SYNOPSYS®

DesignWare® DW_ahb_h2h

Databook

DW_ahb_h2h – Product Code

1.11a
December 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 table shows the revision history for the databook from release to release. This is being tracked from version 1.04b onward.

Version	Date	Description
1.11a	December 2020	**Added:** ■ “AMBA 5 AHB Support” on page 28 ■ “Comparison of Endian Schemes” on page 91 ■ “Timing Exceptions” on page 83 **Updated:** ■ Version changed for 2020.12a release ■ “Features” on page 15 ■ “Performance” on page 84 ■ “Verification” on page 77 **Renamed:** ■ “Synchronizer Methods” to “Basic Core Module (BCM) Library” on page 87 ■ “Clock Adaption” to “Clocks and Resets” on page 20 **Removed:** ■ Index chapter
1.10a	July 2018	**Updated:** ■ Version changed for 2018.07a release ■ “Performance” on page 84 ■ “Parameter Descriptions” on page 49, “Signal Descriptions” on page 59,and “Internal Parameter Descriptions” on page 89 are auto-extracted with change bars from the RTL ■ Added MID Sideband signals: mmid, and smid **Removed:** ■ Chapter 2, “Building and Verifying a Component or Subsystem” and added the contents in the newly created user guide.

Version	Date	Description
1.09a	October 2016	■ Version number change for 2016.10a release ■ "Parameter Descriptions" on page 49 auto-extracted from the RTL ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Replaced Figure 2-2 and Figure 2-3 to remove references to Leda ■ Moved Internal Parameter Descriptions to Appendix ■ Added "Running VCS XPROP Analyzer"
1.08a	June 2015	■ Added "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added "Running Spyglass on Generated Code with coreAssembler" ■ Updated "Deadlock" on page 26 to provide clarity on the deadlock resolving mechanism. ■ "Signal Descriptions" on page 59 auto-extracted from the RTL ■ Added "Internal Parameter Descriptions" on page 89 ■ Added Appendix A, "Basic Core Module (BCM) Library"
1.07a	June 2014	■ Version change for 2014.06a release ■ Added "Performance" section in the "Integration Considerations" chapter
1.07d	May 2013	Corrected a minor typo in Table 5-1. Updated the template
1.06c	October 2012	Added the product code on the cover and in Table 1-1
1.06c	October 2011	Version change for 2011.10a release.
1.06b	June 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced "Related Documents" section in Preface
1.06b	October 2010	Version change for 2010.10a release
1.06a	September 2010	Corrected names of include files and vcs command used for simulation
1.05a	November 2009	Updated databook look-n-feel for consistency with other IIP/VIP/PHY databooks
1.05a	May 2009	Removed references to QuickStarts, as they are no longer supported
1.05a	October 2008	■ Updated description of shsplit connections ■ Version change for 2008.10a release
1.04c	June 2008	Version change for 2008.06a release
1.04b	June 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DesignWare® AHB-to-AHB Bridge component to the Advanced High-Performance Bus (AHB). This component is referred to as DW_ahb_h2h throughout the remainder of this databook and conforms to the *AMBA Specification, Revision 2.0* from Arm®.

The information in this databook includes a functional description, signal and parameter descriptions, and a memory map. 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.

Databook 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_ahb_h2h.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_ahb_h2h.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_ahb_h2h signals.
- Chapter 5, "Verification" provides information on verifying the configured DW_ahb_h2h.
- Chapter 6, "Integration Considerations" includes information you need to integrate the configured DW_ahb_h2h into your design.
- Appendix A, "Basic Core Module (BCM) Library" documents the synchronizer methods (blocks of synchronizer functionality), and the list of BCM library components used in DW_ahb_e2h.
- Appendix B, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.
- Appendix C, "Comparison of Endian Schemes" compares different endian schemes that are used in DW_ahb_h2h.
- Appendix D, "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 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, see the Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI (Documentation Overview).

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:** AMBA

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

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

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

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

Product Code

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

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

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multi-layer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI
DW_axi_a2x	Configurable bridge between AXI and AHB components or AXI and AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI fabric
DW_axi_hmx	Configurable high performance interface from and AHB master to an AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI subsystems
DW_axi_x2h	Bridge from AMBA 3 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems

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

Component Name	Description
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or buses

1 Product Overview

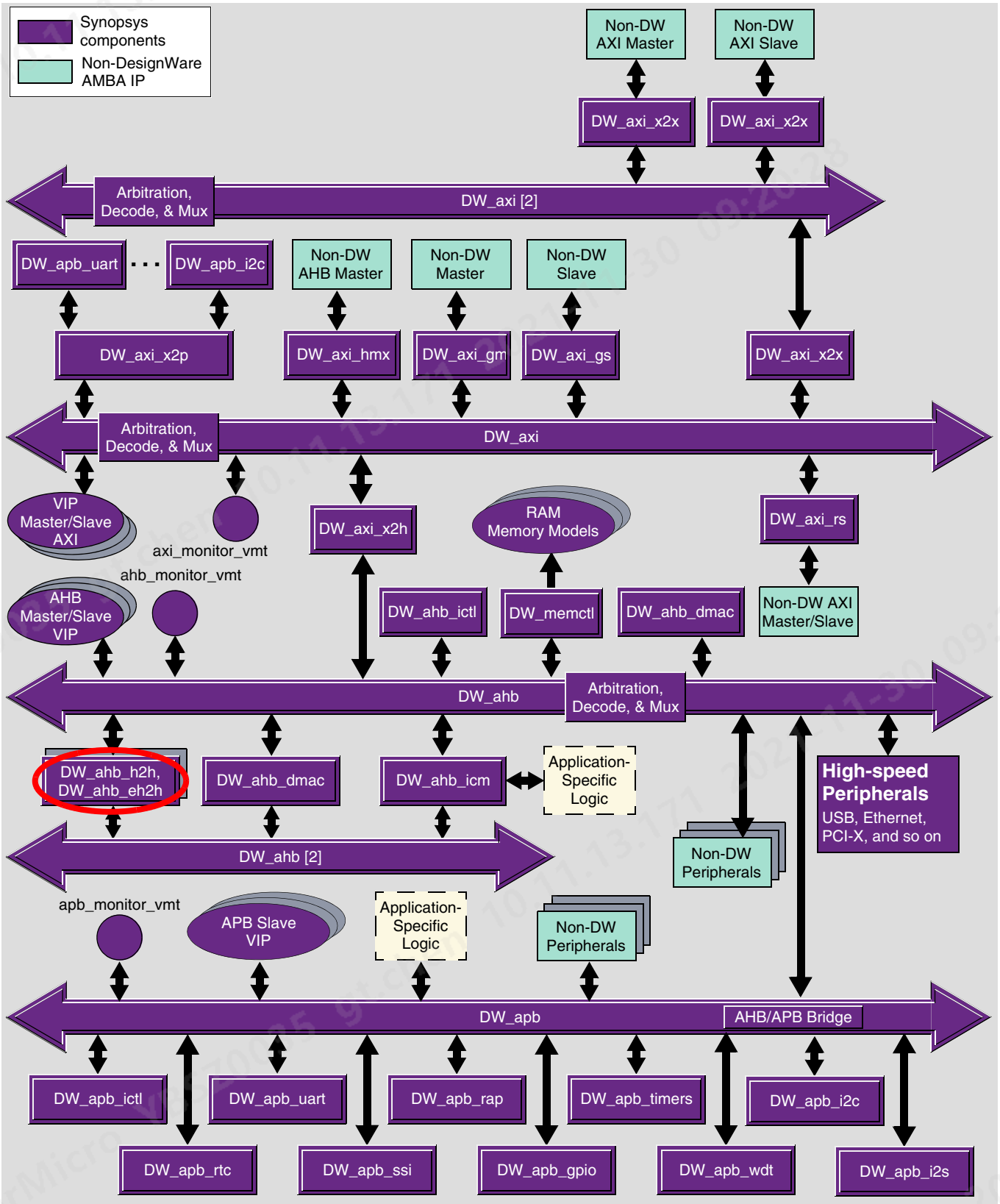
The DW_ahb_h2h is an AHB-to-AHB bridge with a register-based architecture, designed to perform retiming of the datapath to a different clock domain with minimal gate count and low bus performance. This component is part of the DesignWare Synthesizable Components for AMBA 2.

1.1 DesignWare 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 version 3.0-compliant 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. To access the product page and documentation for AMBA components, see the DesignWare IP Solutions for AMBA Interconnect page (SolvNetPlus ID required)

Figure 1-1 Example of DW_ahb_h2h in a Complete System



You can connect, configure, synthesize, and verify the DW_ahb_h2h 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_ahb_h2h component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

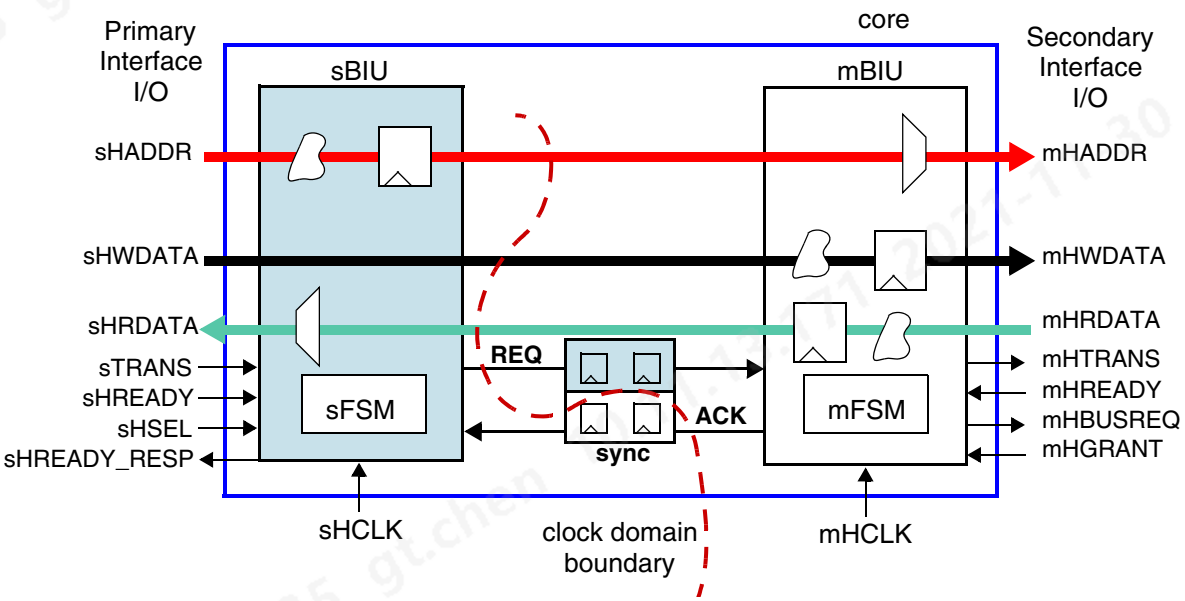
DW_ahb_h2h is an AHB-to-AHB bridge. It is attached as a slave to a primary AHB subsystem and as a master to a secondary AHB subsystem. The DW_ahb_h2h conforms to the *AMBA Specification, Revision 2.0* from Arm®.

1.2.1 DW_ahb_h2h Block Diagram

As illustrated in Figure 1-2, the internal architecture is partitioned into a slave Bus Interface Unit (sBIU) and a master Bus Interface Unit (mBIU). Each BIU is a single-clock design consisting of a finite state machine block (sFSM and mFSM, respectively), data path registers, and data path logic (distinguished as input logic and output multiplexers). The two BIUs interact through handshaking signals produced by the FSMs. Depending on the configured clocking mode, zero, one, or two flip-flops are used to synchronize the handshaking signals (sync block in Figure 1-2) from one clock domain to the other.

Figure 1-2 illustrates the address, read/write data, and slave/master controls of the DW_ahb_h2h.

Figure 1-2 DW_ahb_h2h Block Diagram



1.3 Features

The DW_ahb_h2h is mainly targeted for applications with asynchronous clocks and low gate-count overhead requirements. Synchronous clocks are also inherently supported. You must specify a configuration parameter to reduce internal retiming stages when asynchronous operation mode is not needed. The DW_ahb_h2h has the following features:

1.3.1 System Level

- Configurable asynchronous or synchronous clocks – any clock ratio
- Four clocking modes for synchronous clock configurations – two with and two without clock enables
- Low gate-count implementation (minimum configuration below 2K gates)
- Suboptimal throughput performance (non-buffered architecture)
- High clock-speed operations (fully registered outputs, operating frequency more than 300 MHz)
- Configuration of DesignWare AHB Lite system

1.3.2 AHB Master Interface

- Configurable AHB address width (32 or 64 bits)
- Configurable AHB data width (32, 64, 128, or 256 bits)
- Configurable endianness
- HLOCK generation
- HBUSREQ generation
- HTRANS: generation of IDLE or NSEQ bus cycles (no BUSY and no SEQ)
- Non-pipelined transfers: address phase followed by IDLE cycles until data phase completes
- HBURST: fixed to SINGLE
- All other AHB control signals forwarded unchanged
- AHB Lite configuration to remove redundant logic
- Deadlock detection (optional sensitivity to an external time tick signal)

1.3.3 AHB Slave Interface

- Deadlock protection: SPLIT response generation after deadlock detection at the master interface
- Bus held off (HREADY low) until the secondary transfer data phase completes and is acknowledged back from the master interface
- SPLIT response (from secondary) forwarded back to primary as RETRY

The RETRY response is intended to be used by slaves that are accessed by only one bus master. If you want to access the secondary with multiple masters, you must use a bridge that does not have a RETRY response; for example, the DW_ahb_eh2h.

- Component ID code retrievable from read data bus
- Support for locked transfers (any HTRANS) through HMASTLOCK
- IDLE and BUSY non-locked cycles ignored

1.3.4 Applications of the DW_ahb_h2h

The scope of the DW_ahb_h2h component is to provide flexibility to the system architecture. The bridge addresses the following system-level topics:

- Composition of pre-designed AHB subsystems with:
  - Multiple AHB clock domains
  - Different AHB bus widths, endianness
- Partitioning of an AHB complex system according to system-level and architectural considerations. For example:
  - High-performance, primary system off-loading (grouping of low-performance AHB slaves under a separate lower-performance, secondary bus)
  - Primary or secondary subsystems with bidirectional traffic requirements
  - Timing isolation for timing-critical or asynchronous systems
  - Assembly of loosely coupled systems with low interference requirements
  - Primary system expansion

1.3.5 AMBA 5 AHB Support

- Extended memory types
- Secure Transfers
- Endian Conversion support
- User Signals support for address and data channels

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

1.4 Standards Compliance

The DW_ahb_h2h component conforms to the *AMBA Specification, Revision 2.0* from Arm®. Readers are assumed to be familiar with this specification.

1.5 Verification Environment Overview

The DW_ahb_h2h 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 77 section discusses the specific procedures for verifying the DW_ahb_h2h.

1.6 Licenses

Before you begin using the DW_ahb_h2h, you must have a valid license. For more information, see "Licenses" in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_ahb_h2h 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*.

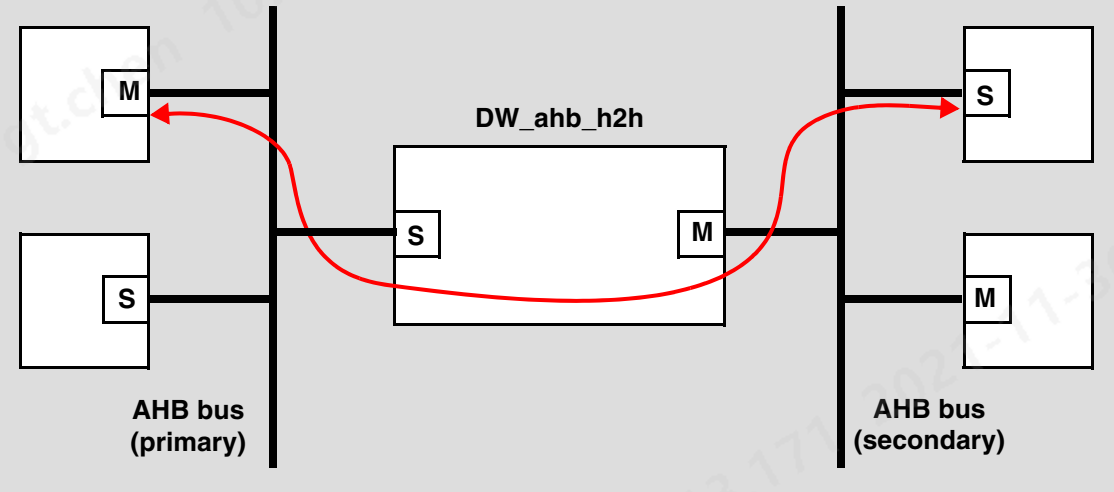
For more information about configuring, synthesizing, and verifying just your DW_ahb_h2h component, see "Overview of the coreConsultant Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_ahb_h2h component within a DesignWare subsystem using coreAssembler, see "Overview of the coreConsultant Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

This chapter describes the DW_ahb_h2h, which is an AHB component attached as a slave to a first AHB subsystem (primary AHB) and as a master to a second AHB subsystem (secondary AHB). The function of DW_ahb_h2h is to establish a communication link between the two subsystems, allowing for data exchange between a primary master and a secondary slave, as illustrated in Figure 2-1.

Figure 2-1 System Overview



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 requesting/granting protocols to the arbiter and split/retry responses from the slaves; all slaves are made non-split capable
- Does not include arbiter as the signals associated with the component are not used: hbusreq and hgrant
- Does not include write data, address, or control multiplexers
- Pause mode is not enabled
- Default master number is changed to 1
- Number of masters is changed to 1

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

2.1 Clocks and Resets

The DW_ahb_h2h supports asynchronous and synchronous clocks. The same core architecture is used in both cases, the only difference being the synchronizer configuration (sync block in Figure 1-2 on page 15).

The DW_ahb_h2h architecture is based on a self-timed handshaking mechanism between the slave and the master bus interface units (BIUs), allowing the component to be clock-ratio independent. The design guarantees correct operation unless a synchronization failure[1] occurs in the synchronizers. A two-phase signaling scheme is used to reduce the average rate of change of signals being synchronized (REQ and ACK in Figure 1-2).

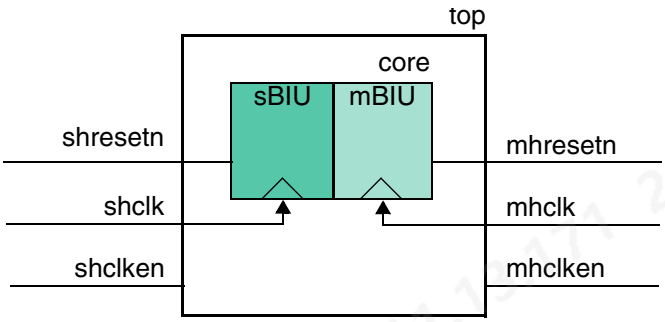
The clocking scheme determines the way in which the two BIUs are clocked and reset, and how each BIU makes use of the clock enable inputs. DW_ahb_h2h supports five different clocking schemes that correspond to the values 0 to 4 of the Clocking Mode configuration parameter (H2H_CLOCK_MODE; see "Parameter Descriptions" on page 49).

