. This transaction has no functional effect on the destination AXI slave, other than completing the locked sequence, since the write strobes disable all write data. The address of the write is either the last address used by DW_axi_hmx for either read or write, or it can be a predefined static address. The type of address that is used can be determined when DW_axi_hmx is configured. The control signals for *prot, *cache, and *size are adopted from the transaction issued before the unlocked transaction.

2.7 Burst Length Translation

Most AHB hburst types are translated to the corresponding AXI arburst/awburst types of length arlen/awlen. For example, an AHB burst with hburst type INCR16 translates to an AXI burst with arburst/awburst type INCR and arlen/awlen of 15 (16 data transfers).

The DW_axi_hmx does not perform upsizing or downsizing; thus, a 32-bit AHB bus transfer translates to a 32-bit AXI bus transfer, regardless of how big the AXI bus is. For example, if bridging between a 32-bit AHB bus and a 128-bit AXI bus, an AHB INCR4 translates to an AXI transfer with arburst/awburst type INCR and arlen/awlen of 3 (4 data transfers each 32-bits), and not an AXI SINGLE transfer of 128 bits.

Note When the DW_axi_hmx is used with the DW_ahb_eh2h to bridge from AHB to AXI—that is, to allow masters on an AHB bus to access slaves on an AXI bus—there is a performance issue relating to write transfers. The DW_ahb_eh2h converts all write transactions to undefined length INCR writes, and the DW_axi_hmx converts these undefined length INCR writes to multiple AXI writes of length 1.

Although this does not affect the data throughput rate from the DW_axi_hmx, it does result in some inefficiencies on the AXI bus:

- Since every beat of write data is associated with a different address transfer, each address transfer must win arbitration on the AXI bus before the associated data beat is allowed to reach the slave.
- For slaves—for example, memory controllers—that are optimized for transfers of a particular burst length, there can be a reduction in throughput.

This issue can be resolved by communicating the exact size of the AHB-INCR transfer to the component in advance. The parameter HMX_ADD_INCR_LEN_PORT when enabled, adds an AHB non-protocol signal hburst_len to the interface which intimates the length of the INCR transfer to the component. DW_axi_hmx can then use this value to initiate higher length INCR transfers rather than SINGLES.

2.7.1 AHB INCR Writes

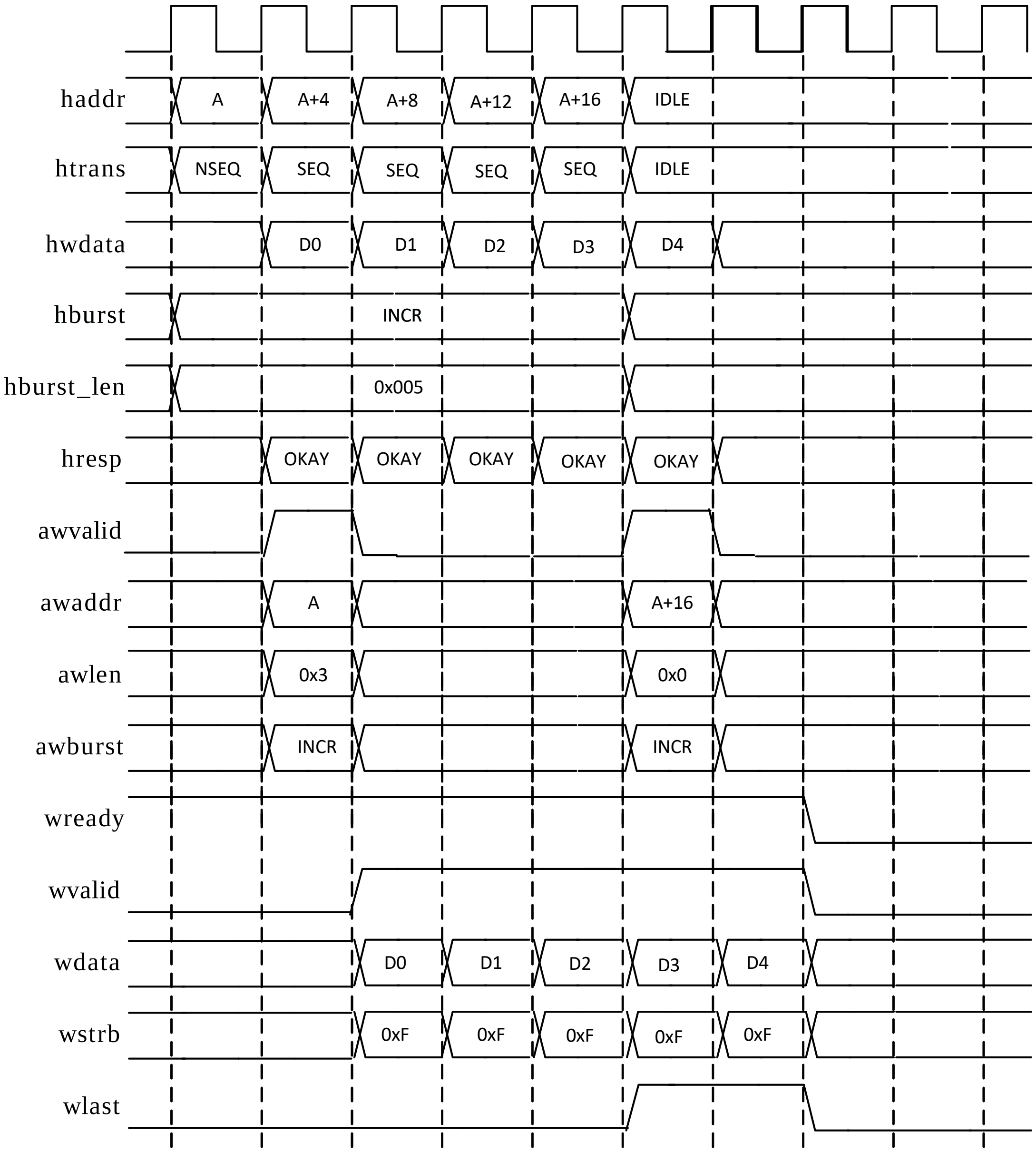
The only type of AHB transfer that may have its length changed by the DW_axi_hmx is an AHB INCR. By default, any undefined length of AHB INCR bursts are translated as AXI SINGLE transactions. AXI bursts of higher length are possible if the parameter HMX_ADD_INCR_LEN_PORT is enabled. This adds custom input port hburst_len to the AHB interface through which the AHB master can intimate the exact size of AHB undefined length INCR transfer to the module, enabling AXI transfers of higher burst length.