The clocking schemes are described in the following sections.

2.1.1 Asynchronous Mode

This operation mode is selected when the Clocking Mode configuration parameter is set to asynchronous (H2H_CLOCK_MODE = 0; see "Parameter Descriptions" on page 49). As illustrated in Figure 2-2, the sBIU is clocked by shclk and reset by shresetn. The mBIU is clocked by mhclk and reset by mhresetn. Clock enable inputs are ignored.

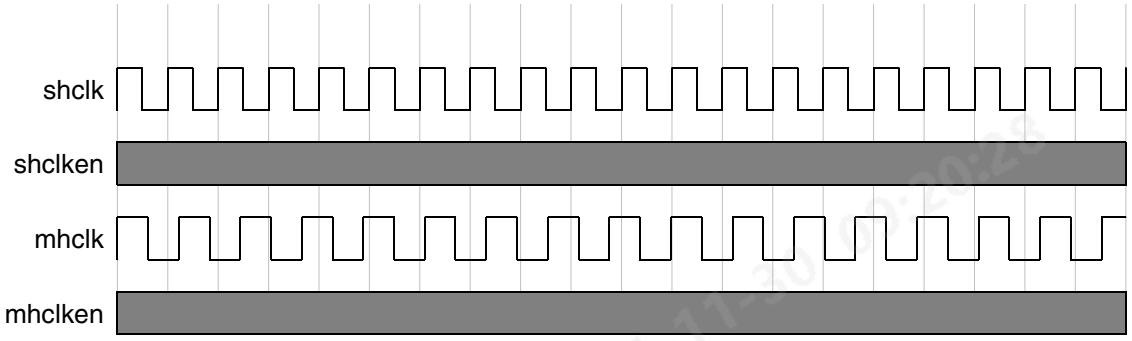
Figure 2-2 Asynchronous Clocking Mode



1. A synchronization failure occurs when metastability effects propagate beyond the synchronization flip-flops and cause incorrect results in the functional logic that is placed after the synchronization stage.

For an example of the timing for the asynchronous mode, see Figure 2-3

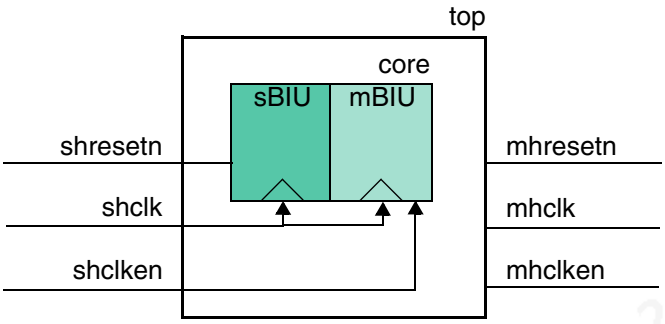
Figure 2-3 Asynchronous Clocking Mode Timing



2.1.2 Synchronous Mode with shclk and shclken

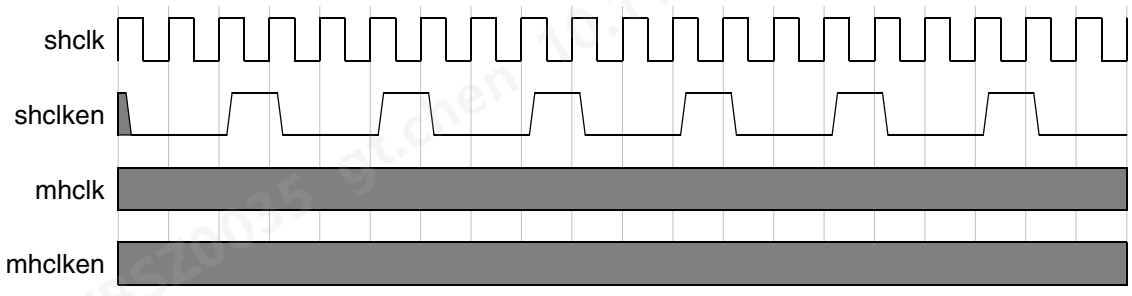
This operation mode is selected when the Clocking Mode configuration parameter is set to synchronous mode, single clock (H2H_CLOCK_MODE = 1; see "Parameter Descriptions" on page 49). As shown in Figure 2-4, both the sBIU and the mBIU are clocked by shclk, and reset by shresetn. The shclken signal acts as an enable in mBIU flip-flops.

Figure 2-4 Synchronous Clocking Mode with shclk and shclken



For an example of the timing for the synchronous mode with shclk and shclken, see Figure 2-3.

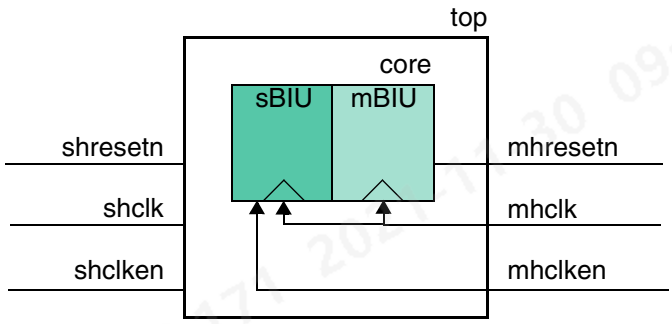
Figure 2-5 Synchronous Clocking Mode Timing for 1:3 shclk/shclken Ratio



2.1.3 Synchronous Mode with mhclk and mhclken

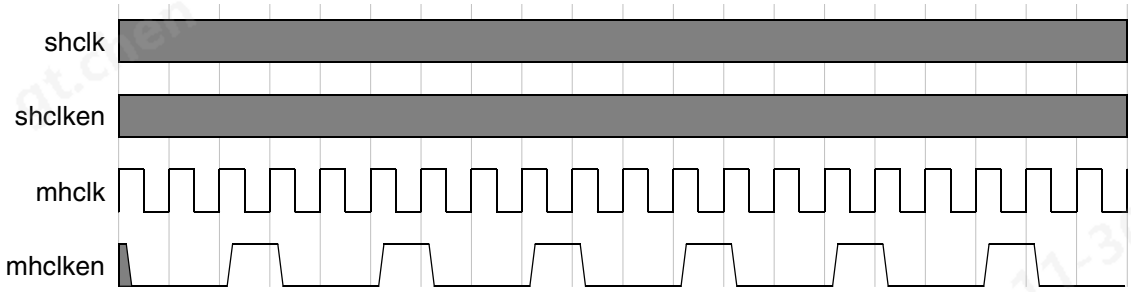
This operation mode is selected when the Clocking Mode configuration parameter is set to synchronous mode, single clock (H2H_CLOCK_MODE = 2; see "Parameter Descriptions" on page 49). As shown in Figure 2-6, both the sBIU and the mBIU are clocked by mhclk and reset by mhresetn. The mhclken signal is used as an enable in sBIU flip-flops.

Figure 2-6 Synchronous Clocking Mode with mhclk and mhclken



For an example of the timing for the synchronous mode with mhclk and mhclken, see Figure 2-7.

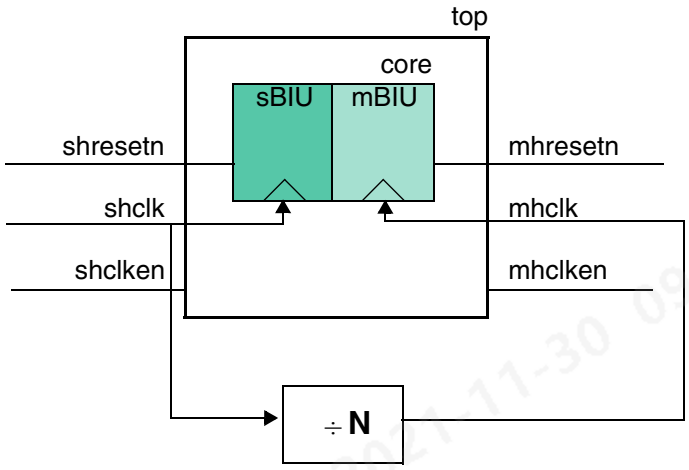
Figure 2-7 Synchronous Clock Mode Timing for 1:3 mhclk/mhclken Ratio



2.1.4 Synchronous Mode with Dual Clocks, shclk Faster Clock

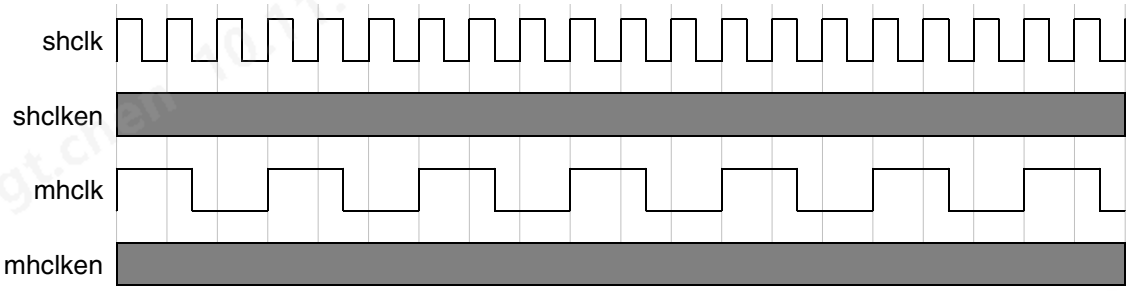
This operation mode is selected when the Clocking Mode configuration parameter is set to synchronous mode with dual clocks (H2H_CLOCK_MODE = 3; see "Parameter Descriptions" on page 49). As illustrated in Figure 2-8, the sBIU is clocked by shclk and reset by shresetn, while the mBIU is clocked by mhclk and reset by mhresetn. The clock enable inputs are ignored. Clocks are treated as synchronous with in-phase positive edges—shclk is treated as the faster clock; mhclk is the slower clock.

Figure 2-8 Synchronous Mode with shclk and mhclk, shclk Faster Clock



For an example of the timing for the synchronous mode with dual clocks and shclk, see Figure 2-9.

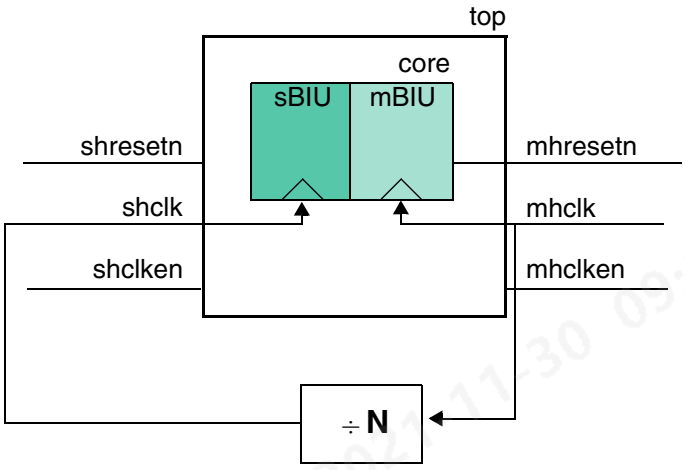
Figure 2-9 Synchronous Mode Timing for 1:3 shclk/mhclk Ratio



2.1.5 Synchronous Mode with Dual Clocks, mhclk Faster Clock

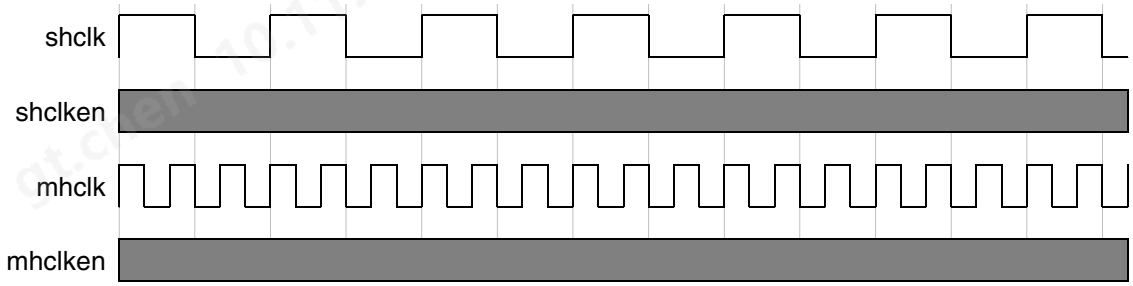
This operation mode is selected when the Clocking Mode configuration parameter (see "Parameter Descriptions" on page 49) is set to synchronous mode with dual clocks (H2H_CLOCK_MODE = 4; see "Parameter Descriptions" on page 49). As shown in Figure 2-10, the sBIU is clocked by shclk and reset by shresetn, while the mBIU is clocked by mhclk and reset by mhresetn. The clock enable inputs are ignored. Clocks are treated as synchronous with in-phase positive edges—mhclk is treated as the faster clock; shclk is the slower clock.

Figure 2-10 Synchronous Mode with shclk and mhclk, mhclk Faster Clock



For an example of the timing for the synchronous mode with dual clocks and mhclk, see Figure 2-11.

Figure 2-11 Synchronous Mode Timing for 1:3 mhclk/shclk Ratio



2.2 Data Path Retiming

The data path is defined as the subset of AHB signals that constitute the information content of an AHB transfer. Data path signals can be divided into three classes:

- Transfer address and attributes
  - haddr
  - htrans
  - hburst
  - hsize
  - hprot
  - hmastlock
- Write data
  - hwdata
- Read data and transfer response
  - hrdata

- hresp

The DW_ahb_h2h provides data path registers for each class of signals. This is illustrated in Figure 1-2 on page 15. Address and attributes are sampled from the primary bus and registered in the shclk clock domain. After the register is loaded, the content is placed (output MUX logic on the address path in Figure 1-2) on the secondary bus. The operation is timed by mhclk. When the multiplexers are open, the registers are not updated to ensure that shclk-timed transitions on the primary signals are never propagated to the secondary bus.

Write data are sampled from the primary bus and registered in the mhclk domain. The register can be loaded only when the primary bus is held off by shready low; this ensures a stable shwdata value on the loading mhclk clock edge.

Read data and transfer response are sampled from the secondary bus in the mhclk domain. After the register has been loaded, the content is placed (output MUX logic on the read data path in Figure 1-2) on the primary bus. The operation is timed by shclk. When the multiplexers are open, the registers are not updated in order to ensure that mhclk-timed transitions on the secondary signals are never propagated to the primary bus. To facilitate system-level timing closure, the read data and response multiplexers are always open for at least two shclk clock periods—that is, shrdata and shresp are treated as multi-cycle paths. The first clock period occurs with shready_resp low. The second occurs with shready_resp high.

When the output multiplexers are closed, all zeros are driven on address, attribute, and read data.

2.3 Transfer Forwarding

An access on the slave signal is forwarded as a master transfer onto the secondary interface. The response received by the master from the secondary interface is forwarded back to the slave. The forwarded transfer has the following characteristics:

- Adapted address and write data bus width
- Adapted read data bus width
- Adapted endianness
- SPLIT response replaced by RETRY
- Same size, lock, and protection attributes
- Bursts broken into single NSEQ beats

For more information about the input and output signals for data forwarding, see “Transfer Forwarding” on page 38.

2.4 Data Flow Control

Data flow control (bus handshaking) is achieved by basic AHB protocol mechanisms doing the following:

- A primary transfer is recognized by the sBIU when shsel is active and shtrans is SEQ or NSEQ. If the transfer is qualified as locked (shmastlock high), IDLE and BUSY cycles also are recognized. When a transfer is recognized, the sBIU activates the mBIU (REQ signal in Figure 1-2 on page 15). The shready_resp signal is driven low. The primary bus is held off until the acknowledge (ACK signal in Figure 1-2) is received. The acknowledge is generated by mBIU upon data phase completion.

- After mBIU has been activated, the secondary bus is requested. The mhbusreq signal (and possibly mhlock) is asserted. Once the address phase is granted, as indicated by mhgrant and mhready being sampled high, mhbusreq (mhlock) can be de-asserted. The mhbusreq signal can remain asserted, depending on whether the current address phase is the last transfer or whether additional back-to-back transfers are expected to follow.
- While the mBIU executes a sequence of back-to-back transfers, the primary is held off with mhtrans=IDLE and mhbusreq asserted. This happens only when the sBIU requests locked transfers or back-to-back NSEQ/SEQ non-locked transfers.

2.5 Deadlock

An mhclk synchronous time-tick signal (see "Signal Descriptions" on page 59) is available as an input to the mBIU (see "Signal Descriptions" on page 59). Positive edges on mttick are internally detected and counted to provide an indication for a deadlock condition to the mBIU. A deadlock condition is detected when the master interface has a pending transfer to perform (mhbusreq is asserted) and when mhgrant is not asserted before two consecutive mttick positive edges are counted.

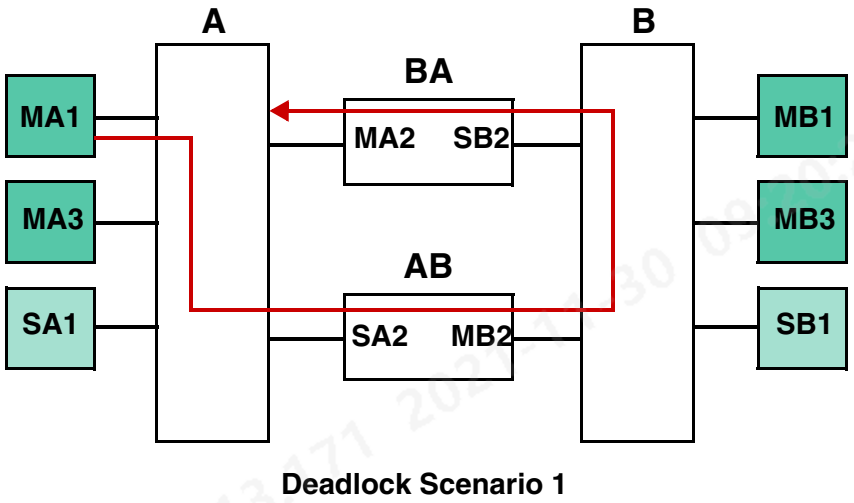
The period of the time-tick signal is under user control (from few to many mhclk clock periods, depending on the application). If a transfer takes too long to complete, it is aborted—that is, the sBIU returns a SPLIT response to the primary bus—and the DW_ahb_h2h clears the split master on the next positive edge detected on the mttick signal. If the DW_ahb_h2h component is not split-capable (see the parameter list in 3 on page 49), the SPLIT response is replaced by an ERROR response.

SPLIT response is used to protect the system from deadlock, especially for bidirectional applications. For example, suppose there are two AHB buses—bus A and bus B connected through two bridges, AB and BA, respectively. There are two potential deadlock scenarios in this example system:

1. Bus A initiates a transfer to bridge AB, which stalls bus A and requests bus B. Bus B is free, so bridge AB initiates a transfer in bus B, at which point bus B routes the transfer to bridge BA. Bridge BA stalls bus B and requests bus A, which is busy; this results in a deadlock condition.
2. Bus A initiates a transfer to bridge AB. At almost the same time, bus B initiates a transfer to bridge BA. Bridge AB stalls bus A and requests bus B. Bridge BA stalls bus B and requests bus A. Bus A and bus B are both busy, resulting in a deadlock condition.

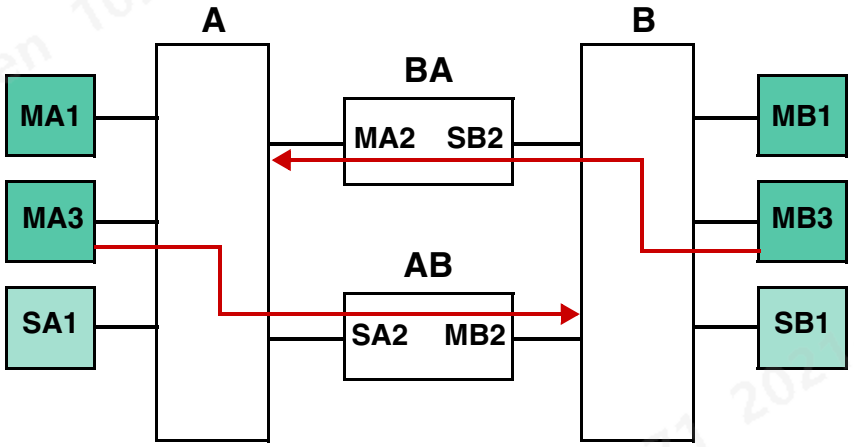
The previous scenarios are illustrated in Figure 2-12 and Figure 2-13.

Figure 2-12 Deadlock Scenario for Address Loop



Deadlock Scenario 1

Figure 2-13 Deadlock Scenario for Same-Time Access



Deadlock Scenario 2

The DW_ahb_h2h component does not cope with scenario 1. The issue must be resolved at the system level, ensuring that address spaces are properly allocated in both bus A and bus B. The address range of bridge BA should never overlap with the address range forwarded by bridge AB.

The DW_ahb_h2h copes with scenario 2, with the combination of the mttick input to the mBIU and the SPLIT response generation capability in the sBIU. When a SPLIT response is generated in scenario 2 (for example, by SA2 of Bridge AB; see Figure 2-13), the master that was accessing the bridge is removed (MA3). The bus on which the SPLIT response is issued (bus A) becomes free for the other bridge (MA2 of bridge BA), and the deadlock is resolved. For that mechanism to operate effectively, it is important to ensure that the two time-tick waveforms (inputs for MA2 and MB2 in scenario 2) are properly decoupled. The mttick input based timeout mechanism can be used in scenario 2 for both normal/locked transfers to bring the system out of deadlock.

You can use the H2H_IS_SPLIT_CAPABLE parameter in coreAssembler in order to create a split response. This H2H_IS_SPLIT_CAPABLE parameter is propagated from the interface connected to the DW_ahb_h2h slave interface. If you export the interface, the DW_ahb_h2h inherits its configuration for the H2H_IS_SPLIT_CAPABLE parameter from the exported interface connection. In this case, you may see that the H2H_IS_SPLIT_CAPABLE parameter is checked yes, but is also greyed out so that you cannot disable it.

If you need to disable this inherited condition, follow these steps in coreAssembler:

1. In the Add Subsystem Components activity, right click on the exported slave interface icon (green interface pin on the left side of the schematic).
2. Go to “Edit Interface parameters” and check the “no” button next to Split Capable.

Now when you go back to the “Configure Component” step and select the configuration for the DW_ahb_h2h connected to this interface, you can see that the Split Capable is disabled and greyed out. It remains greyed out because the DW_ahb_h2h connections inherit the settings from the interface to which they are connected.

2.6 Bypass

The Clocking Mode parameter (H2H_CLK_MODE) determines the internal synchronization architecture and how clock enables are used by the core. An external input signal is provided to enable dynamic bypass (such as, during operation) of all synchronization stages. This signal must always be 0. It can be driven to 1 only in one of the following exceptional cases:

- IP test, for example FPGA validation
- Primary and secondary interfaces driven by the same clock
- Special test cases of different synchronous clocks

When synchronizers are bypassed, the throughput of the DW_ahb_h2h is maximum. For more information about the input and output signals that drive this functionality, see “Synchronizers Bypass” on page 47.

2.7 AMBA 5 AHB Support

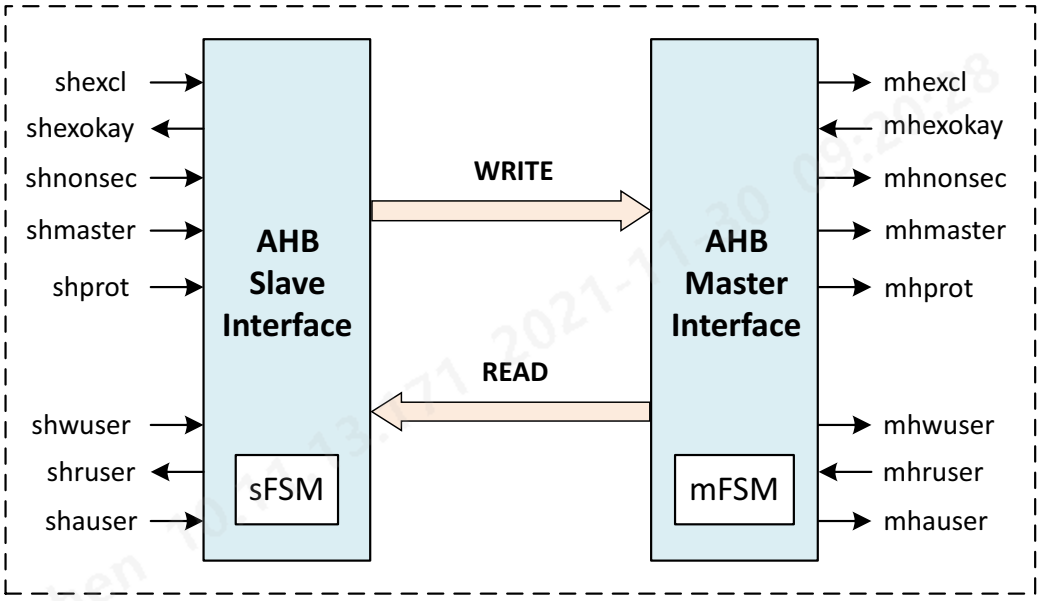
You must have DWC-AMBA-AHB5-Fabric-Source add-on license to access AHB5 features (H2H_AHB_INTERFACE_TYPE==1).

DW_ahb_h2h is updated to support the following features as part of the AMBA 5 AHB protocol specification.

- “Secure Transfer” on page 29
- “Exclusive Transfer” on page 31
- “Endian Conversion Support” on page 34
- “User Signal Support” on page 37

Figure 2-14 describes the additional signals add for AMBA 5 AHB enhancement.

Figure 2-14 DW_ahb_h2h Block Diagram with Newly Added Signals



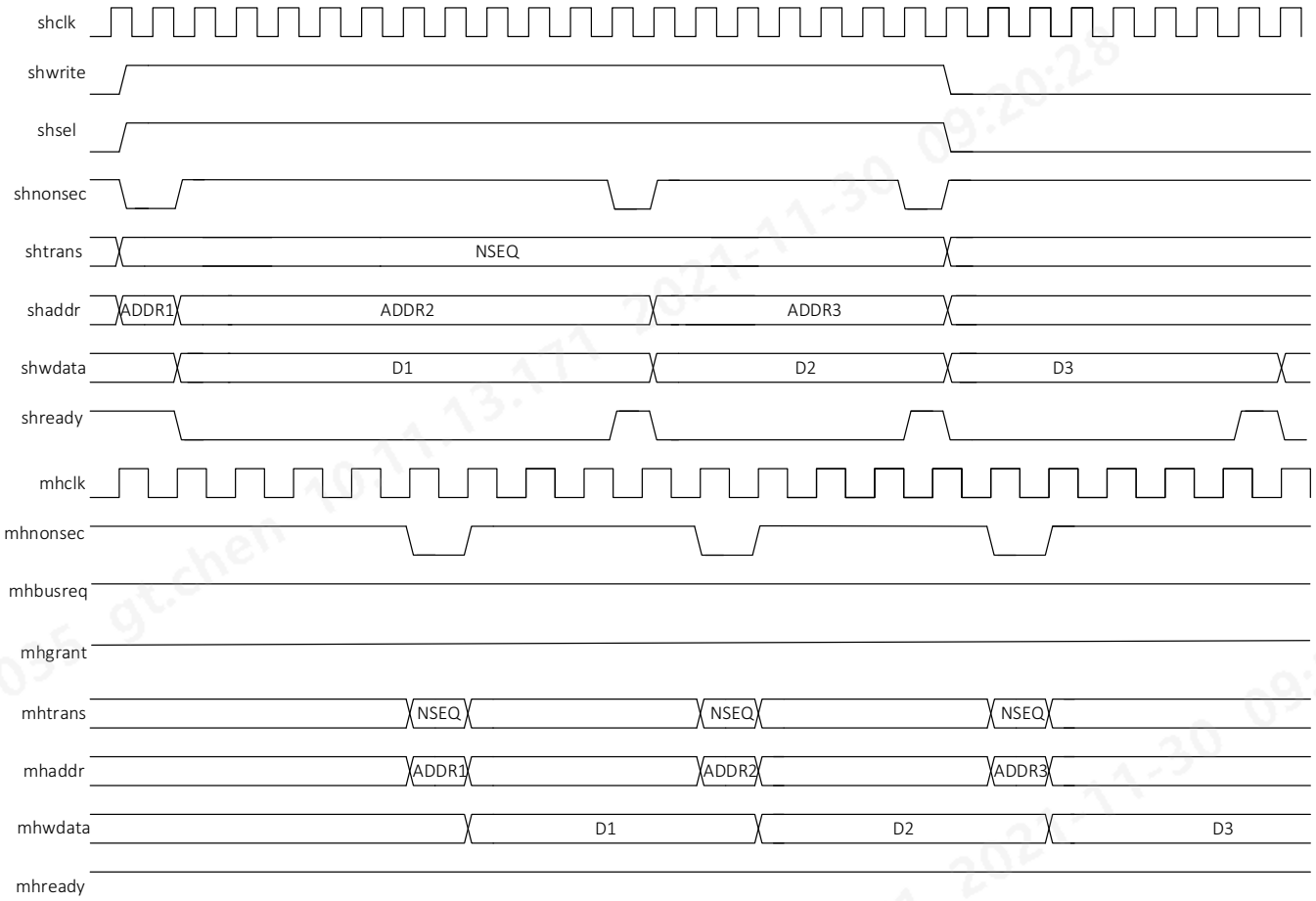
2.7.1 Secure Transfer

AMBA 5 AHB extends the TrustZone security foundation from the processor to the entire system. This way, DW_ahb_h2h supports secure transfer. The shnonsec signal is added to distinguish between a secure and non-secure transfer in the DW_ahb_h2h slave interface. The shnonsec signal is an address phase signal. This signal is asserted for a non-secure transfer and de-asserted for a secure transfer.

2.7.1.1 Secure Write Transfer

Figure 2-15 describes a secure write transfer in DW_ahb_h2h.

Figure 2-15 Secure Write Transfer

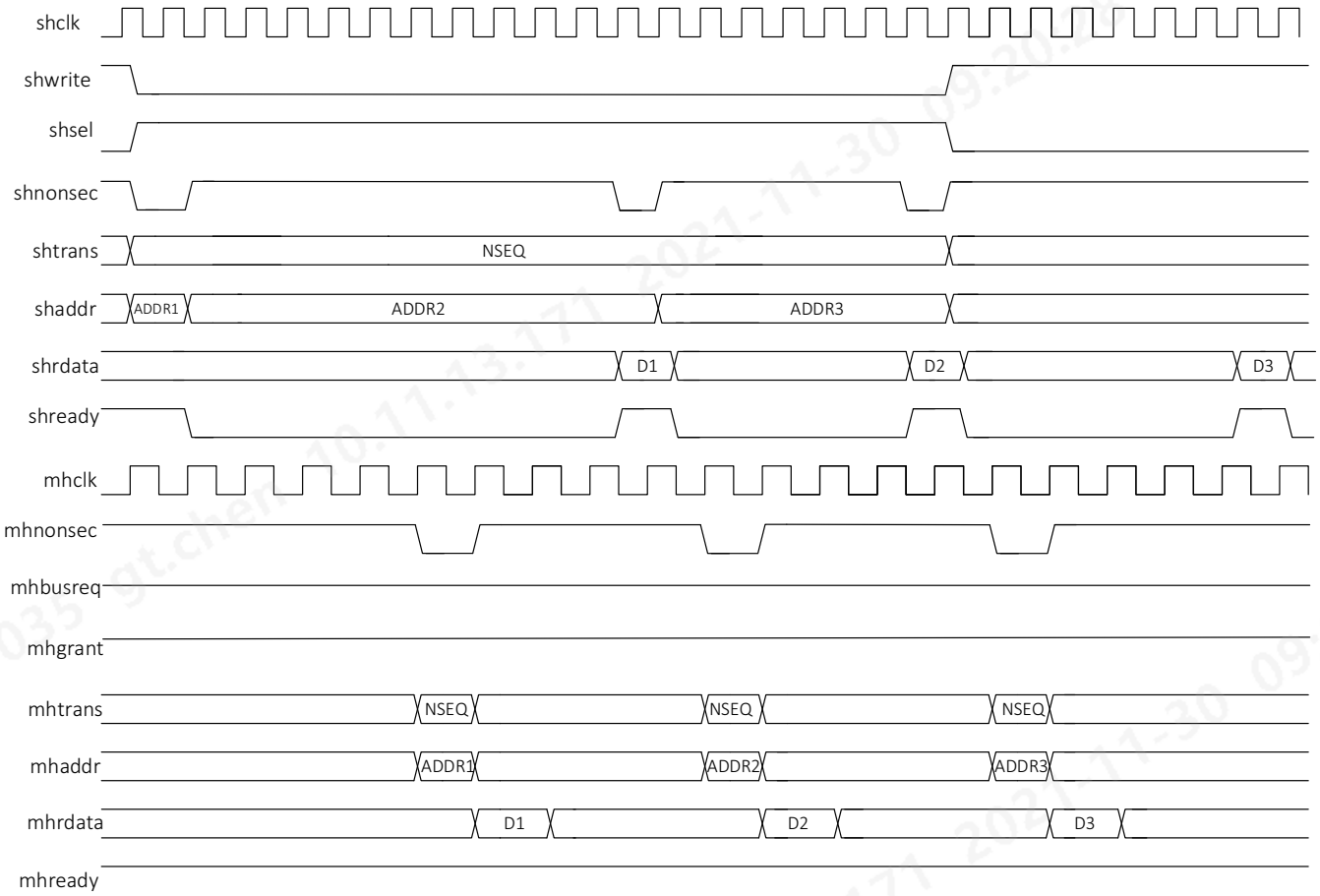


Three consecutive secure writes are performed in the timing diagram shown in Figure 2-16. Master selects the desired slave by asserting the hsel signal. The address and data are placed on the bus. The address is transferred to the slave after the bus grants the access to the desired slave. Slave is always ready to accept the data. The shready signal is asserted when the data is properly written to the desired address.

2.7.1.2 Secure Read Transfer

Figure 2-16 describes a secure read transfer in DW_ahb_h2h. The shwrite and shnonsec signals are pulled low to indicate that it is a secure read transfer.

Figure 2-16 Secure Read Transfer



2.7.2 Exclusive Transfer

DW_ahb_h2h Exclusive transfer feature provides a mechanism to support semaphore-type operations. The sequence of activities for a normal exclusive transfer is explained in the following steps:

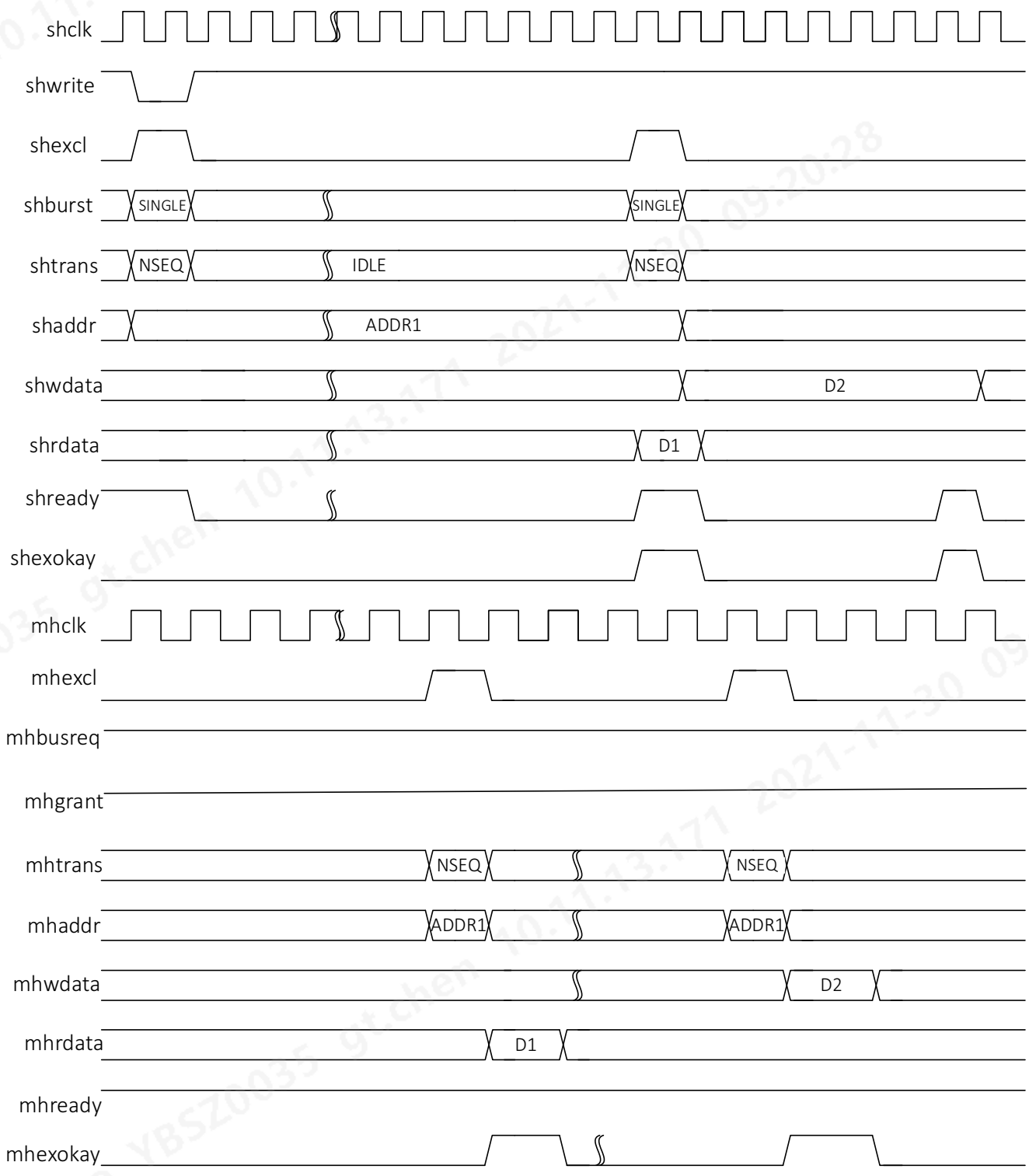
1. Perform an exclusive Read transfer (ADDR1).
2. Calculate the new value to be written to the ADDR1 address using the data obtained from exclusive read.
3. Perform the exclusive write transfer to the ADDR1 address with the new value.
   a. Success - (memory is updated) - If no other master has written to that location since the Exclusive Read.

b. Failure - (memory is not updated) - If another master has written to that location since the Exclusive Read.