If HMX_ADD_INCR_LEN_PORT is enabled when an INCR burst is received, DW_axi_hmx takes the burst length from the new input port (hburst_len). The maximum AXI burst length is specified by 2^HMX_BLW (HMX_BLW is the width of the AXI burst length signal) If the hburst_len exceeds the maximum AXI transaction length, DW_axi_hmx breaks it up into multiple AXI transfers. The address required for the same is sourced directly from the AHB address bus, and not generated by hmx.

The new input port hburst_len ensures that DW_axi_hmx does not prefetch any data that AHB does not claime for READs. Similarly for WRITEs, DW_axi_hmx does not start an AXI transaction that cannot be completed due to insufficient data from AHB.

Figure 2-8 shows an AHB INCR transaction with write burst to AXI length conversion. In this example, HMX_BLW has a value 2. After four write transactions are processed, DW_axi_hmx generates another address with an awlen of 0x0 (one beat).

Figure 2-8 AHB INCR Transaction with Write Burst to AXI Length Conversion



In Figure 2-8, the AHB INCR transaction results in five data transfers, which translates to two AXI transactions on the secondary port: the first transaction with awlen of 3 and the second transaction with awlen of 0. The AHB master side needs to ensure that all beats in the INCR transaction are completed otherwise the component transmits unknown data.

Note Ensure that the hburst_len signal is retained constant during an AHB INCR transaction.

2.8 Transaction Blocking

In the AXI protocol, the execution order of read and write transactions is not guaranteed. You can configure the HMX_BLOCK_WRITE parameter to control how the DW_axi_hmx blocks transactions in order to enforce read and write ordering. Examples of writes and reads through the DW_axi_hmx are illustrated in Figure 2-9 and Figure 2-10.

Note Read transactions are always blocking for the following options; that is, no new transaction can be issued until all the read data has been accepted on the AHB side of the DW_axi_hmx.

- HMX_BLOCK_WRITE = 0 – The DW_axi_hmx does nothing to enforce the ordering of transactions that execute on the AXI side. Reads are blocking, but writes are buffered and new transactions can be issued after all the write data has been issued on the AXI side. In this case, a strong read-write ordering cannot be guaranteed. This is the default behavior of DW_axi_hmx.
- HMX_BLOCK_WRITE = 1 – The DW_axi_hmx still buffers write transactions, but does not issue them until there are no outstanding transactions on the AXI side of DW_axi_hmx; no new transactions are issued until the DW_axi_hmx receives the burst response for the write transaction. In this case, strong read-write ordering is enforced at all times through the DW_axi_hmx.
- HMX_BLOCK_WRITE = 2 – You can dynamically control the blocking attributes of both read and write transactions with the bufferable attribute of the AHB transaction (hprot[2]).
  - If a read is received with hprot[2] = 0 (non bufferable), it is not issued by the DW_axi_hmx until there are no outstanding transactions on the AXI side, and no new transactions are issued until the read has completed.
  - If a read is received with hprot[2] = 1 (bufferable), it can be issued before previous transactions have completed; as before, no new transactions are issued until the read has completed.
  - If a write is received with hprot[2] = 0, it is not issued by the DW_axi_hmx until there are no outstanding transactions on the AXI side, and no new transactions are issued until the write has completed (burst response received).
  - If a write is received with hprot[2] = 1, it can be issued before previous transactions have completed, and new transactions can be issued as soon as all the write data for the transaction has been issued on the AXI side.

Figure 2-9 DW_axi_hmx Write Transaction

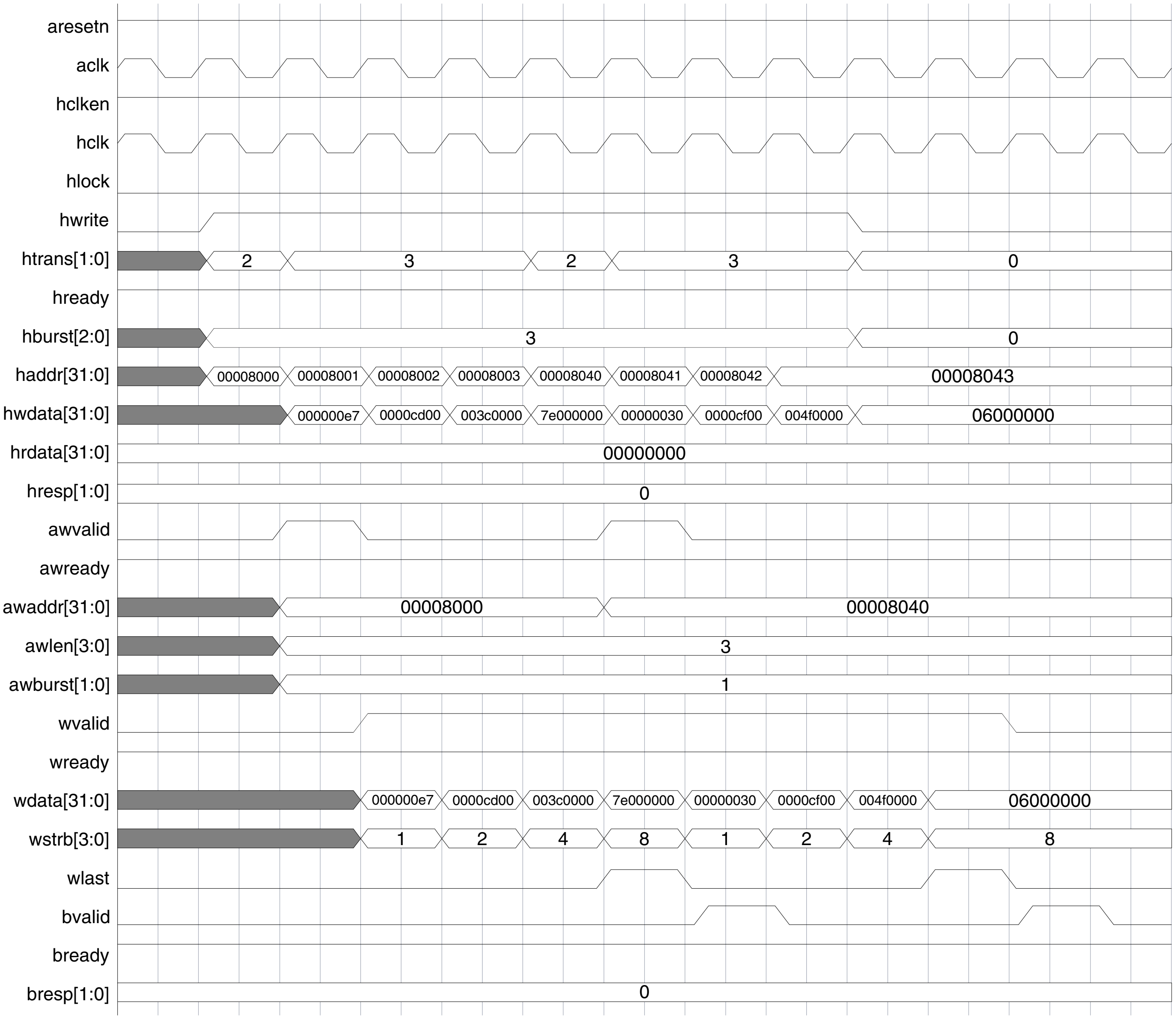
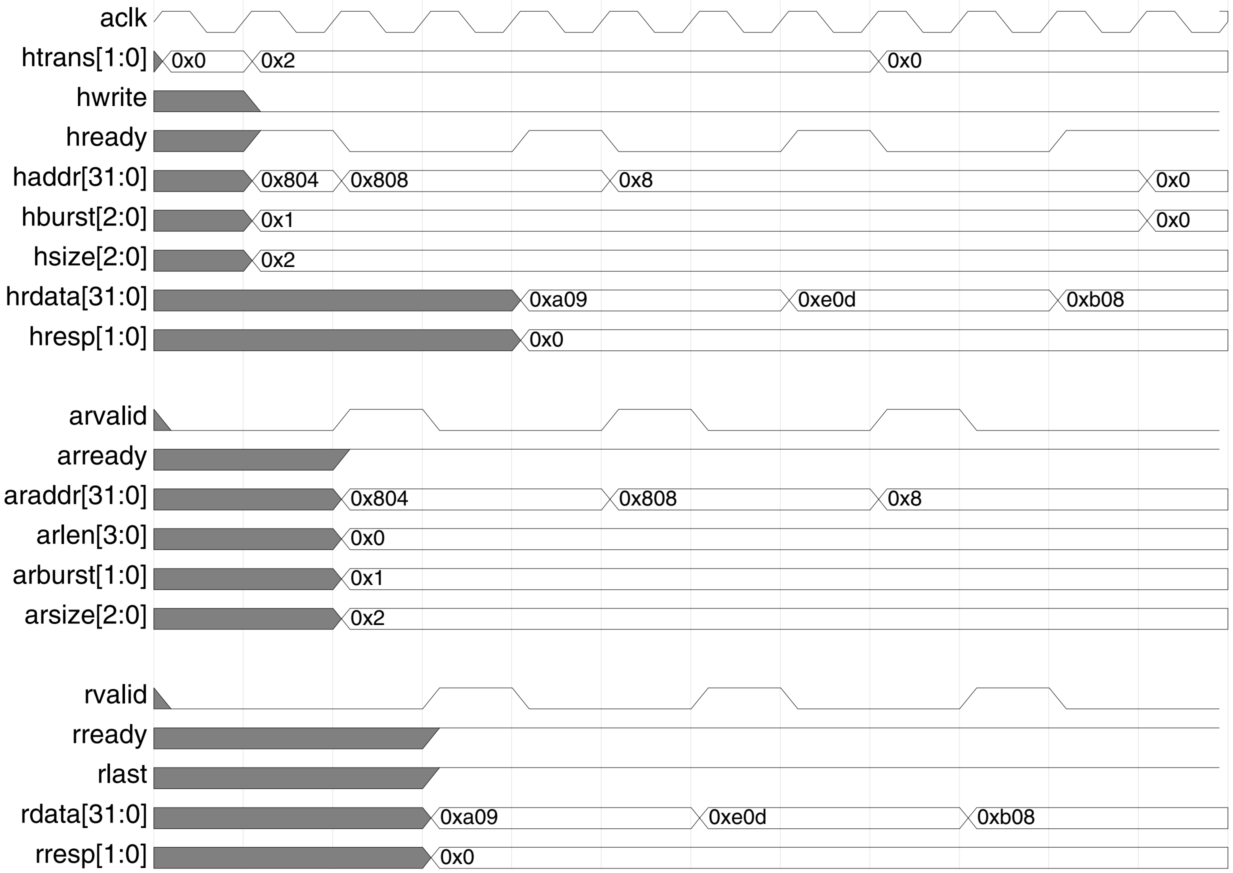


Figure 2-10 DW_axi_hmx Read Transaction



2.9 HPROT Translation

Bits 3:2 of the hprot bus are routed through to bits 1:0 of awcache/arcache. Bits 3:2 of awcache/arcache are set to 0. Bit 1 of hprot is routed through to bit 0 of awprot/arprot. Bit 0 of hprot is inverted and connected to bit 2 of awprot/arprot. Bit 1 of awprot/arprot can be set by the HMX_NONSECURE configuration parameter to a static value; the default is secure.

2.10 Read Data Responses (rresp)

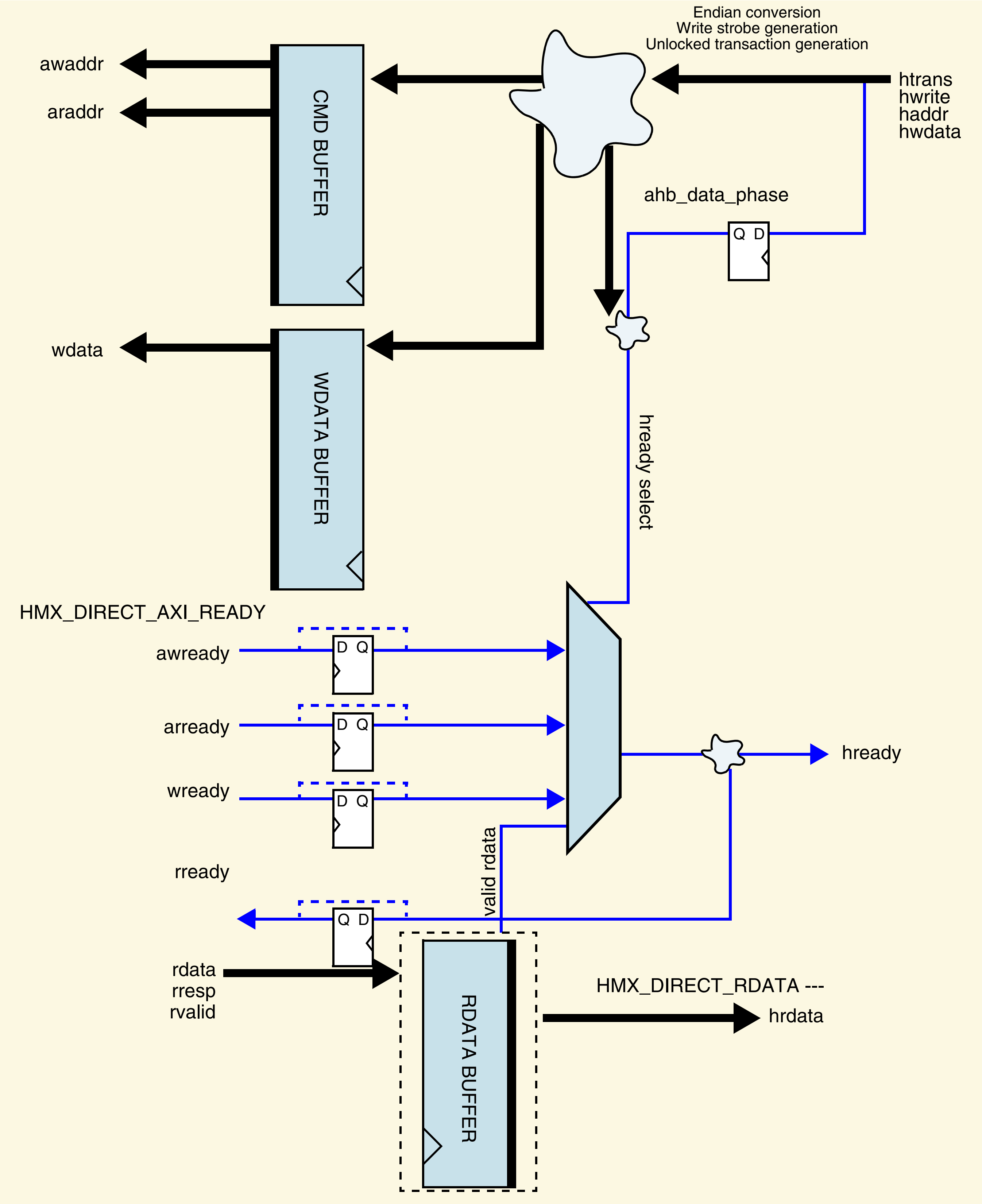
The AXI read data channel response (rresp) types SLVERR and DECERR are translated on AHB hresp to ERROR. The AXI rresp of OKAY is translated to AHB hresp OKAY. The AXI rresp EXOKAY is never used, since DW_axi_hmx never generates exclusive access transactions.