4. The exclusive write response is sent to master indicating success or failure.

Figure 2-17 describes the timing diagram for an exclusive access sequence in DW_ahb_h2h.

Figure 2-17 Exclusive Access Sequence



Note

- There can be non-exclusive transfers in between an Exclusive Read and an Exclusive Write.
- If an Exclusive write transfer fails, master is expected to repeat the entire Exclusive access sequence.

2.7.3 Endian Conversion Support

AMBA 5 AHB specifies the following two forms of big-endian data access:

- **BE8** - Byte invariant Big endian is derived from the fact that a byte access (8-bit) uses the same data bus bits as a little-endian access for the same address.
- **BE32** - Word invariant Big endian is derived from the fact that a word access (32-bit) uses the same data bus bits for the Most Significant (MS) and the Least Significant (LS) bytes as a little-endian access for the same address.

Figure 2-18 shows how to interpret the endian transformation diagrams.

Figure 2-18 Key for Endian Transformation Diagrams

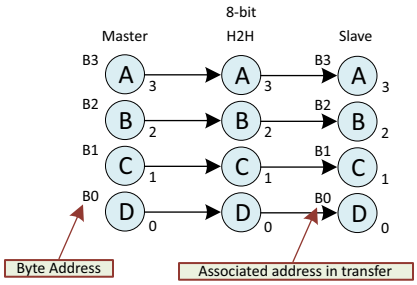
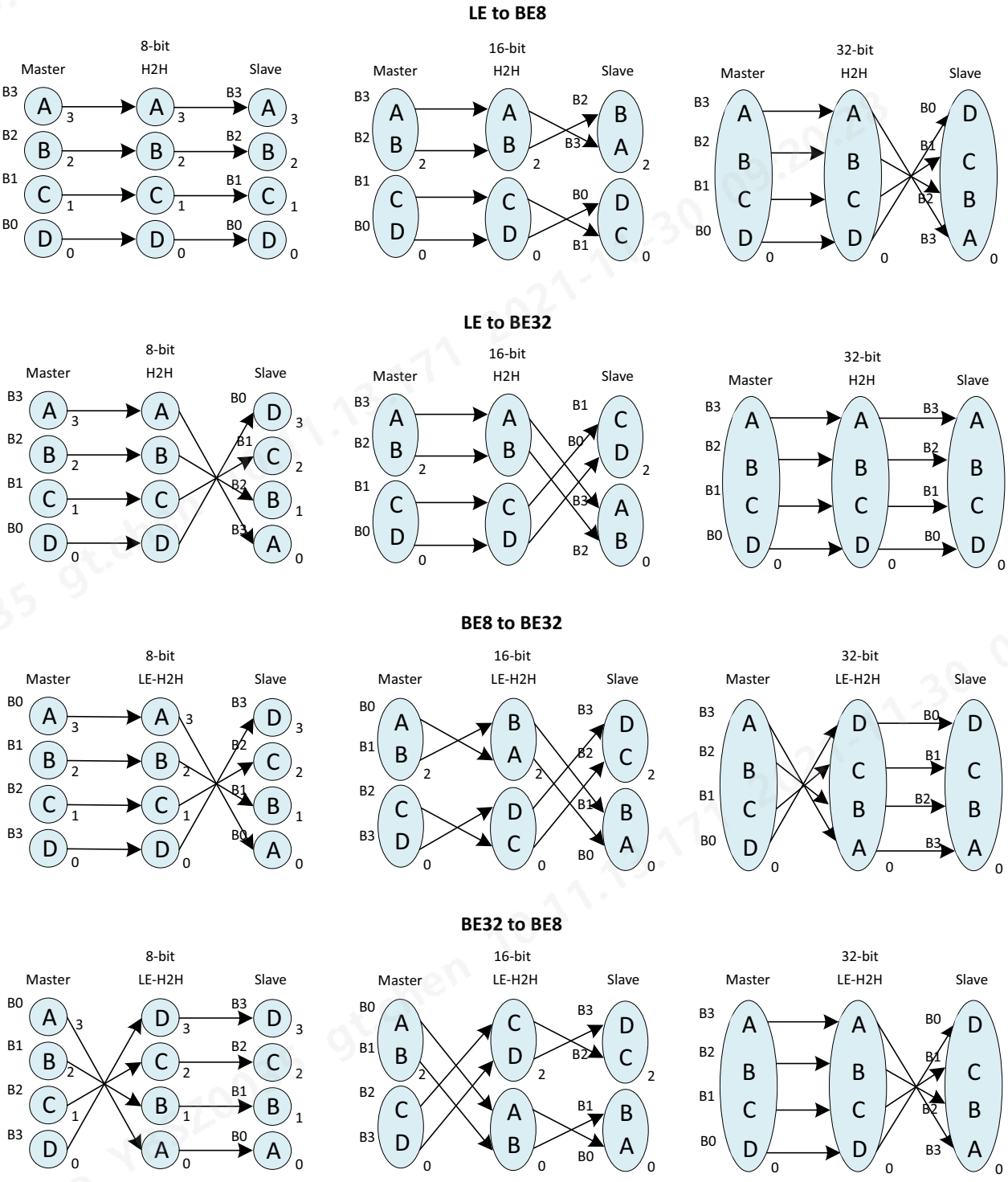


Figure 2-19 Endian Conversion Diagram

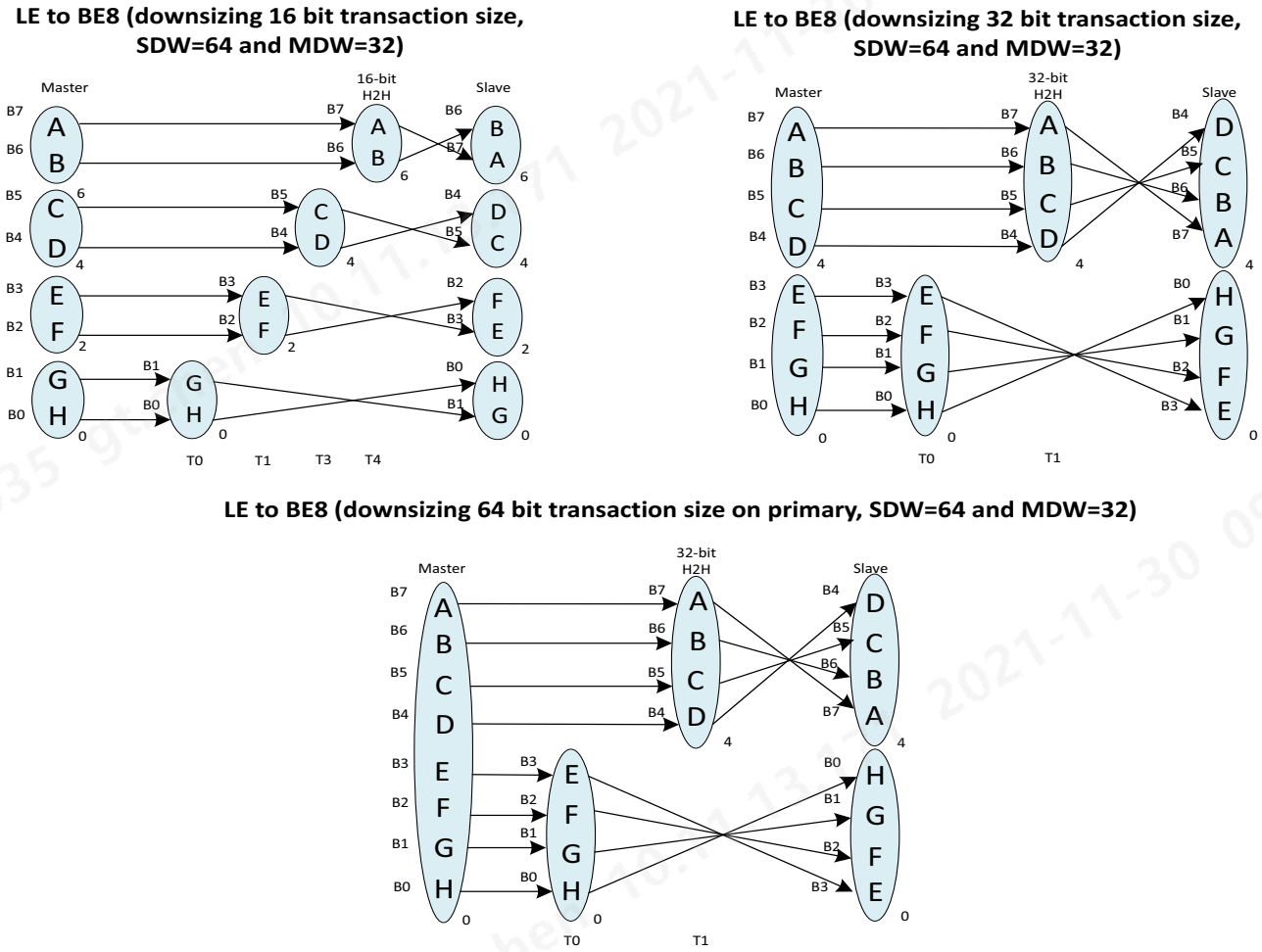


See "Comparison of Endian Schemes" on page 91

Figure 2-20 shows the endian conversion diagram for a downsizing configuration.

- The secondary interface is configured for Big-endian (BE8).
- The primary data port width is greater than the secondary interface.
- The primary interface is configured with 16, 32 and 64-bit transaction size.
- The endian mapping transformation is performed using the secondary side transaction size.

Figure 2-20 Endian Conversion for Downsizing Configuration



Note Select a valid H2H_PHY_SBIG_ENDIAN and H2H_PHY_MBIG_ENDIAN values for the big endian modes (BE8 and BE32) support.

2.7.4 User Signal Support

DW_ahb_h2h supports user signals for the address and data channels. DW_ahb_h2h supports the following two user signal transfer modes:

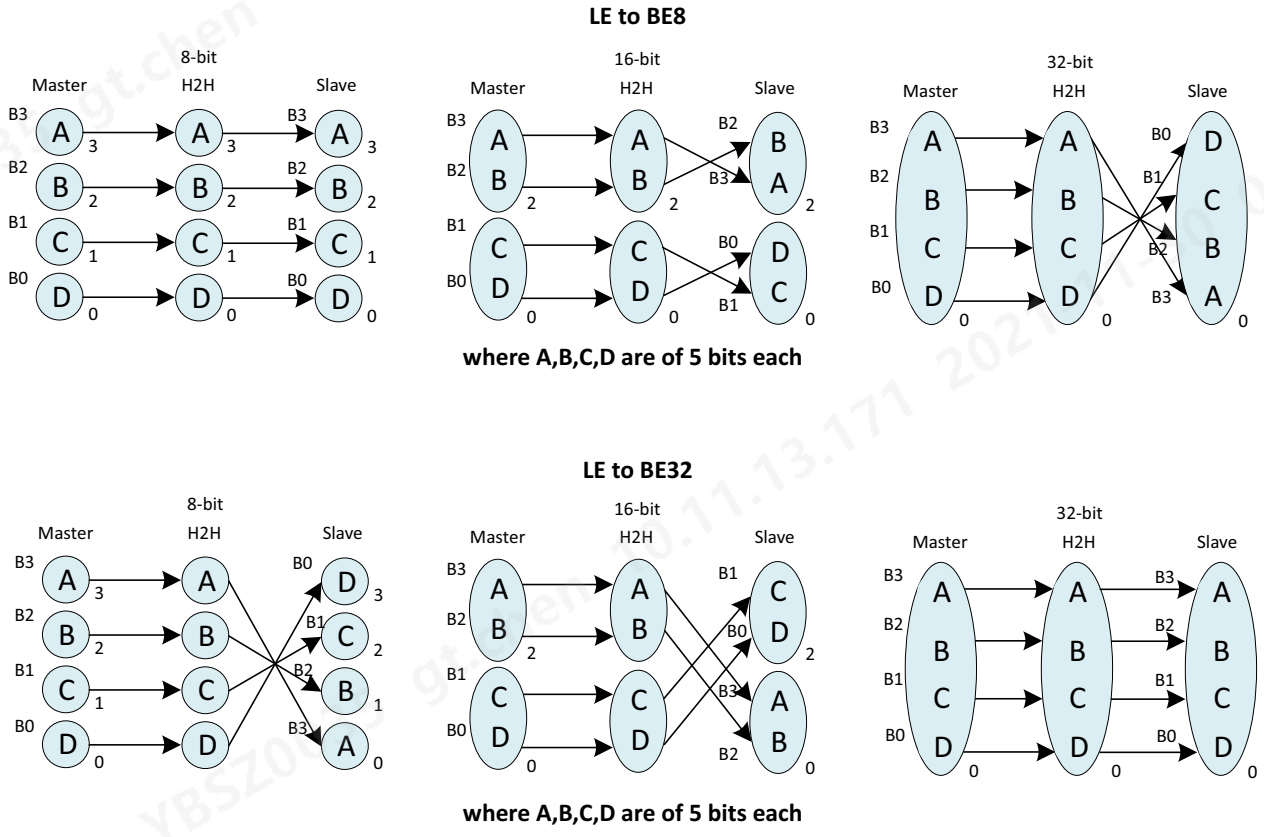
- Passthrough mode - In Pass through signal transfer mode, the user signals are transported from the slave interface to the master interface without any change.
- Data Aligned mode - In Data Aligned mode, the user channel width is aligned to the respective data bus width. Specify the number of user bits per byte (RUSER_BITS_PER_BYTE and WUSER_BITS_PER_BYTE) based on which the user channel width is calculated.

You can select the user signal transfer mode using the USER_SIGNAL_XFER_MODE parameter.

An example of Data Aligned mode in AHB 5 configuration:

- H2H_PHY_SDATA_WIDTH = 32
- WUSER_BITS_PER_BYTE = 5
- H2H_SHRUSER_WIDTH = (32/8)*5 = 20

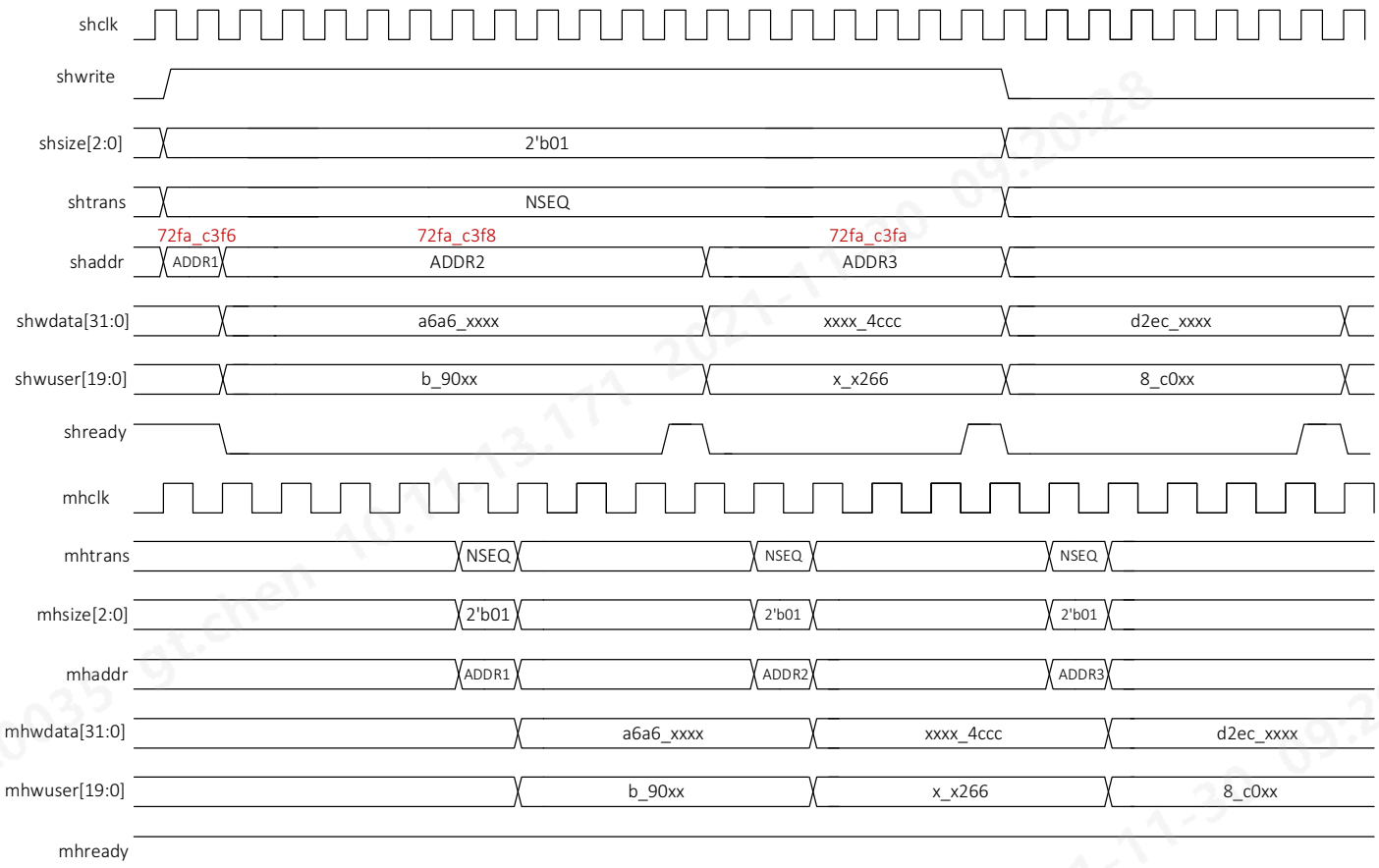
Figure 2-21 Endianness for Data Aligned User Signals



In data aligned mode, the number of bits per byte are configurable from 1 to 8. In Figure 2-21, the write user bits per byte is 5 and the endian conversion occurs for a block of 5 bits.