2.11 Combinatorial READY Paths

DW_axi_hmx allows the IP integrator to easily trade-off low latency and high frequency, by including configuration parameters to control whether certain combinatorial paths should be allowed between AXI and AHB (low latency) or should be registered (high frequency).

Figure 2-11 highlights the combinatorial paths controlled by the following configuration parameters.

Figure 2-11 DW_axi_hmx Combinatorial Paths Overview



2.11.1 HMX_DIRECT_AXI_READY

If the CMD buffer is not full, then a new AHB transaction can be accepted by DW_axi_hmx, therefore the hready output can be driven high. If the CMD Buffer is full, there is no room for a new AHB transaction and hready should be low. However, if the CMD buffer is full, but on the current cycle is shifting out one entry to the AXI write address or read address channel, then it is possible to shift in one entry the same cycle. This requires a combinatorial path from the arready and awready inputs to the hready output. By default this flow-through path is allowed, but the configuration parameter HMX_DIRECT_AXI_READY can be disabled to break this combinatorial path. If the arready/awready to hready combinatorial path is the critical path in the subsystem, disabling HMX_DIRECT_AXI_READY increases the operating frequency of the entire subsystem.

Similarly, the WDATA buffer full/not full state, and the AXI write data channel wready signal, can be used combinatorially to determine whether hready should be driven high to accept the next AHB write data beat. This path is also combinatorial/registered depending on the value of HMX_DIRECT_AXI_READY, just like the awready/arready to hready paths.

2.11.2 HMX_DIRECT_AHB_READY

If the RDATA buffer is not full, then a new AXI read data channel transfer can be accepted by DW_axi_hmx, therefore the rready output can be driven high. If the RDATA buffer is full, there is no room for a new read data beat and rready should be low. However, if the RDATA buffer is full, but on the current cycle is shifting out one data beat to the AHB, then it is possible to shift in one AXI read data beat the same cycle. This requires a combinatorial path from the hready signal to the rready output. By default this flow-through path is allowed, but the configuration parameter HMX_DIRECT_AHB_READY can be disabled to break this combinatorial path. If the hready to rready combinatorial path is the critical path in the subsystem, disabling HMX_DIRECT_AHB_READY increases the operating frequency of the entire subsystem.

Note If both HMX_DIRECT_AXI_READY and HMX_DIRECT_AHB_READY are enabled (the default), then combinatorial paths from arready/awready/wready inputs to the rready output exists.

2.11.3 HMX_DIRECT_RDATA

The RDATA Buffer is provided by DW_axi_hmx by default to provide some buffering between the AXI read data channel and the AHB bus. By setting the HMX_DIRECT_RDATA configuration parameter to false, the RDATA Buffer is removed and the AXI read data channel signals are used to combinatorially drive the read data and response to the AHB bus. If HMX_DIRECT_RDATA is false (the default), then HMX_DIRECT_AHB_READY determines if a combinatorial path exists from hready to rready. If HMX_DIRECT_RDATA is true, there is always a combinatorial path from hready to rready. In either case, HMX_DIRECT_AXI_READY determines if there is a combinatorial path from arready/awready/wready to hready.

3
Parameter Descriptions

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

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

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

- Top Level Parameters on page 32
- Performance Parameters on page 35

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
	AHB
AHB Address Width	This parameter specifies the address bus width on the AHB. The value of this parameter must be less than or equal to HMX_AXI_ADDR_WIDTH. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** HMX_ADDR_WIDTH
AHB Data Width	This parameter specifies the data bus width on the AHB interface, which is the same as the data width on the AXI interface. There is no distinction made between read and write channels. **Values:** 8, 16, 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** HMX_DATA_WIDTH
AHB is Big Endian	If set to true, the incoming AHB bus to DW_axi_hmx is big endian. If set to false, the incoming AHB is little endian. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** HMX_ENDIAN_AHB
Include port to specify AHB INCR length ?	This parameter is used to add a port (hburst_len) to the HMX top-level interface to specify the exact length of an AHB INCR burst. Setting this parameter will add design logic to allow INCR bursts to generate multiple AXI transfers to complete the full burst. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** HMX_ADD_INCR_LEN_PORT

Table 3-1 Top Level Parameters (Continued)

Label	Description
	AXI
AXI Interface Type	This parameter is used to select the AXI Interface type as AXI3 or AXI4. By default, DW_axi_hmx supports AXI3 interface. **Values:** ■ AXI3 (0) ■ AXI4 (1) **Default Value:** AXI3 **Enabled:** DWC-AMBA-Fabric-Source License required to enable AXI4 mode **Parameter Name:** HMX_AXI_INTERFACE_TYPE
AXI Address Width	This parameter defines the width of the AXI bus. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** HMX_AXI_ADDR_WIDTH
AXI ID Width	This parameter defines the width of the transaction ID field of the AXI system. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 4 **Enabled:** Always **Parameter Name:** HMX_ID_WIDTH
AXI Burst Length Width	This parameter defines the length of the burst (AWLEN and ARLEN) field of the AXI system. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** Always **Parameter Name:** HMX_BLW
Nonsecure AXI Access	When set to true, all transactions are non-secure. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** HMX_NONSECURE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Use default unlock address	If set to true, DW_axi_hmx uses a preset address for all unlock transactions. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** HMX_USE_UNLOCK_ADDR
Default unlock address	Preset address for all unlock transactions. **Values:** 0x0, ..., 0xffffffffffffffff **Default Value:** 0x0 **Enabled:** Always **Parameter Name:** HMX_UNLOCK_ADDR
Endian Conversion	
Convert Endianness	If set to true, the outgoing AXI bus endianness is the opposite of the incoming AHB bus endianness. For example, if the incoming AHB bus is set to big endian, the outgoing AXI bus is little endian. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** HMX_ENDIAN_CONVERT

3.2 Performance Parameters

Table 3-2 Performance Parameters

Label	Description
Request Data Path Buffer Configuration	
Combinational AXI Readys	If set to true, AXI awready, wready, and arready inputs are combinationally connected to AHB hready. If set to false, the paths from all AXI ready signals to hready are registered. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** HMX_DIRECT_AXI_READY
Command Buffer Depth	This parameter sets the command buffer depth. Higher values allow requests to be buffered, rather than being stalled, if the AXI address channel stalls DW_axi_hmx transactions. Higher values also increase gate count. If HMX_DIRECT_AXI_READY is false, it is recommended to set the depth to 2 in order to avoid performance degradation. **Values:** ■ 1 (1) ■ 2 (2) **Default Value:** 1 **Enabled:** Always **Parameter Name:** HMX_CMD_BUFFER
Write Data Buffer Depth	This parameter sets the write data buffer depth. Higher values allow AHB write data to be buffered, rather than being stalled, if the AXI write data channel stalls DW_axi_hmx transactions. Higher values also increase gate count. If HMX_DIRECT_AXI_READY is false, it is recommended to set the depth to 2 in order to avoid write performance degradation. **Values:** ■ 1 (1) ■ 2 (2) **Default Value:** 1 **Enabled:** Always **Parameter Name:** HMX_WDATA_BUFFER

Table 3-2 Performance Parameters (Continued)

Label	Description
	Response Data Path Buffer Configuration
Combinational HREADY	If set to true, the hready input is combinationally connected to the rready output. If set to false, the hready input is registered. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** HMX_DIRECT_RDATA==0 **Parameter Name:** HMX_DIRECT_AHB_READY
Remove Read Data Buffer	If set to true, there is no RDATA buffer. The read response channel is directly connected to AHB. If set to false, the RDATA buffer is instantiated and all read responses are buffered. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** HMX_DIRECT_RDATA
Read Data Buffer Depth	Depth of AXI read data and read response buffer. Higher values allow AXI responses to be buffered, rather than stalled, if the AHB stalls DW_axi_hmx transactions. Higher values also increase gate count. **Values:** ■ 1 (1) ■ 2 (2) **Default Value:** 1 **Enabled:** HMX_DIRECT_RDATA==0 **Parameter Name:** HMX_RDATA_BUFFER

Table 3-2 Performance Parameters (Continued)

Label	Description
Protocol	
Transaction Blocking	Following are the different values for this parameter: ■ Disabled - Outstanding AXI write transactions need not be completed (write response received from the AXI) before subsequent AHB transactions can be driven onto the AXI address channels. ■ Block Writes - Outstanding AXI write transactions MUST be completed (write response received from the AXI) before subsequent AHB transactions can be driven onto the AXI address channels. ■ Dynamically Block Reads and Writes - The behavior depends on the level of hprot[2]. When hprot[2]==0, the AHB transaction (read or write) is stalled until all outstanding AXI transactions have completed. Subsequent AHB transactions are stalled until this transaction completes. When hprot[2]==1, outstanding AXI write transactions need not be completed before this AHB transaction can be driven onto the AXI address channels. **Values:** ■ Disabled (0) ■ Block Writes (1) ■ Dynamically Block Reads & Writes (2) **Default Value:** Disabled **Enabled:** Always **Parameter Name:** HMX_BLOCK_WRITE

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:

- Clock and Resets on page 41
- AHB Interface Signals on page 42
- AXI Write Address Channel on page 45
- AXI Write Data Channel on page 48
- AXI Write Response Channel on page 50
- AXI Read Address Channel on page 52
- AXI Read Data Channel on page 55
- Error Signals on page 57
- Miscellaneous AXI Signals on page 58

4.1 Clock and Resets Signals

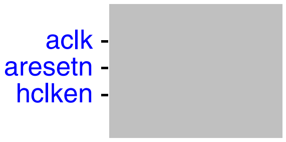


Table 4-1 Clock and Resets Signals

Port Name	I/O	Description
aclk	I	AXI clock signal. All AXI interface input signals are sampled on the rising edge of aclk. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aresetn	I	AXI reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of aclk. DW_axi_hmx 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
hclken	I	Enables aclk when high. This signal can be used to slow down the clock on the AHB side. **Exists:** Always **Synchronous To:** aclk **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.2 AHB Interface Signals

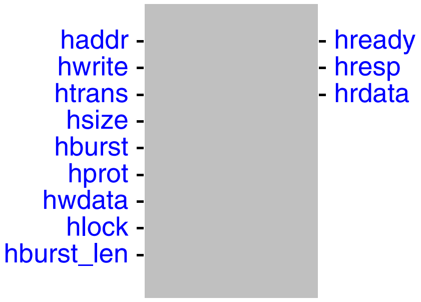


Table 4-2 AHB Interface Signals

Port Name	I/O	Description
hready	O	Ready response from selected slave. Indicates that the current transfer is complete. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hresp[1:0]	O	Transfer response. Indicates response type from slave. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hrdata[(HMX_DATA_WIDTH-1):0]	O	Transfer read data. Used to transfer data from bus slaves to bus master during read operations. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
haddr[(HMX_ADDR_WIDTH-1):0]	I	AHB address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 AHB Interface Signals (Continued)

Port Name	I/O	Description
hwrite	I	Transfer write control. When HIGH, this signal indicates write transfer. When LOW, this signal indicates read transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
htrans[1:0]	I	Transfer control. Indicates type of transfer being performed. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hsize[2:0]	I	Transfer size. Indicates size of transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hburst[2:0]	I	Transfer type. Indicates if transfer constitutes part of a burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hprot[3:0]	I	Protection control. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwdata[(HMX_DATA_WIDTH-1):0]	I	Transfer write data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 AHB Interface Signals (Continued)