Figure 2-22 waveform shows the write channel user signals for data aligned mode for the above configuration. The endian mode is 'Little-endian' on both master and slave interfaces.

Figure 2-22 User Signalling in Data Aligned Mode



2.8 Additional Notes on Functionality

The following functions and input/output signals of the DW_ahb_h2h depend on specific configuration settings.

2.8.1 Transfer Forwarding

This section describes transfer forwarding (previously described on page 25) in more detail, along with the various inputs and outputs that deal with this functionality.

The forwarding functionality consists of converting AHB transfers from primary to secondary buses, and converting AHB responses from secondary to primary buses.

The functionality is activated as soon as the slave signal is accessed by the primary AHB. An access is recognized by the sBIU when the following is detected on the slave signal:

- shsel = 1
- shready = 1

- shtrans = NSEQ or SEQ

A locked access is recognized when the following is the case:

- shsel = 1
- shready = 1
- shmastlock = 1 and shtrans = any

When an access is recognized, the following two situations occur:

- A request toggle REQ for the mBIU is generated.
- The shready_resp signal is driven low.

The request toggle is synchronized according to the configured clocking scheme and forwarded to the mBIU. When the mBIU receives a request, it produces a transfer on the secondary AHB with the following characteristics:

- mhbusreq and mhlock are driven according to the current bus grant status, sBIU request type (last, more to follow, and locked), and component configuration (AHB Lite protocol or full AHB).
- Adapted address and data widths, endianness
- Same size and protection
- mhtrans = NSEQ
- mhburst = SINGLE

When the secondary transfer data phase is completed with mhready, the mBIU samples mhresp (and mhrdata if needed) and generates an acknowledge toggle back to the sBIU. The ACK toggle is synchronized according to the configured clocking scheme and forwarded back to the sBIU. When the sBIU receives the ACK, it completes the pending primary transfer as follows:

- The shready_resp signal is driven high.
- The shresp signal is driven according to mhresp (SPLIT masked as RETRY).
- The shrdata signal is driven according to mhrdata.

Example timing diagrams are shown in Figure 2-23, Figure 2-24 on page 41, Figure 2-25 on page 42, Figure 2-26 on page 43, and Figure 2-27 on page 44.

Figure 2-23 Asynchronous Mode Timing (H2H_CLOCK_MODE = 0) – Three Consecutive Transfers (A0-A1-A2)

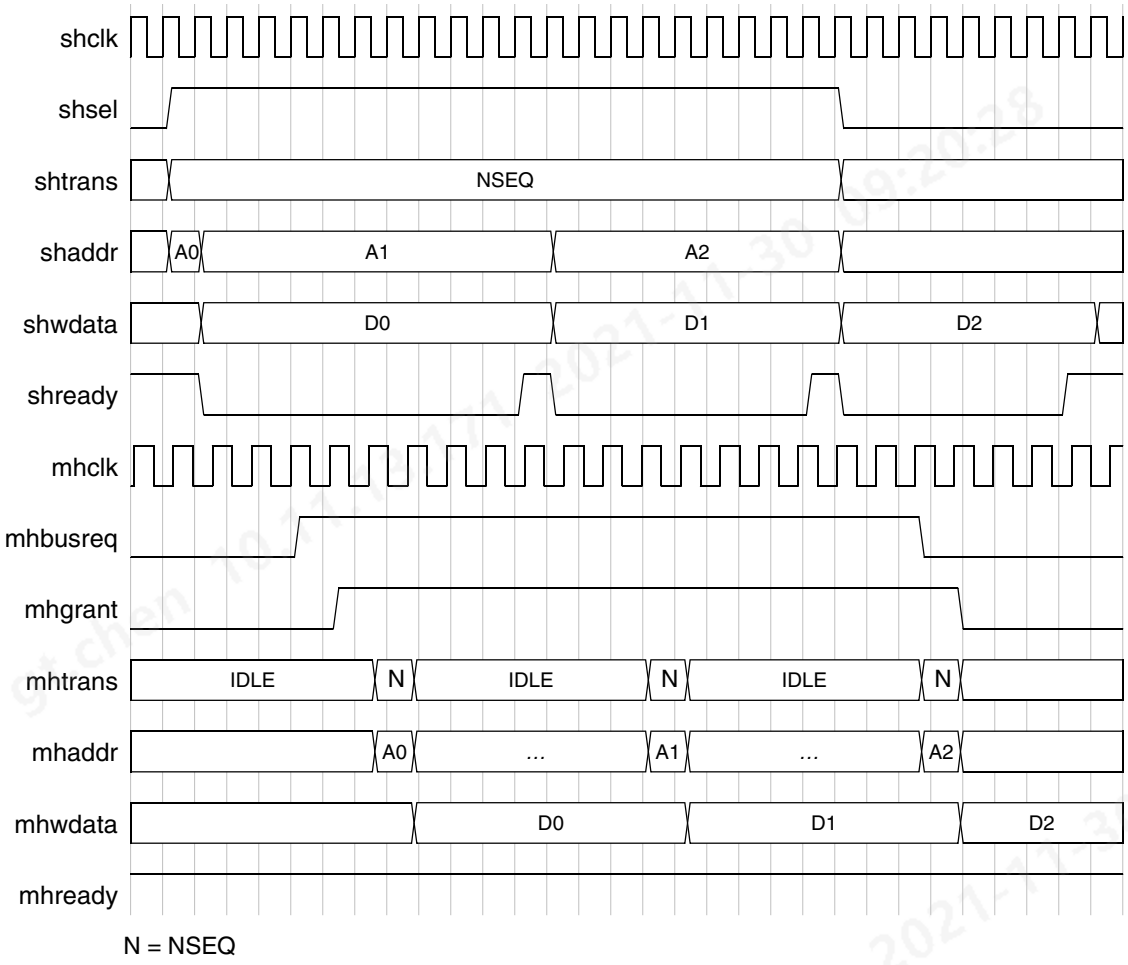


Figure 2-24 Synchronous Clocking Mode Timing (H2H_CLOCK_MODE = 1, 2:1 Frequency Ratio) – Three Consecutive Transfers (A0-A1-A2)

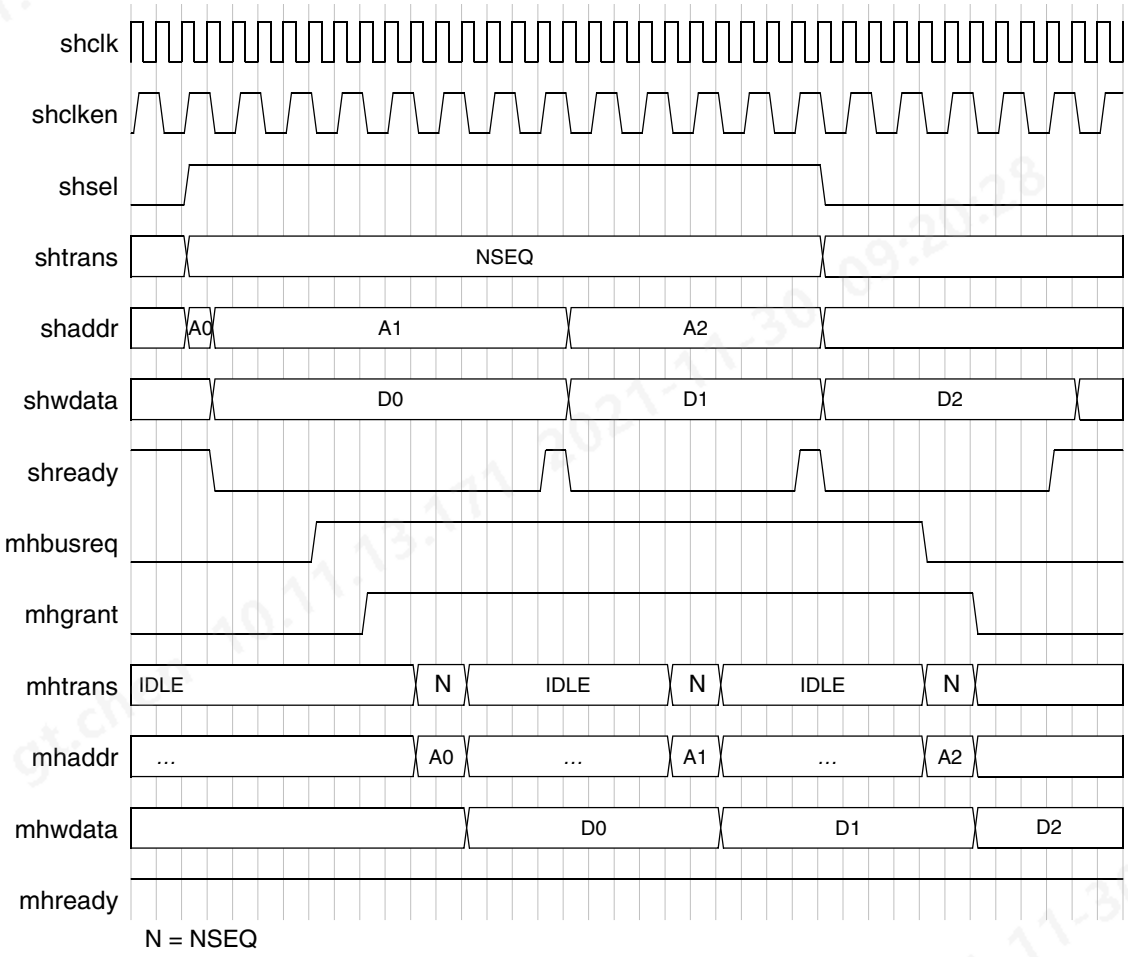


Figure 2-25 Synchronous Clocking Mode Timing (H2H_CLOCK_MODE = 2, 1:2 Frequency Ratio) – Three Consecutive Transfers (A0-A1-A2)

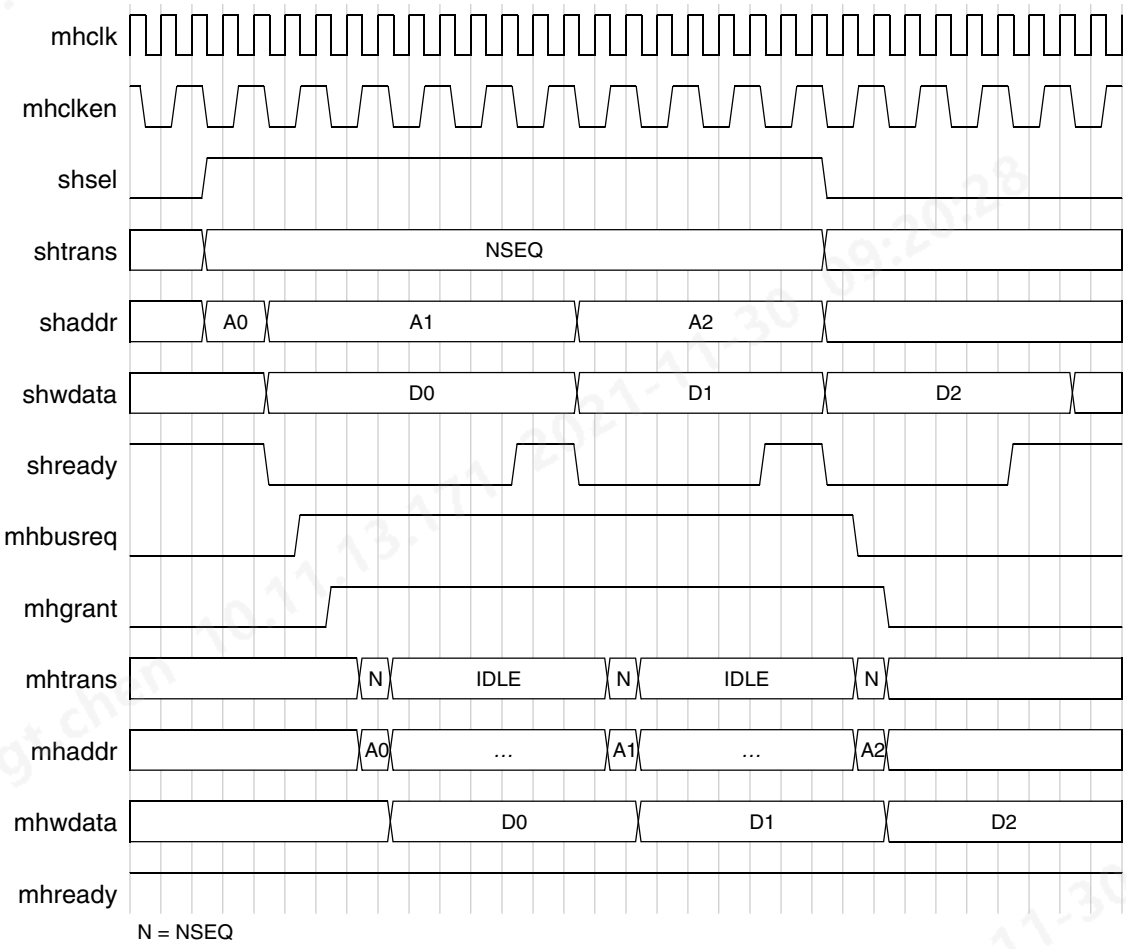


Figure 2-26 Synchronous Clocking Mode Timing (H2H_CLOCK_MODE = 3, 2:1 Frequency Ratio) – Three Consecutive Transfers (A0-A1-A2)

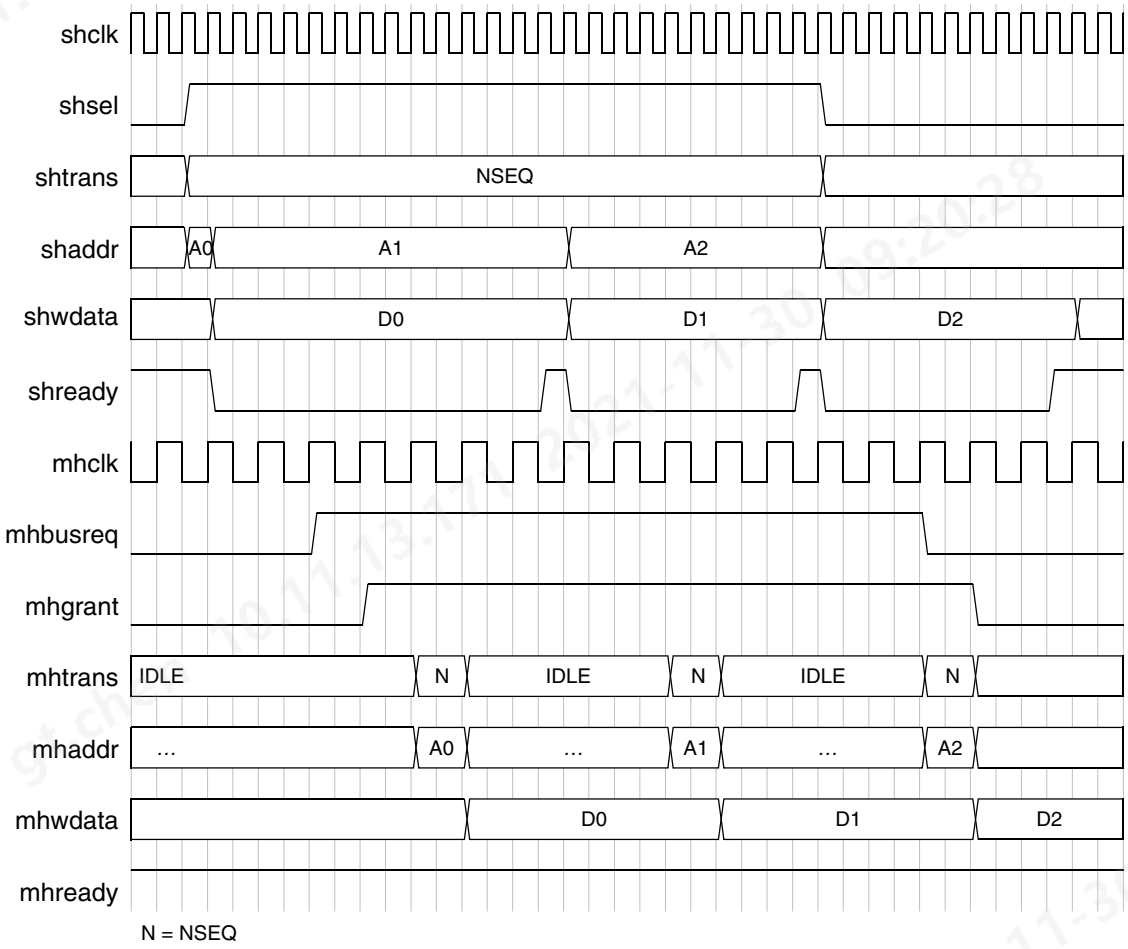
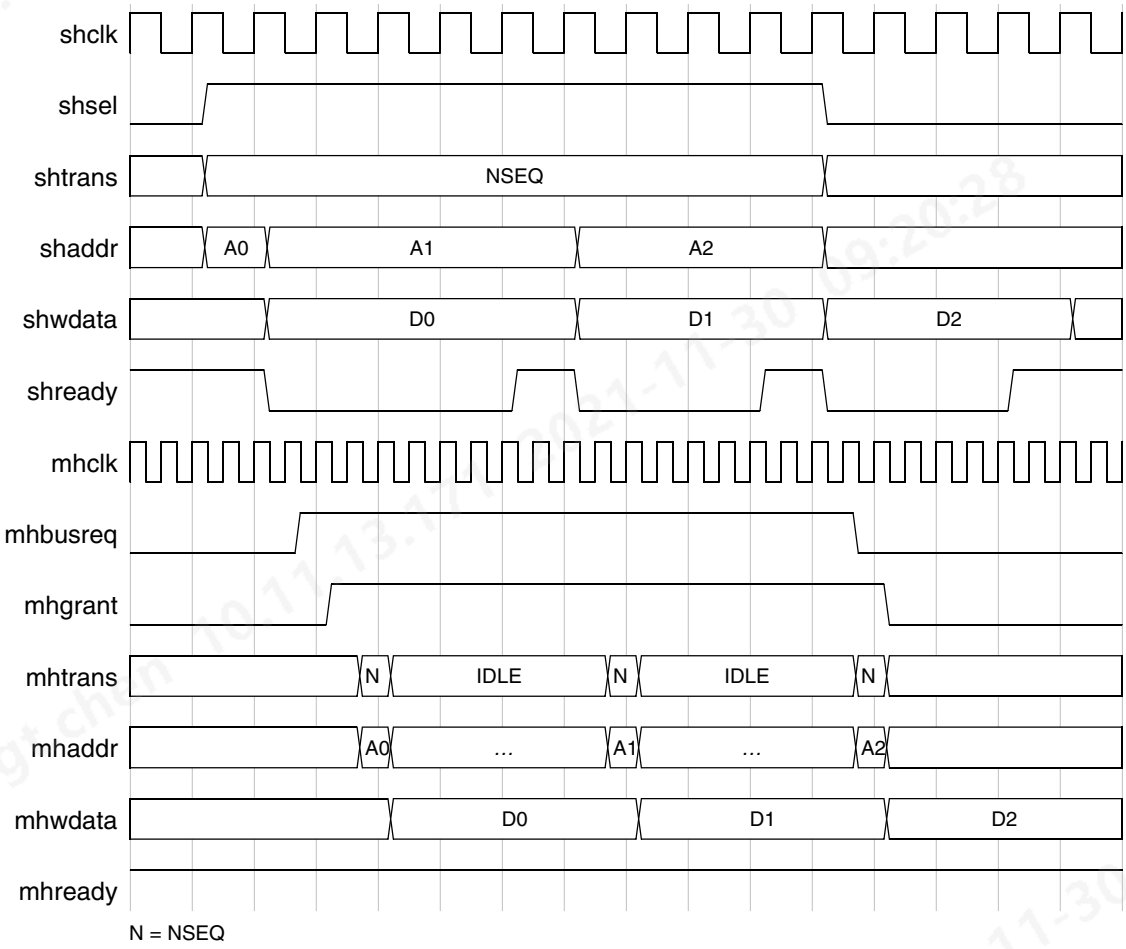


Figure 2-27 Synchronous Clocking Mode Timing (H2H_CLOCK_MODE = 4, 1:2 Frequency Ratio) – Three Consecutive Transfers (A0-A1-A2)



2.8.1.1 shaddr – mhaddr

Addresses always are forwarded unchanged when the bus width of the two interfaces is equal. If the primary is wider, the Most Significant Bits (MSBs) of shaddr are discarded. If the secondary is wider, the MSBs of mhaddr are padded with the value of the parameter H2H_PHY_MADDR_HIBIT.

Table 2-1 Address Translation

H2H_PHY_SADDR_WIDTH	H2H_PHY_MADDR_WIDTH	mhaddr
32	32	shaddr[31:0]
32	64	H2H_PHY_MADDR_HIBIT[31:0], shaddr[31:0]
64	32	shaddr[31:0]
64	64	shaddr[63:0]

2.8.1.2 shburst – mhburst

The mhburst signal is always set to SINGLE (3'b000). The shburst is unused by the core.

2.8.1.3 shmastlock – mhlock

When a slave access to the DW_ahb_h2h is qualified by shmastlock, the master interface asserts mhlock. If the master is already granted, one IDLE bus cycle is inserted before driving the address phase. If mhgrant is low, mhlock remains asserted until mhgrant becomes high.

2.8.1.4 shprot – mhprot

The shprot signals are always forwarded unchanged.

2.8.1.5 shsize – mhsize

The shsize signal always is forwarded unchanged. The DW_ahb_h2h does not perform any consistency check between the value of shsize and that of the H2H_PHY_SDATA_WIDTH or H2H_PHY_MDATA_WIDTH configuration parameters.

2.8.1.6 shtrans

BUSY and IDLE transfers qualified by shsel are ignored. NSEQ transfers are forwarded unchanged. SEQ transfers are forwarded as NSEQ. When shmastlock and shsel are asserted, BUSY and IDLE cycles on shtrans cause mhlock and mhbusreq to remain asserted.

2.8.1.7 mhtrans

The mhtrans signal is always IDLE unless an address phase is performed. In this case, mhtrans = NSEQ.

2.8.1.8 shwrite – mhwrite

These signals are always forwarded unchanged.

2.8.1.9 shwdata – mhwdata

The shwdata active byte lanes are routed into mhwdata according to shsize, shaddr least significant bits (LSBs), master data width, and endianness. The content of each individual active byte lane is preserved.

2.8.1.10 shsplit[15:0] – shmaster

After a SPLIT response, the DW_ahb_h2h always clears the split master on the next positive edge detected on the mttick signal. The split master is sampled from the shmaster input during the SPLIT response. To clear the SPLIT response, the shsplit bit corresponding to the split master number is asserted.

The DW_ahb_h2h shsplit output should be connected to the hsplit input of the AHB bus—for the DW_ahb, it should be connected to hsplit_s*J*[15:0], where *J* corresponds to the index of the DW_ahb_h2h slave on the AHB bus.

If you are using an AHB Lite bus which is not split capable, the hsplit[15:0] outputs can be left unconnected.

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

2.8.1.11 mhrdata – shrdata

The mhrdata active byte lanes are routed into shrdata according to mhsize, mhaddr LSBs, slave data width, and endianness. The content of each individual active byte lane is preserved. After reset, the shrdata

internal register is loaded with the component ID code. This value is placed onto the bus accessing the bridge when shsel_ID active.

2.8.1.12 mhresp – shresp

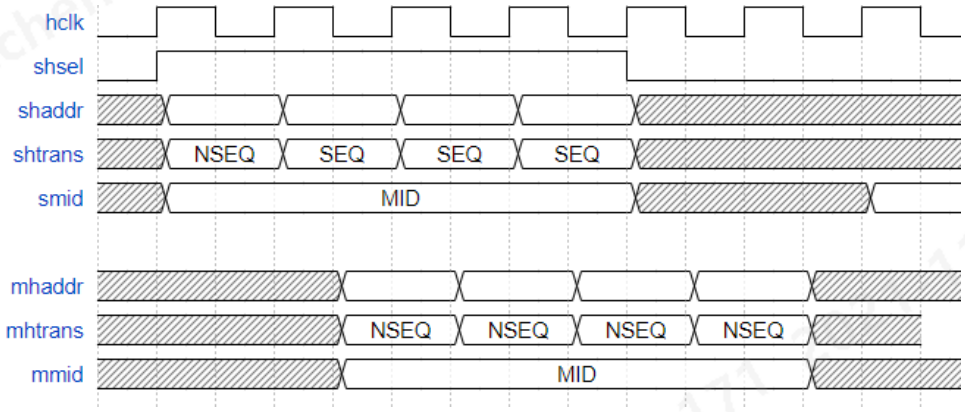
OKAY, ERROR, and RETRY values on mhresp are forwarded unchanged. A SPLIT response on the secondary interface is never forwarded. A SPLIT response is masked as RETRY. When the DW_ahb_h2h issues a SPLIT, the response is always the result of an internal time-out condition.

2.8.1.13 Non-Standard Master ID Sideband Signal

Sideband signal smid/mmid is used to transmit additional information about the transfer in progress on AHB slave. The sideband signal is expected to be constant for a burst operation. The H2H_MID_WIDTH parameter specifies the width of a non-standard Master ID sideband signals. When set to 0, the Master ID sideband signals are removed. The value of smid signal is directly transmitted as mmid signal on the secondary master interface. Figure 2-28 is an example AHB transaction in DW_ahb_h2h.

Note Non-standard mid signals are present in the AHB Lite mode. In AHB5 mode, the master ID is transmitted as the hmaster signal.

Figure 2-28 AHB Transaction in DW_ahb_h2h With MID Sideband Signal



2.8.2 Data Flow Control

This section describes the data flow control function (previously described on page 25) in more detail, along with the various inputs and outputs that deal with this functionality.

2.8.2.1 shready, shsel, and shready_resp

These signals are used by the sFSM (Figure 1-2) to perform bus-level data flow control on the primary AHB. The shready_resp signal is driven low as soon as both shsel and shready are sampled high. The signal is driven back to high on completion of the transfer secondary data phase. This means that the primary bus is always held off until the secondary data phase completes.

2.8.2.2 mhready, mhbusreq, mhgrant, and mhtrans

These signals are used by the mFSM (Figure 1-2) to perform bus-level data flow control on the secondary AHB. The mhbusreq signal is driven high as soon as a request is made by the sBIU. If the request is qualified as the last (shsel is de-asserted during the primary data phase), mhbusreq is driven back to low after mhgrant has gone high. The mhtrans signal is driven to IDLE in between consecutive transfers, while mhbusreq stays asserted. The mhready signal provides the timing for the acknowledge that the mBIU gives back in response to an sBIU request. If H2H_IS_LITE = 1, arbitration logic for hbusreq generation is not implemented, mhgrant is internally tied high, and mhbusreq is always driven high.

2.8.3 Deadlock

The mttick signal is an externally provided periodic signal (periodicity in the range of tens or hundreds of bus cycles). When H2H_IS_TTICK_SENSITIVE = 1, this periodic signal is used to prevent deadlock conditions that may arise, especially for bidirectional applications. When a transfer is forwarded by the DW_ahb_h2h, low-to-high transitions are detected on this signal and counted. If two mttick transitions are counted while a transfer is pending (primary system stalled, mhbusreq asserted) and mhgrant is not asserted, the transfer is aborted with a SPLIT response given back to the primary data phase. This is the only case where the DW_ahb_h2h inserts a SPLIT response. If H2H_IS_SPLIT_CAPABLE = 0, the data phase is aborted with an ERROR response.

2.8.4 Synchronizers Bypass

The SYNC_BYPASS signal allows the full bypass of synchronizers placed on handshaking signals between the sFSM and the mFSM. The signal should be normally driven to 0. It can be driven to 1 when the primary and secondary run with the same clock.

2.8.5 Component ID

At reset, the internal read data register is loaded with the component ID code. The position of the ID code byte within the register is determined by the endianness of the slave interface. Accessing the bridge with shsel_ID active allows this value to be placed onto the shrdata output bus.

2.8.6 BUS_ID

This signal is the 32-bit component ID code. It is driven with the VERSION_ID value. To determine this value, see the “AMBA 2 Release Notes” section in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Release Notes*.

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 Basic Configuration Parameters

Table 3-1 Basic Configuration Parameters

Label	Description
Basic Configuration	
AHB Lite master	Specifies whether the master is optimized (simplified) according to AHB LITE protocol (True) or the master is full AHB (False). For more information on AHB Lite, refer to the "Functional Description" chapter in the DesignWare DW_ahb Databook. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** H2H_IS_LITE
Select AHB Interface Type	This parameter selects the AHB interface type. **Values:** ■ AHB (0) ■ AHB5 (1) **Default Value:** AHB **Enabled:** H2H_IS_LITE && DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_AHB_INTERFACE_TYPE
SPLIT capable slave	Generates SPLIT response and SPLIT clear from the slave interface. When this parameter is set to True, the slave interface is split-capable; otherwise, the slave interface is not split-capable. **Values:** ■ false (0) ■ true (1) **Default Value:** (H2H_IS_LITE== 1) ? 0 : 1 **Enabled:** H2H_IS_LITE == 0 **Parameter Name:** H2H_IS_SPLIT_CAPABLE
Time-tick sensitive	Specifies if the master interface is sensitive to the time tick (mTTICK) input signal, which acts like a timeout while the master interface is attempting to gain ownership of the bus to perform a transfer. When the timeout expires, the slave interface responds to the transfer with SPLIT. **Values:** ■ false (0) ■ true (1) **Default Value:** (H2H_IS_LITE== 1) ? 0 : 1 **Enabled:** H2H_IS_LITE != 1 **Parameter Name:** H2H_IS_TTICK_SENSITIVE

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Master ID Signal Width	The parameter specifies the width of a non standard Master ID sideband signals. When set to 0, the Master ID sideband signals are removed. In AHB5 mode, Master ID transmitted as hmaster signals. **Values:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 0 **Enabled:** H2H_IS_LITE==1 **Parameter Name:** H2H_MID_WIDTH
HRESP width 1-bit?	This parameter selects the width of HRESP port. ■ False : HRESP port width is 2-bit. ■ True : HRESP port width is 1-bit (AMBA AHB-Lite protocol). **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** H2H_IS_LITE==1 **Parameter Name:** H2H_SCALAR_HRESP
AHB Slave interface Configuration	
Address bus bit-width	Specifies the address bus width for the DW_ahb_h2h slave interface. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** H2H_PHY_SADDR_WIDTH
Data bus bit-width	Specifies the width of the DW_ahb_h2h slave interface data bus. **Values:** ■ 32 (32) ■ 64 (64) ■ 128 (128) ■ 256 (256) **Default Value:** 32 **Enabled:** Always **Parameter Name:** H2H_PHY_SDATA_WIDTH

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Data bus endianness	Specifies the data bus endianness of the primary AHB system to which the bridge is attached as an AHB slave. ■ AHB5 : Little-Endian, (BE-8) Big Endian and (BE-32) Big Endian are supported. ■ non-AHB5 : Little-Endian and Big-Endian (AMBA 2 AHB) are supported. **Values:** ■ Little-Endian (0) ■ Big-Endian (AMBA 2 AHB) (1) ■ (BE-8) Big Endian (2) ■ (BE-32) Big Endian (3) **Default Value:** Little-Endian **Enabled:** DWC-AMBA-AHB5-Fabric-Source License required to enable AHB5 Endian schemes **Parameter Name:** H2H_PHY_SBIG_ENDIAN
AHB Master interface Configuration	
Address bus bit-width	Specifies the address bus width of the DW_ahb_h2h master interface. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** H2H_PHY_MADDR_WIDTH
Data bus bit-width	Specifies the width of the DW_ahb_h2h master interface data bus. **Values:** ■ 32 (32) ■ 64 (64) ■ 128 (128) ■ 256 (256) **Default Value:** 32 **Enabled:** Always **Parameter Name:** H2H_PHY_MDATA_WIDTH

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Data bus endianness	Specifies the Data bus endianness of the secondary AHB system to which the bridge is attached as an AHB master. ■ AHB5 : Little-Endian, (BE-8) Big Endian and (BE-32) Big Endian are supported. ■ non-AHB5 : Little-Endian and Big-Endian (AMBA 2 AHB) are supported. **Values:** ■ Little-Endian (0) ■ Big-Endian (AMBA 2 AHB) (1) ■ (BE-8) Big Endian (2) ■ (BE-32) Big Endian (3) **Default Value:** Little-Endian **Enabled:** DWC-AMBA-AHB5-Fabric-Source License required to enable AHB5 Endian schemes **Parameter Name:** H2H_PHY_MBIG_ENDIAN
address bits 63 downto 32	Specifies the value driven on mHADDR[63:32] when the slave address bus width is narrower than that of the master. **Values:** 0x00000000, ..., 0xffffffff **Default Value:** 0x00000000 **Enabled:** H2H_PHY_MADDR_WIDTH > H2H_PHY_SADDR_WIDTH **Parameter Name:** H2H_PHY_MADDR_HIBIT

Table 3-1 Basic Configuration Parameters (Continued)

Label	Description
Clocking Configuration	
Clocking Mode	Specifies the clock mode. ■ 0: Asynchronous mode. Clocks are considered asynchronous. The slave BIU is clocked by sHCLK and reset by sHRESETn and the master BIU is clocked by mHCLK and reset by mHRESETn. The clock enable inputs are ignored. ■ 1: Synchronous mode, single clock, and clock enable (sHCLK, sHCLKEN). Both the master and the slave BIUs are clocked by sHCLK. The master BIU uses the sHCLKEN input signal as a clock enable. ■ 2: Synchronous mode, single clock, and clock enable (mHCLK, mHCLKEN). Both the master and the slave BIUs are clocked by mHCLK. The slave BIU uses the mHCLKEN input signal as a clock enable. ■ 3: Synchronous mode, dual clock, and no clock enables (sHCLK [faster clock], mHCLK). Both sHCLK and mHCLK are synchronous clocks but sHCLK is considered to be the faster clock. The master BIU is clocked by mHCLK. The slave BIU is clocked by sHCLK. ■ 4: Synchronous mode, dual clock, and no clock enables (sHCLK, mHCLK [faster clock]). Both sHCLK and mHCLK are synchronous clocks, but mHCLK is considered to be the faster clock. The master BIU is clocked by mHCLK. The slave BIU is clocked by sHCLK. **Values:** ■ asynchronous (0) ■ synchronous shclk shclken (1) ■ synchronous mhclk mhclken (2) ■ synchronous shclk (faster) mhclk (3) ■ synchronous mhclk (faster) shclk (4) **Default Value:** asynchronous **Enabled:** Always **Parameter Name:** H2H_CLK_MODE
Clock Domain Crossing Synchronization Depth?	This parameter determines the number of stages in the synchronizers when the clocking mode is asynchronous (H2H_CLK_MODE=0). For other clocking modes this parameter has no effect. **Values:** ■ 2-stage synchronization with both stages positive-edge capturing (2) ■ 3-stage synchronization with all stages positive-edge capturing (3) ■ 4-stage synchronization with all stages positive-edge capturing (4) **Default Value:** 2-stage synchronization with both stages positive-edge capturing **Enabled:** H2H_CLK_MODE==0 **Parameter Name:** H2H_NUM_SYNC_FLOPS

3.2 AHB5 Configuration Parameters

Table 3-2 AHB5 Configuration Parameters

Label	Description
AHB5 Configuration	
Include AHB5 Extended Memory Types Property?	Select this parameter to include Extended Memory Types property in DW_ahb_h2h. When set to 1, the width of hprot is increased from 4 to 7 and extended memory types information is passed through DW_ahb_h2h. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** H2H_AHB_INTERFACE_TYPE==1 && H2H_IS_PROT_CAPABLE==1 **Parameter Name:** H2H_AHB_EXTD_MEMTYPE
Include AHB5 Secure Transfers Property?	Select this parameter to include AHB5 Secure Transfers Property in DW_ahb_h2h. When set to 1, DW_ahb_h2h adds hnonsec signal to its interface to support this property. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** H2H_AHB_INTERFACE_TYPE==1 **Parameter Name:** H2H_AHB_SECURE
Include AHB5 Exclusive Transfers Property?	Select this parameter to include AHB5 exclusive transfers property in DW_ahb_h2h. When set to 1, DW_ahb_h2h adds hexcl and hexokay signals to its master and slave interface to support this property. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** H2H_AHB_INTERFACE_TYPE==1 **Parameter Name:** H2H_AHB_EXCLUSIVE

3.3 User signal Configuration Parameters

Table 3-3 User signal Configuration Parameters

Label	Description
User signal Configuration	
Select Data Channel User Signal Transfer Mode	This parameter selects whether the data channel user signals are to be transported as pass through or aligned to data. **Values:** ■ Pass Through (0) ■ Aligned to data (1) **Default Value:** Pass Through **Enabled:** Always **Parameter Name:** USER_SIGNAL_XFER_MODE
Number of User Signal bits per Data Byte for Write channel	This parameter specifies the number of user signal bits corresponding to each byte of Write data bus. **Values:** 0, ..., 8 **Default Value:** 0 **Enabled:** USER_SIGNAL_XFER_MODE==1 **Parameter Name:** WUSER_BITS_PER_BYTE
Number of User Signal bits per Data Byte for Read channel	This parameter specifies the number of user signal bits corresponding to each byte of Read data bus. **Values:** 0, ..., 8 **Default Value:** 0 **Enabled:** USER_SIGNAL_XFER_MODE==1 **Parameter Name:** RUSER_BITS_PER_BYTE
Width of Address Channel User Bus	This parameter specifies the width of address channel user signal bus. When set to 0, the address channel user signals are removed from the interface. **Values:** 0, ..., 256 **Default Value:** 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_HAUSER_WIDTH
Read Channel	
Width of Read Data Channel User Bus for Slave interface	This parameter specifies the width of Read data channel user signal bus for slave interface. When set to 0, the read data channel user signals are removed from the interface. **Values:** 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_SDATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 256 **Default Value:** (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_SDATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_SHRUSER_WIDTH