Port Name	I/O	Description
hlock	I	Bus lock. Asserted by bus master to indicate need for locked transaction. hlock is used only if HMX_AXI_INTERFACE_TYPE is selected to be of type AXI 3 because hlock remains unused and the master must ensure that hlock is not driven high, to avoid unpredictable behavior. hlock is included for interface consistency in AXI 4 mode. For information about setting the HMX_AXI_INTERFACE_TYPE parameter, see the "Parameters" chapter. **Exists:** Always **Synchronous To:** (HMX_AXI_INTERFACE_TYPE == 0) ? "aclk" :"None" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hburst_len[(HMX_HBURST_LEN_W-1):0]	I	This signal indicates the length of AHB INCR Burst. This AHB non-protocol signal provides length of unspecified INCR bursts in advance, thereby allowing INCR bursts up to HMX_BLW beats to be sent in one AXI transaction. This bus must remain constant for the duration of the AHB burst. **Exists:** (HMX_ADD_INCR_LEN_PORT==1) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 AXI Write Address Channel Signals

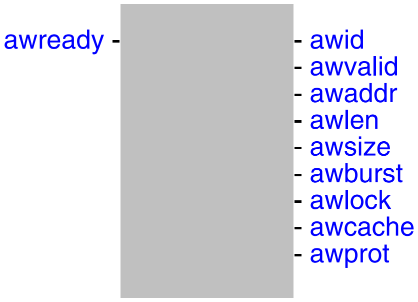


Table 4-3 AXI Write Address Channel Signals

Port Name	I/O	Description
awid[(HMX_ID_WIDTH-1):0]	O	AXI write address identification. Gives identification tag for write address signals. The value of HMX_ID_WIDTH is hardwired to 0. For information about setting HMX_ID_WIDTH, see the "Parameters" chapter in the DW_axi_hmx Databook. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awvalid	O	AXI write address valid. Indicates that a valid write address and control information are available. Address and control information remain stable until awready signal is high. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr[(HMX_AXI_ADDR_WIDTH-1):0]	O	AXI write address. Specifies address of first transfer in write burst transaction. Associated control signals are used to determine addresses of remaining transfers in the burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

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

Port Name	I/O	Description
awlen[(HMX_BLW-1):0]	O	AXI Write Burst length. Specifies exact number of transfers in burst; determines number of data transfers associated with the address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize[2:0]	O	AXI write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exact byte lanes to update. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awburst[1:0]	O	AXI Write Burst. Combined with size information, this signal shows how address for each transfer with burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock[(1-HMX_AXI_INTERFACE_TYPE):0]	O	AXI write lock. Provides additional information about characteristics of the transfer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN
awcache[3:0]	O	AXI write cache. Indicates bufferable, cacheable, write-through, and write-back attributes of transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

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

Port Name	I/O	Description
awprot[2:0]	O	AXI write protection. Indicates normal, privileged, or secure protection level of transaction and whether the transaction is data access or instruction access. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awready	I	AXI write address ready. Indicates that slave is ready to accept address and associated control signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 AXI Write Data Channel Signals

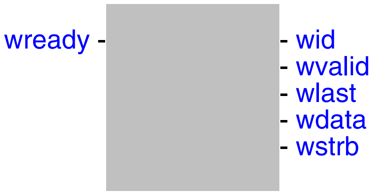


Table 4-4 AXI Write Data Channel Signals

Port Name	I/O	Description
wid[(HMX_ID_WIDTH-1):0]	O	AMBA AXI3 write identification tag of write data transfer. **Exists:** HMX_AXI_INTERFACE_TYPE==0 **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wvalid	O	AXI write valid. This signal indicates that valid write data and strobes are available. ■ 0: Write data and strobes not available ■ 1: Write data and strobes available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wlast	O	AXI write last. Indicates the last transfer in a write burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdata[(HMX_DATA_WIDTH-1):0]	O	AXI write data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

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

Port Name	I/O	Description
wstrb[((HMX_DATA_WIDTH/8)-1):0]	O	AXI write strobes. Indicates byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. The wstrb signal is associated with wdata[(8 x n)+7:(8 x n)]. Only the strobes for the configured data width are generated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wready	I	AXI write ready. Indicates that the slave can accept the write data. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 AXI Write Response Channel Signals

bid
bvalid
bresp
bready

Table 4-5 AXI Write Response Channel Signals

Port Name	I/O	Description
bready	O	AXI response ready. Indicates that the 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
bid[(HMX_ID_WIDTH-1):0]	I	AXI write response identification tag. Value must match awid value of write transaction to which slave responds. This signal is driven from the RESP buffer. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bvalid	I	AXI write response valid. Indicates that valid write response is available. This signal is driven from the RESP buffer status output. ■ 0: Write response not available ■ 1: Write response available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-5 AXI Write Response Channel Signals (Continued)

Port Name	I/O	Description
bresp[1:0]	I	AXI write response. Indicates status of write transaction; responses supported by DW_axi_hmx are OKAY and SLVERR. This signal is driven from the RESP buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.6 AXI Read Address Channel Signals

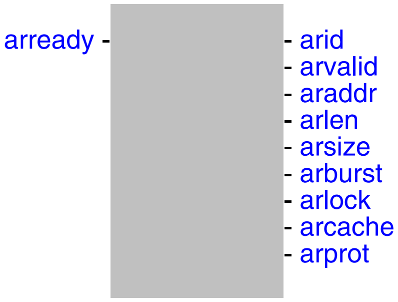


Table 4-6 AXI Read Address Channel Signals

Port Name	I/O	Description
arid[(HMX_ID_WIDTH-1):0]	O	AXI read address identification. Provides identification tag for read address signals. The value of HMX_ID_WIDTH is hardwired to 0; for information about setting HMX_ID_WIDTH, see the "Parameters" chapter in the DW_axi_hmx Databook. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arvalid	O	AXI read address valid. When high, indicates that the 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
araddr[(HMX_AXI_ADDR_WIDTH-1):0]	O	AXI read address. Provides initial address of read burst transaction. Only start address of burst is provided, and control signals issued alongside address show how the address is calculated for remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 **AXI Read Address Channel Signals (Continued)**

Port Name	I/O	Description
arlen[(HMX_BLW-1):0]	O	Burst length. The burst length provides the exact number of transfers in a burst and determines the number of data transfers associated with the address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize[2:0]	O	AXI 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
arburst[1:0]	O	AXI read Burst. Combined with size information, this signal shows how address for each transfer with burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock[(1-HMX_AXI_INTERFACE_TYPE):0]	O	AXI read lock. Provides additional information about the atomic characteristics of the transfer. Encoded value indicates whether the transfer is normal, exclusive, or locked access. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache[3:0]	O	AXI read cache. Indicates bufferable, cacheable, read-through, and read-back attributes of transaction. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