Table 3-3 User signal Configuration Parameters (Continued)

Label	Description
Width of Read Data Channel User Bus for Master interface	This parameter specifies the width of Read data channel user signal bus for master interface. When set to 0, the read data channel user signals are removed from the interface. **Values:** 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_MDATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 256 **Default Value:** (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_MDATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : H2H_SHRUSER_WIDTH **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_MHRUSER_WIDTH
Write Channel	
Width of Write Data Channel User Bus for Slave interface	This parameter specifies the width of Write data channel user signal bus for slave interface. When set to 0, the write data channel user signals are removed from the interface. **Values:** 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_SDATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 256 **Default Value:** (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_SDATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_SHWUSER_WIDTH
Width of Write Data Channel User Bus for Master interface	This parameter specifies the width of Write data channel user signal bus for master interface. When set to 0, the write data channel user signals are removed from the interface. **Values:** 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_MDATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 256 **Default Value:** (USER_SIGNAL_XFER_MODE==1) ? ((H2H_PHY_MDATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : H2H_SHWUSER_WIDTH **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** H2H_MHWUSER_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:

- Primary Interface on page 61
- Secondary Interface on page 66
- AHB5 on page 71
- Component ID Selection on sHRDATA on page 73
- Timeout Control Interface on page 74
- Synchronizer Bypass on page 75

4.1 Primary Interface Signals

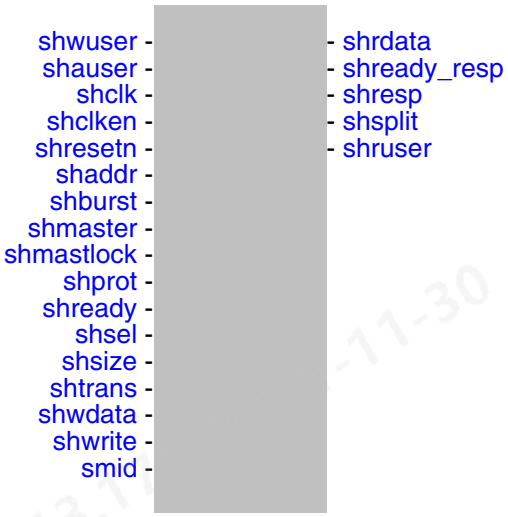


Table 4-1 Primary Interface Signals

Port Name	I/O	Description
shrdata[(SDW-1):0]	O	Slave port read data. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shready_resp	O	Current data phase complete. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
shresp[(H2H_HRESP_WIDTH-1):0]	O	Slave port response control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** (H2H_SCALAR_HRESP==1) ? "Yes" : (H2H_IS_LITE==1) && (!(H2H_IS_SPLIT_CAPABLE==1)) ? "0:0=Yes;1:1=No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 Primary Interface Signals (Continued)

Port Name	I/O	Description
shsplit[15:0]	O	Split clear indication for the arbiter. **Exists:** Always **Synchronous To:** (H2H_IS_SPLIT_CAPABLE==1) && (H2H_IS_TTICK_SENSITIVE == 1) ? "shclk" : "None" **Registered:** (H2H_IS_SPLIT_CAPABLE==1) && (H2H_IS_TTICK_SENSITIVE == 1) ? "0:0=No;15:1=Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
shruser[(H2H_SHRUSER_WIDTH-1):0]	O	Slave port read data user signal. **Exists:** H2H_SHRUSER_WIDTH>0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shwuser[(H2H_SHWUSER_WIDTH-1):0]	I	Slave port write data user signal. **Exists:** H2H_SHWUSER_WIDTH>0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** (USER_SIGNAL_XFER_MODE==0) ? "Yes" : (H2H_PHY_SDATA_WIDTH > H2H_PHY_MDATA_WIDTH) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shauser[(H2H_HAUSER_WIDTH-1):0]	I	Slave port address user signal. **Exists:** H2H_HAUSER_WIDTH>0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shclk	I	Slave port clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shclken	I	Slave port clock enable. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE==1) ? "shclk" : "None" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 Primary Interface Signals (Continued)

Port Name	I/O	Description
shresetn	I	Slave port reset. Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of slave port clock. DW_ahb_h2h does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
shaddr[(SAW-1):0]	I	Slave port address. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shburst[2:0]	I	Slave port burst control. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shmaster[(H2H_HMASTER_WIDTH-1):0]	I	Address bus owning master on primary side. **Exists:** (H2H_AHB_INTERFACE_TYPE==1 && H2H_MID_WIDTH!=0) || ((H2H_IS_SPLIT_CAPABLE == 1) && (H2H_IS_TTICK_SENSITIVE == 1)) **Synchronous To:** (H2H_AHB_INTERFACE_TYPE==1 && H2H_MID_WIDTH!=0) || ((H2H_IS_SPLIT_CAPABLE == 1) && (H2H_IS_TTICK_SENSITIVE == 1)) ? (H2H_CLK_MODE==2 ? "mhclk" : "shclk") : "None" **Registered:** (H2H_AHB_INTERFACE_TYPE==1 && H2H_MID_WIDTH!=0) || ((H2H_IS_SPLIT_CAPABLE == 1) && (H2H_IS_TTICK_SENSITIVE == 1)) ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shmastlock	I	Slave port lock control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 Primary Interface Signals (Continued)

Port Name	I/O	Description
shprot[(H2H_HPROT_WIDTH-1):0]	I	Slave port protection control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shready	I	Slave port previous data phase complete. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shsel	I	Slave select. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shsize[2:0]	I	Slave port size control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shtrans[1:0]	I	Slave port transfer control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shwdata[(SDW-1):0]	I	Slave port write data. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** H2H_PHY_SDATA_WIDTH > H2H_PHY_MDATA_WIDTH ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 Primary Interface Signals (Continued)

Port Name	I/O	Description
shwrite	I	Slave port direction control. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
smid[(H2H_MID_WIDTH-1):0]	I	Optional. Non standard Master ID sideband signal input(in AHB-Lite mode). **Exists:** (H2H_AHB_INTERFACE_TYPE==0 && H2H_MID_WIDTH!=0) **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Secondary Interface Signals

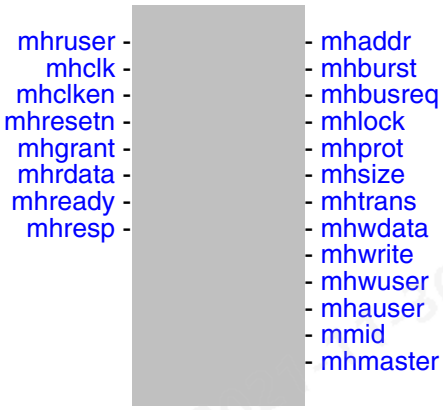


Table 4-2 Secondary Interface Signals

Port Name	I/O	Description
mhaddr[(MAW-1):0]	O	Master port address. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE==1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhburst[2:0]	O	Master port burst control(Connected directly to 1 or 0). **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhbusreq	O	Master port bus request status. **Exists:** Always **Synchronous To:** (H2H_IS_LITE==0) ? (H2H_CLK_MODE==1 ? "shclk" : "mhclk") : "None" **Registered:** H2H_IS_LITE==0 ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2 Secondary Interface Signals (Continued)

Port Name	I/O	Description
mhlock	O	Master port lock control. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhprot[(H2H_HPROT_WIDTH-1):0]	O	Master port proctection control. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhsize[2:0]	O	Master port size control. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhtrans[1:0]	O	Master port transfer control. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** 0:0=No;1:1=Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwdata[(MDW-1):0]	O	Master port write data. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwrite	O	Master port direction control(in asynchronous mode configuration when H2H_CLK_MODE = 0). **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2 Secondary Interface Signals (Continued)

Port Name	I/O	Description
mhruser[(H2H_MHRUSER_WIDTH-1):0]	I	Master port read data user signal. **Exists:** H2H_MHRUSER_WIDTH>0 **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** (USER_SIGNAL_XFER_MODE==0)? "Yes" : (H2H_PHY_MDATA_WIDTH > H2H_PHY_SDATA_WIDTH) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwuser[(H2H_MHWUSER_WIDTH-1):0]	O	Master port write data user signal. **Exists:** H2H_MHWUSER_WIDTH>0 **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhauser[(H2H_HAUSER_WIDTH-1):0]	O	Master port address channel user signal. **Exists:** H2H_HAUSER_WIDTH>0 **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mmid[(H2H_MID_WIDTH-1):0]	O	Optional. Non standard Master ID sideband signal output (in AHB-Lite mode). **Exists:** (H2H_AHB_INTERFACE_TYPE==0 && H2H_MID_WIDTH!=0) **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhmaster[(H2H_HMASTER_WIDTH-1):0]	O	Address bus owning master information on secondary side. **Exists:** (H2H_AHB_INTERFACE_TYPE==1 && H2H_MID_WIDTH!=0) **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Secondary Interface Signals (Continued)

Port Name	I/O	Description
mhclk	I	Master port clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhclken	I	Master clock Enable. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE==2) ? "mhclk" : "None" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhresetn	I	Master port reset. Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of master port clock. DW_ahb_h2h does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
mhgrant	I	Bus grant status. **Exists:** Always **Synchronous To:** (H2H_IS_LITE==1) ? "None" : (H2H_CLK_MODE==1 ? "shclk" : "mhclk") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhrdata[(MDW-1):0]	I	Master port read data. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** H2H_PHY_MDATA_WIDTH > H2H_PHY_SDATA_WIDTH ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Secondary Interface Signals (Continued)

Port Name	I/O	Description
mhready	I	Master port data phase complete. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhresp[(H2H_HRESP_WIDTH-1):0]	I	Master port response control. **Exists:** Always **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 AHB5 Signals

Table 4-3 AHB5 Signals

Port Name	I/O	Description
mhnonsec	O	Master port secure transfer signal. **Exists:** H2H_AHB_SECURE!=0 **Synchronous To:** (H2H_CLK_MODE == 1) ? "shclk" : "mhclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhexcl	O	Master port exclusive transfer signal. **Exists:** H2H_AHB_EXCLUSIVE!=0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shexokay	O	Slave port exclusive transfer response signal. **Exists:** H2H_AHB_EXCLUSIVE!=0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhexokay	I	Master port exclusive transfer response signal. **Exists:** H2H_AHB_EXCLUSIVE!=0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
shnonsec	I	Slave port secure transfer signal. **Exists:** H2H_AHB_SECURE!=0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3 AHB5 Signals (Continued)

Port Name	I/O	Description
shexcl	I	Slave port exclusive transfer signal. **Exists:** H2H_AHB_EXCLUSIVE!=0 **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 Component ID Selection on sHRDATA Signals

Table 4-4 Component ID Selection on sHRDATA Signals

Port Name	I/O	Description
bus_id[(H2H_ID_WIDTH-1):0]	O	Component ID code. This signal is driven with the VERSION_ID value. To determine this value, refer to the DW_ahb_h2h releases table in the "AMBA 2 Release Notes". **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shsel_id	I	When 1, the content of the HRDATA internal register is made available on shrdata signals. After reset, the register is defaulted to the bridge component ID code. **Exists:** Always **Synchronous To:** H2H_CLK_MODE==2 ? "mhclk" : "shclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Timeout Control Interface Signals

mttick -

Table 4-5 Timeout Control Interface Signals

Port Name	I/O	Description
mttick	I	Periodic low-frequency signal in the mHCLK clock domain used for timeout SPLIT or ERROR response generation. **Exists:** Always **Synchronous To:** (H2H_IS_TTICK_SENSITIVE==1) ? ((H2H_CLK_MODE==1) ? "shclk" :"mhclk") : "None" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.6 Synchronizer Bypass Signals

sync_bypass -

Table 4-6 Synchronizer Bypass Signals

Port Name	I/O	Description
sync_bypass	I	When 1, internal synchronization stages are bypassed. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Verification

This chapter provides an overview of the testbench available for the DW_ahb_h2h verification. After the DW_ahb_h2h has been configured and the verification setup, simulations can be run automatically. For information on running simulations for DW_ahb_h2h in coreAssembler or coreConsultant, see the "Simulating the Core" section in the *DesignWare Synthesizable Components for AMBA 2 User Guide*.

Note The DW_ahb_h2h verification testbench is built with DesignWare Verification IP (VIP). Make sure 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 *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

Note The packaged test benches are only for validating the IP configuration in coreConsultant GUI. It is not for system level validation.
IPs that have the Vera test bench packaged, these test benches are encrypted.

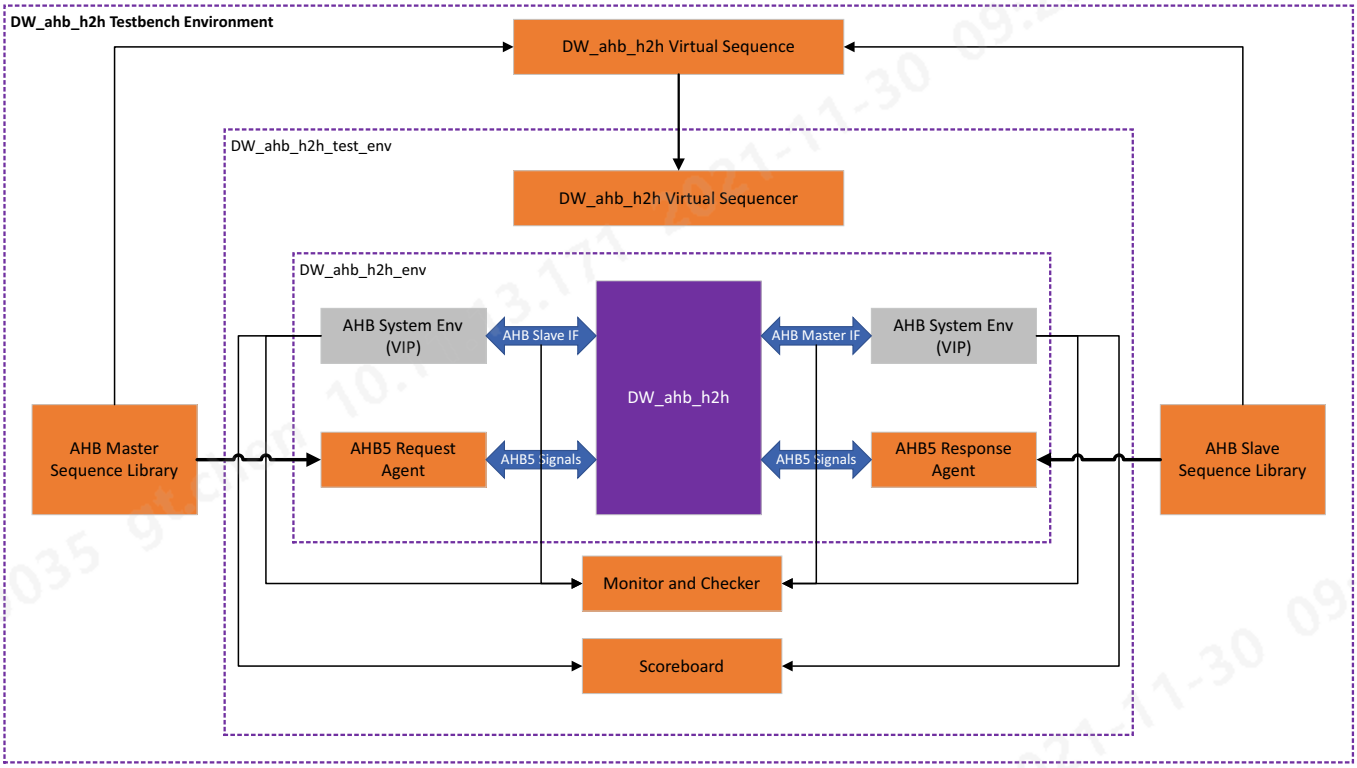
This chapter discusses the following sections:

5.1 Verification Environment

DW_ahb_h2h is verified using a UVM-methodology-based constrained random verification environment. The environment is capable of generating random scenarios and the test case has hooks to control the scenarios to be generated.

Figure 5-1 shows the verification environment of the DW_ahb_h2h testbench:

Figure 5-1 DW_ahb_h2h UVM Verification Environment



The testbench consists of the following elements:

- Testbench uses the standard SVT VIP for the protocol interfaces:
  - AMBA SVT VIP
    - AHB VIP Interface for connecting master and slave ports of DUT.
    - AHB system environment (svt_ahb_system_env): AHB master agent and AHB slave agent for providing the VIP supported randomized transactions and a bus structure instantiated on either side of DUT.
- Testbench uses the custom components:
  - AHB5 request agent

    This component includes driver and sequencer for ahb5 request signals. This has been used to incorporate AHB5 specific signals such as the hexcl signal (Exclusive transfer property), which are not supported by the current VIP.

- AHB5 response agent

  This includes driver and sequencer for ahb5 response signals. This drives ahb5 specific response signals like hexokay (Exclusive transfer property) which are not supported by current AHB VIP.
- Master and slave interfaces for AHB5 specific signals to ensure proper AHB5 signal connectivity.
- AHB Master sequence library

  This sequence library contains a collection of sequences used by the AHB masters on primary port.
- AHB Slave sequence library

  This sequence library contains a collection of sequences used by the AHB slaves on secondary port.
- Monitors

  There are two custom monitors used for sampling data and data user signals from the interface and sent to the scoreboard via TLM ports for performing data integrity checks.
- Scoreboard

  This component is used for performing all data integrity checks and hence cross-verifying the correctness of the RTL implementation. The Scoreboard receives the relevant data corresponding to each transaction from the respective monitors (VIP and custom) from primary and secondary side. There are checkers related to Endianness, User-signals, secure transfer(AHB5), Extended memory types(AHB5), Exclusive access(AHB5) etc which employ comparative analysis to check if the value for the various attributes as set on the primary side are intact when observed on the secondary side.

5.2 Testbench Directories and Files

The DW_ahb_h2h verification environment contains the following directories and associated files.

Table 5-1 shows the various directories and associated files:

Table 5-1 DW_ahb_h2h Testbench Directory Structure

Directory	Description
<configured workspace>/sim/testbench	Contains the top level testbench module (test_top.sv) and 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/	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.

5.3 Packaged Testcases

The simulation environment that comes as a package files 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 test cases shipped, and their description is explained in Table 5-2.

Table 5-2 DW_ahb_h2h Test Description

Test Name	Test Description
test_100_h2h_random	This test verifies DUT functionality with random sequences. This test issues random sequence for all the Masters present in the environment on primary side of DUT and at the same time random response is provided for the slave present on the secondary side of the design. When the design is configured to have AHB5 interface, then additional sequence for secure, exclusive extended memory types starts along with the random stimulus.