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

Port Name	I/O	Description
arprot[2:0]	O	AXI read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arready	I	AXI 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:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.7 AXI Read Data Channel Signals

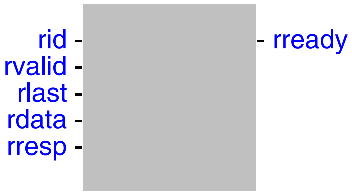


Table 4-7 AXI Read Data Channel Signals

Port Name	I/O	Description
rready	O	AXI read ready. Indicates that the 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
rid[(HMX_ID_WIDTH-1):0]	I	AXI Read identification tag. Value is generated by the slave and must match arid value of read transaction to which it responds. This signal is driven from the RDATA buffer. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rvalid	I	AXI read valid. Indicates that required read data is available and read transfer can complete. This signal is driven from the RDATA buffer. ■ 0: Read data not available ■ 1: Read data available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-7 AXI Read Data Channel Signals (Continued)

Port Name	I/O	Description
rlast	I	AXI read last. Indicates last transfer in read burst. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rdata[(HMX_DATA_WIDTH-1):0]	I	AXI read data. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp[1:0]	I	AXI read response. Indicates status of read transaction; responses supported by DW_axi_hmx are OKAY and SLVERR. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.8 Error Signals

xwerrclr
xwdecerr
xwslverr

Table 4-8 Error Signals

Port Name	I/O	Description
xwdecerr	O	AXI flag for decode write error response. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
xwslverr	O	AXI flag for slave error response **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
xwerrclr	I	Clears xwdecerr and xwslverr flags. **Exists:** Always **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.9 Miscellaneous AXI Signals

Table 4-9 Miscellaneous AXI Signals

Port Name	I/O	Description
nopx	O	AXI no pending transaction. Indicates that there are no pending transactions and that the AXI clock can be shut off. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

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
HMX_HBURST_LEN_W	10

6 Verification

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

Note The DW_axi_hmx 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:

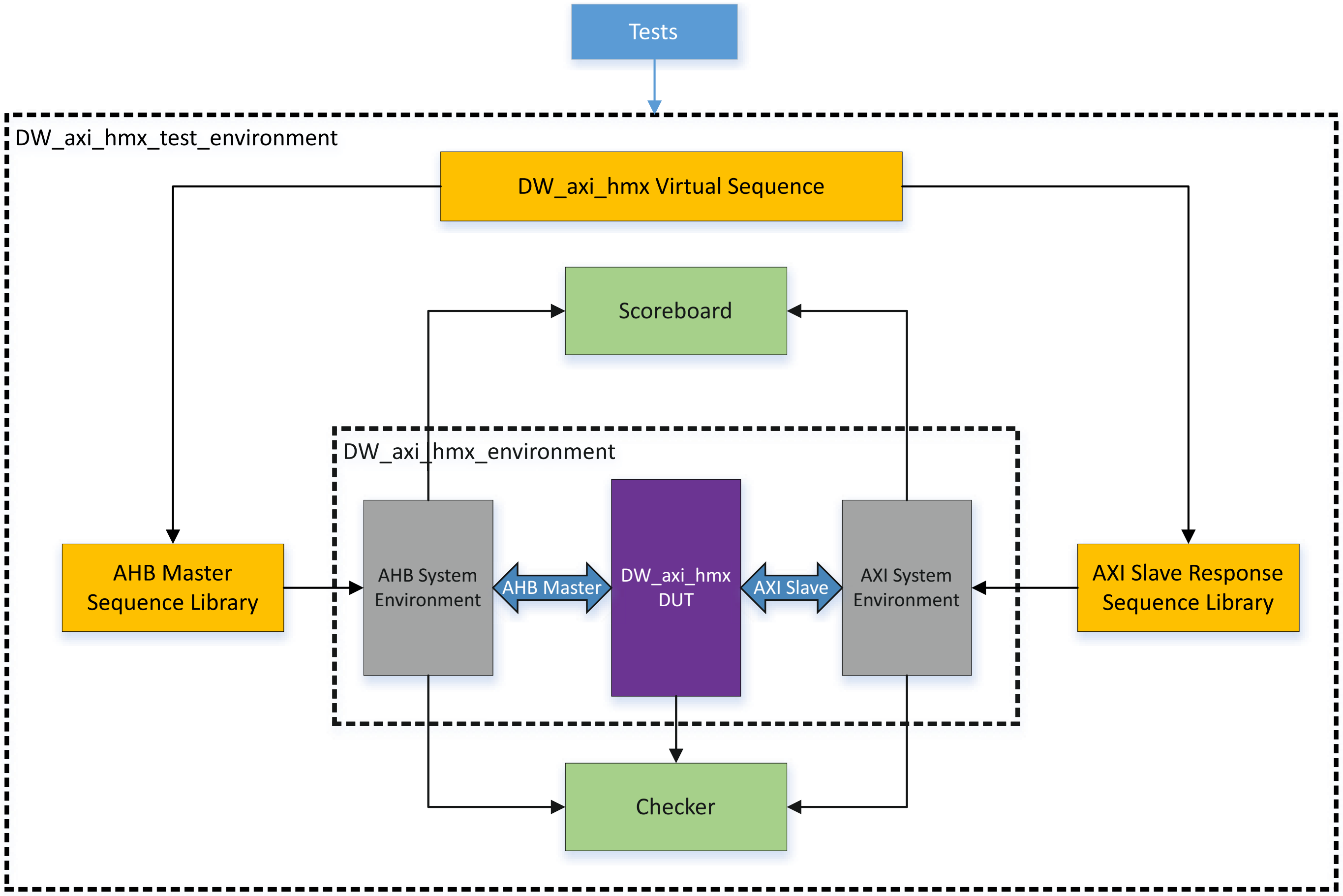
- "Verification Environment" on page 61
- "Testbench Directories and Files" on page 62
- "Packaged Testcases" on page 63

6.1 Verification Environment

DW_axi_hmx 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_hmx testbench:

Figure 6-1 DW_axi_hmx 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
        - AHB VIP for the interface with AHB Data transfer interface
        - AHB Master VIP model is used to generate random sequences
        - AXI Slave VIP model is used to generate random response sequences

6.2 Testbench Directories and Files

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

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

Table 6-1 DW_axi_hmx Testbench Directory Structure

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

6.3 Packaged Testcases

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

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

Table 6-2 DW_axi_hmx Test Description

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

7
Integration Considerations

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

7.1 Hardware Considerations

7.1.1 Usage Requirements

When connecting the subsystem, the hgrant signal at the AHB master driving DW_axi_hmx AHB master port must be tied high. The hbusreq signal from the AHB master is not connected to DW_axi_hmx and is not necessary since the hgrant is always given to this master.