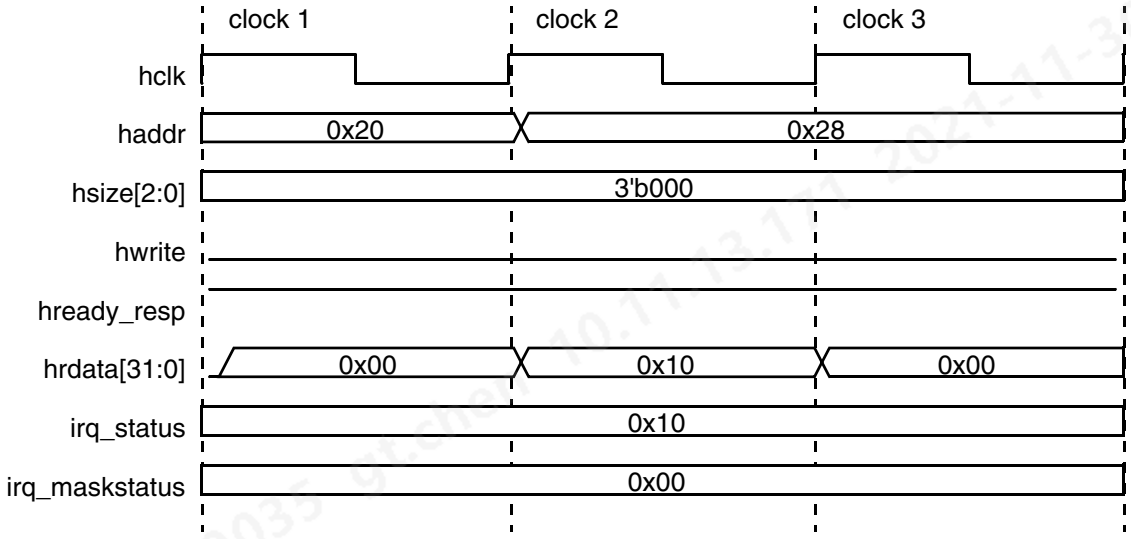
6 Integration Considerations

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

6.1 Read Accesses

For reads, registers less than the full access width return zeros in the unused upper bits. An AHB read takes two hclk cycles. The two cycles can be thought of as a control and data cycle, respectively. As shown in the Figure 6-1, the address and control is driven from clock 1 (control cycle); the read data for this access is driven by the slave interface onto the bus from clock 2 (data cycle) and is sampled by the master on clock 3. The operation of the AHB bus is pipelined, so while the read data from the first access is present on the bus for the master to sample, the control for the next access is present on the bus for the slave to sample.

Figure 6-1 AHB Read

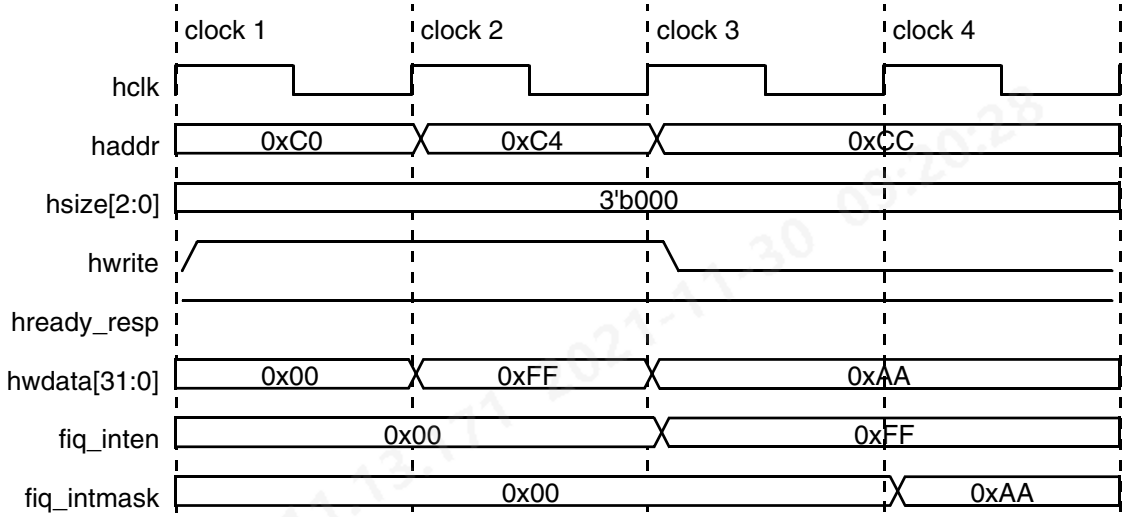


6.2 Write Accesses

When writing to a register, bit locations larger than the register width or allocation are ignored. Only pertinent bits are written to the register. Similar to the read case, a write access may be thought of as comprising a control and data cycle. As illustrated in Figure 6-2, the address and control is driven from

clock1 (control cycle), and the write data is driven by the bus from clock 2 (data cycle) and sampled by the destination register on clock 3.

Figure 6-2 AHB Write

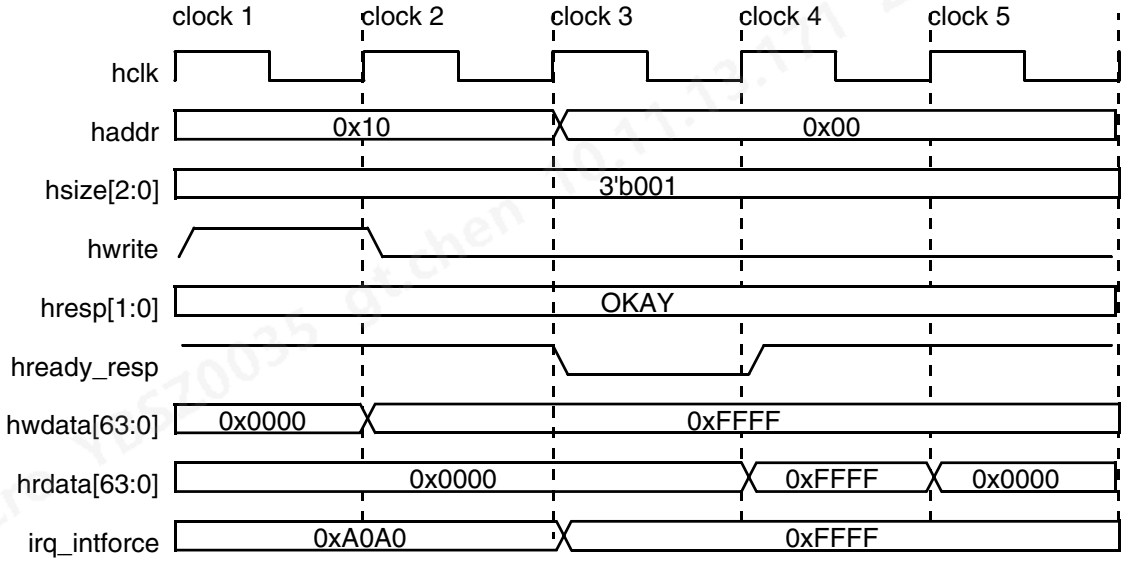


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

6.3 Consecutive Write-Read

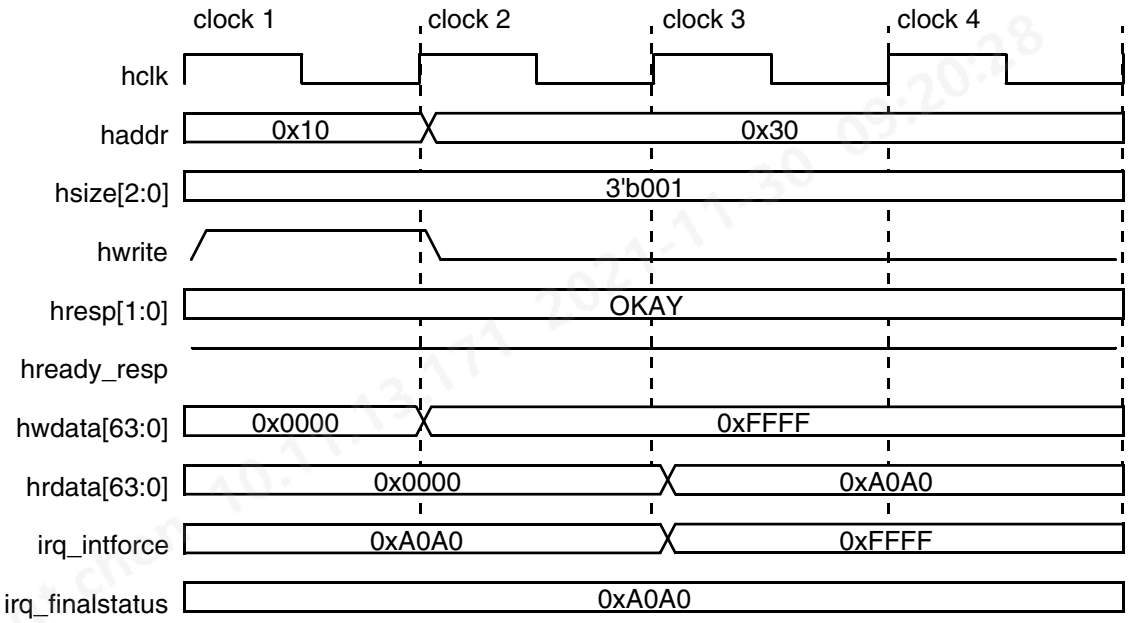
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_h2h inserts a “wait state” when it detects a read immediately after a write to the same address. As shown in Figure 6-3, the control for a write is driven on clock 1, followed by the write data and the control for a read from the same address on clock 2.

Figure 6-3 AHB Wait State Read/Write



Sensing the read after a write to the same address, the slave interface drives hready_resp low from clock 3; hready_resp is driven high on clock 4 when the new write data can be read; and the bus samples hready_resp high on clock 5 and reads the newly written data. Figure 6-4 shows a normal consecutive write-read access.

Figure 6-4 AHB Consecutive Read/Write



6.4 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>
   ```

6.5 Timing Exceptions

- For details on multi cycle paths, see the DW_ahb_h2h.sdc file generated by the Design Compiler.
- For details on quasi-static signals on the design, refer to manual.sgdc report generated by the SpyGlass tool.

6.6 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_ahb_h2h.

6.6.1 Power Consumption, Frequency, Area and DFT Coverage

Table 6-1 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_ahb_h2h using the industry standard 7nm technology library.

Table 6-1 Synthesis Results for DW_ahb_h2h

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	shclk=300.3003 MHz mhclk=300.3003 MHz	1913	5 nW	0.044 mW	100	99.03	99.5

Table 6-1 Synthesis Results for DW_ahb_h2h (Continued)

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		
			Static Power	Dynamic Power	StuckAtTest	Transition	SpyGlass StuckAtCov(%)
Typical Configuration - 1 H2H_IS_LITE = 1 H2H_AHB_INTERFACE_TYPE = 1 H2H_CLK_MODE = 3 H2H_PHY_SDATA_WIDTH = 256 H2H_PHY_MDATA_WIDTH = 32 H2H_PHY_SADDR_WIDTH = 64 H2H_PHY_MADDR_WIDTH = 64 H2H_PHY_SBIG_ENDIAN = 3 H2H_PHY_MBIG_ENDIAN = 2 H2H_AHB_EXTD_MEMTYPE = 1 H2H_AHB_SECURE = 1 H2H_AHB_EXCLUSIVE = 1 USER_SIGNAL_XFER_MODE = 1 RUSER_BITS_PER_BYTE = 5 WUSER_BITS_PER_BYTE = 5 H2H_HAUSER_WIDTH = 8 H2H_MID_WIDTH = 4	shclk=300.3003 MHz mhclk=300.3003 MHz	7740	20 nW	0.110 mW	100	99.8	99.9

Table 6-1 Synthesis Results for DW_ahb_h2h (Continued)

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Typical Configuration - 2 H2H_PHY_SADDR_WIDTH = 32 H2H_PHY_MADDR_WIDTH = 32 H2H_PHY_SDATA_WIDTH = 32 H2H_PHY_MDATA_WIDTH = 32 H2H_CLK_MODE = 0 H2H_PHY_SBIG_ENDIAN = 0 H2H_PHY_MBIG_ENDIAN = 0 H2H_IS_SPLIT_CAPABLE = 1 H2H_IS_TTICK_SENSITIVE = 1 H2H_IS_LOCK_CAPABLE = 0	shclk=300.3003 MHz mhclk=300.3003 MHz	1783	5 nW	0.034 mW	100	99.75	99.4

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 appendix contains the following sections:

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_ahb_h2h.

Table A-1 BCM Library Components

BCM Module Name	BCM Description	DWBB Equivalent
DW_ahb_h2h_bcm21	Single clock data bus synchronizer	DW_sync
DW_ahb_h2h_bcm02	Universal Multiplexer	DW01_mux_any

A.2 Synchronizer Methods

This appendix describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_ahb_h2h to cross clock boundaries.

This appendix contains the following sections:

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_ahb_h2h

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_ahb_h2h 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_ahb_h2h

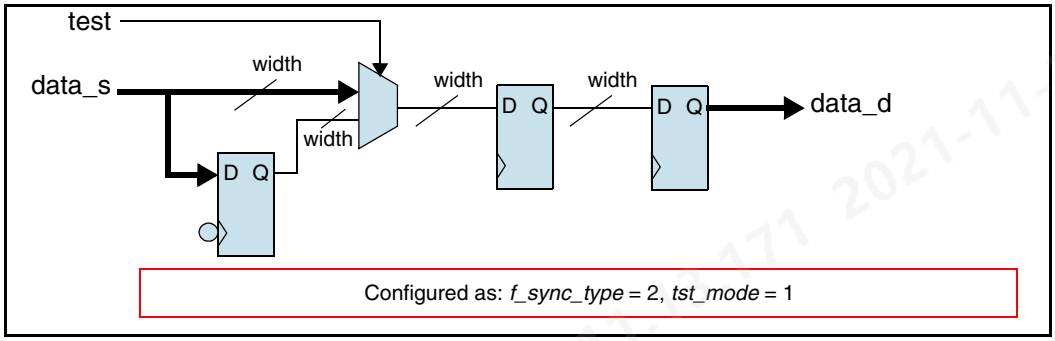
Synchronizer Module File	Synchronizer Type and Number
DW_ahb_h2h_bcm21.v	Synchronizer 1: Simple Multiple Register Synchronizer

Note The BCM21 is a basic multiple-register-based synchronizer module used in the design. It can be replaced with equivalent technology-specific synchronizer cell.

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_ahb_h2h)

This is a single clock data bus synchronizer for synchronizing control signals that crosses asynchronous clock boundaries. The synchronization scheme uses 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)



B
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 B-1 Internal Parameters

Parameter Name	Equals To
ERROR	2'b01
H2H_HMASTER_WIDTH	{(H2H_AHB_INTERFACE_TYPE ==1 && H2H_MID_WIDTH ! =0) ? H2H_MID_WIDTH : 4}
H2H_HPROT_WIDTH	{(H2H_AHB_EXTD_MEMTYPE == 0) ? 4 : 7}
H2H_HRESP_WIDTH	= ((H2H_SCALAR_HRESP ==1) ? 1:2)
H2H_ID_WIDTH	32
H2H_IS_PROT_CAPABLE	1
MAW	H2H_PHY_MADDR_WIDTH
MDW	H2H_PHY_MDATA_WIDTH
SAW	H2H_PHY_SADDR_WIDTH
SDW	H2H_PHY_SDATA_WIDTH

C
Comparison of Endian Schemes

This appendix compares different endian schemes that are used in DW_ahb_h2h.

Figure C-1 diagram describes the comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) endian schemes for a byte access.

Figure C-1 Comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) for Byte Access

Little Endian (LE)

transfer_addr	0x1000			
actual_addr				0x1000
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data				D0

transfer_addr	0x1001			
actual_addr			0x1001	
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D1	

transfer_addr	0x1002			
actual_addr		0x1002		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data		D2		

transfer_addr	0x1003			
actual_addr	0x1003			
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D3			

Big Endian (BE8)

transfer_addr	0x1000			
actual_addr				0x1000
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data				D0

transfer_addr	0x1001			
actual_addr			0x1001	
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D1	

transfer_addr	0x1002			
actual_addr		0x1002		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data		D2		

transfer_addr	0x1003			
actual_addr	0x1003			
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D3			

Big Endian (BE32)

transfer_addr	0x1000			
actual_addr	0x1000			
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D0			

transfer_addr	0x1001			
actual_addr		0x1001		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data		D1		

transfer_addr	0x1002			
actual_addr			0x1002	
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D2	

transfer_addr	0x1003			
actual_addr				0x1003
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data				D3

Big Endian (AMBA 2 AHB)

transfer_addr	0x1000			
actual_addr	0x1000			
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D0			

transfer_addr	0x1001			
actual_addr		0x1001		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data		D1		

transfer_addr	0x1002			
actual_addr			0x1002	
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D2	

transfer_addr	0x1003			
actual_addr				0x1003
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data				D3

Figure C-2 diagram describes the comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) endian schemes for a half word access.

Figure C-2 Comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) for Half-word Access

Little Endian (LE)

transfer_addr	0x1000			
actual_addr			0x1001	0x1000
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D1	D0

transfer_addr	0x1002			
actual_addr	0x1003	0x1002		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D3	D2		

Big Endian (BE8)

transfer_addr	0x1000			
actual_addr			0x1000	0x1001
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D0	D1

transfer_addr	0x1002			
actual_addr	0x1002	0x1003		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D2	D3		

Big Endian (BE32)

transfer_addr	0x1000			
actual_addr	0x1001	0x1000		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D1	D0		

transfer_addr	0x1002			
actual_addr			0x1003	0x1002
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D3	D2

Big Endian (AMBA 2 AHB)

transfer_addr	0x1000			
actual_addr	0x1000	0x1001		
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D0	D1		

transfer_addr	0x1002			
actual_addr			0x1002	0x1003
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data			D2	D3

Figure C-3 diagram describes the comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) endian schemes for a word access.

Figure C-3 Comparison of LE, BE8, BE32 and BE (AMBA 2 AHB) for Word Access

Little Endian (LE)

transfer_addr	0x1000			
actual_addr	0x1003	0x1002	0x1001	0x1000
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D3	D2	D1	D0

Big Endian (BE8)

transfer_addr	0x1000			
actual_addr	0x1000	0x1001	0x1002	0x1003
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D0	D1	D2	D3

Big Endian (BE32)

transfer_addr	0x1000			
actual_addr	0x1003	0x1002	0x1001	0x1000
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D3	D2	D1	D0

Big Endian (AMBA 2 AHB)

transfer_addr	0x1000			
actual_addr	0x1000	0x1001	0x1002	0x1003
data_bus_lanes	[31:24]	[23:16]	[15:8]	[7:0]
data	D0	D1	D2	D3

D
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 DesignWare Synthesizable Components.
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.
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.