7.1.2 Configuration Recommendations

To facilitate timing closure, you can configure a direct-ready feed-through in order to have either registered or unregistered ready signals from the AHB to the AXI, and vice versa. If the DIRECT_AXI_READY configuration parameter is set to True, the AXI ready signals arready, awrready, and wready are combinatorially connected to the AHB signal hready. If the DIRECT_AHB_READY configuration parameter is set to True, hready is combinatorially connected to rready. The DIRECT_AHB_READY configuration parameter is only meaningful when used in conjunction with a read data buffer. Without a read data buffer, hready is always directly connected to rready. The signal bready is not affected by any timing mode, because all write transactions are buffered and the write responses are processed independently of hready. Finally, HMX_DIRECT_RDATA instantiates the read data buffer if set to true.

7.2 Performance

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

7.2.1 Power Consumption, Frequency, Area, and DFT Coverage

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

Table 7-1 Synthesis Results for DW_axi_hmx

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtCov (%)
			Static	Dynamic	StuckAtTest	Transition	
Default Configuration (cconfig_default)	400 MHz	3282	3 nW	0.135 mW	99.67	99.59	99.7
Typical Configuration - 1 HMX_ADDR_WIDTH = 32 HMX_AXI_ADDR_WIDTH = 32 HMX_DATA_WIDTH = 8 HMX_ID_WIDTH = 1 HMX_DIRECT_AXI_READY = 1 HMX_DIRECT_AHB_READY = 1 HMX_DIRECT_RDATA = 1 HMX_CMD_BUFFER = 1 HMX_WDATA_BUFFER = 1 HMX_RDATA_BUFFER = 1 HMX_ENDIAN_AHB = 0 HMX_ENDIAN_CONVERT = 1 HMX_NONSECURE = 0 HMX_BLOCK_WRITE = 0 HMX_USE_UNLOCK_ADDR = 0	400 MHz	2498	2 nW	0.100 mW	99.56	99.45	99.7

Table 7-1 Synthesis Results for DW_axi_hmx

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtCov (%)
			Static	Dynamic	StuckAtTest	Transition	
Typical Configuration - 2 HMX_ADDR_WIDTH = 64 HMX_AXI_ADDR_WIDTH = 64 HMX_ADD_INCR_LEN_PORT = 1 HMX_BLOCK_WRITE = 1 HMX_CMD_BUFFER = 2 HMX_DATA_WIDTH = 512 HMX_DIRECT_AHB_READY = 0 HMX_DIRECT_AXI_READY = 0 HMX_DIRECT_RDATA = 0 HMX_ENDIAN_AHB = 1 HMX_ENDIAN_CONVERT = 0 HMX_ID_WIDTH = 12 HMX_NONSECURE = 0 HMX_RDATA_BUFFER = 2 HMX_USE_UNLOCK_ADDR = 1 HMX_WDATA_BUFFER = 2	400 MHz	38912	30 nW	1.570 mW	99.97	99.96	99.9

Table 7-1 Synthesis Results for DW_axi_hmx

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax Coverage (%)		SpyGlass StuckAtCov (%)
			Static	Dynamic	StuckAtTest	Transition	
Typical Configuration - 3 HMX_AXI_INTERFACE_TYPE = 1 HMX_ADDR_WIDTH = 64 HMX_ADD_INCR_LEN_PORT = 0 HMX_AXI_ADDR_WIDTH = 64 HMX_BLOCK_WRITE = 0 HMX_BLW = 8 HMX_CMD_BUFFER = 1 HMX_DATA_WIDTH = 256 HMX_DIRECT_AHB_READY = 0 HMX_DIRECT_AXI_READY = 1 HMX_DIRECT_RDATA = 0 HMX_ENDIAN_AHB = 0 HMX_ENDIAN_CONVERT = 0 HMX_ID_WIDTH = 1 HMX_MST_CONTINUE_ON_ERR = 0 HMX_NONSECURE = 1 HMX_RDATA_BUFFER = 1 HMX_UNLOCK_ADDR = 5678 HMX_USE_UNLOCK_ADDR = 1 HMX_WDATA_BUFFER = 1 SIM_HCLK_PERIOD = 10	400 MHz	9118	7 nW	0.381 mW	99.88	99.84	99.8

7.2.2 Latency

The latency from AHB-to-AXI signaling is one aclk cycle.

7.2.3 Throughput

There are performance degradation issues when ready signals are registered and the buffer depth is set to 1. In this case, data cannot stream through the 1-deep buffers, and every access requires two cycles. This holds true for every single beat of a burst. To avoid this penalty, the buffer depth has to be set to 2. To maintain streaming of one access every cycle, the receiving interface must maintain the same access rate, as well.

7.3 Verification Integration

For information about the verification environment, see "Verification" on page 61,

7.3.1 Location of Pre-instantiated Instance

workspace/export/DW_axi_hmx_inst.v

7.3.2 Deviations From Standard Protocols

Used point-to-point connect to an AHB master. There is no AHB bus fabric necessary.

8
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_hmx.

8.1 BCM Library Components

Table 8-1 describes the list of BCM library components used in DW_axi_hmx.

Table 8-1 List of BCM Library Components used in the Design

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_x2x_bcm06	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_axi_x2x_bcm57	Synchronous Write-Port, Asynchronous. Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff
DW_axi_x2x_bcm65	Synchronous (Single Clock) FIFO with Static Flags	DW_fifo_s1_sf

A Glossary

active command queue

Command queue from which a model is currently taking commands; see also command queue.

application design

Overall chip-level design into which a subsystem or subsystems are integrated.

BFM

Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.

big-endian

Data format in which most significant byte comes first; normal order of bytes in a word.

blocked command stream

A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.

blocking command

A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.

command channel

Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.

command stream

The communication channel between the testbench and the model.

component

A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.

configuration

The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.

configuration intent

Range of values allowed for each parameter associated with a reusable core.

cycle command

A command that executes and causes HDL simulation time to advance.

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

peripheral

Generally refers to a small core that has a bus connection, specifically an APB interface.

RTL

Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.

SDRAM

Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.

SDRAM controller

A memory controller with specific connections for SDRAMs.

slave

Device or model that is controlled by and responds to a master.

SoC

System on a chip.

soft IP

Any implementation IP that is configurable. Generally referred to as synthesizable IP.

sparse data

Data bus data which is individually byte-enabled. The AXI bus allows sparse data transfers using the WSTRB signal, each byte lane individually enabled. The AHB bus does not allow sparse data transfers.

static controller

Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.

synthesis intent

Attributes that a core developer applies to a top-level design, ports, and core.

synthesizable IP

A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.

technology-independent

Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.

Testsuite Regression Environment (TRE)

A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.

VIP

Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.

wrap, wrapper

Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.

zero-cycle command

A command that executes without HDL simulation time advancing.

Index