SYNOPSYS®

DesignWare DW_ahb_eh2h Databook

DW_ahb_eh2h – Product Code

1.11a
July 2018

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.

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.
690 E. Middlefield Road
Mountain View, CA 94043

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	July 2018	**Updated:** ■ Version changed for 2018.07a release ■ Added MID Sideband signals: mmid, and smid ■ "Performance" on page 86 ■ "Parameter Descriptions" on page 39, Chapter 4, "Signal Descriptions", "Register Descriptions" on page 59, and Appendix B, "Internal Parameter Descriptions" are auto-extracted with change bars from the RTL **Removed:** ■ Parameter EH2H_RAM_SYNC has been deprecated. ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.
1.10a	October 2016	■ Version number changed for 2016.10a release ■ "Parameter Descriptions" on page 39 and "Register Descriptions" on page 59 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 an entry for the xprop directory in Table 2-1 and Table 2-4. ■ Added "Running VCS XPROP Analyzer"

(Continued)

Version	Date	Description
1.09a	June 2015	■ Added “Running SpyGlass® Lint and SpyGlass® CDC” ■ Added “Running SpyGlass® on Generated Code with coreAssembler” ■ Corrected completion of split response to a read (register space or external slave) after a register space read. ■ Updated “Deadlock Conditions” on page 21 to provide a solution to exit the deadlock scenario. ■ “Signal Descriptions” on page 47 auto-extracted from the RTL ■ Added “Internal Parameter Descriptions” on page 95 ■ Added Appendix A, “Synchronizer Methods”
1.08a	June 2014	■ Version change for 2014.06a release ■ Added “Performance” section in the “Integration Considerations” chapter
1.07f	May 2013	■ Version change for 2013.05a release ■ Updated the template
1.07e	Oct 2012	Added the product code on the cover and in Table 1-1
1.07e	Mar 2012	Corrected offset values for EWS and MEWS registers in RAL description
1.07d	Nov 2011	Version change for 2011.11a release
1.07c	Oct 2011	Version change for 2011.10a release
1.07b	Jun 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced “Related Documents” section in Preface
1.07b	May 2011	Corrected address offsets for EH2H_COMP_PARAM_1, EH2H_COMP_VERSION, and EH2H_COMP_TYPE registers
1.07b	Oct 2010	Version change for 2010.10a release
1.07a	Sep 2010	■ Added material about limitations with respect to defined length burst support ■ Corrected names of include files and vcs command used for simulation
1.06a	Dec 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.06a	Jul 2009	Corrected value and default of EH2H_RAM_SYNC parameter
1.06a	Jun 2009	Corrected name of mhbusreq signal in I/O diagram table
1.06a	May 2009	Removed references to QuickStarts, as they are no longer supported
1.06a	Oct 2008	■ Updated “Clock Adaptation” section ■ Version change for 2008.10a release
1.05b	Jun 2008	■ Version change for 2008.06a release ■ Added more detail about transfer changing from a SINGLE to an INCR

(Continued)

Version	Date	Description
1.05a	Feb 2008	Added more detail about cases where the DW_ahb_eh2h can change a SINGLE to an INCR
1.05a	Nov 2007	■ Added synchronization parameters in Clocking Configuration ■ Clarifications on bursts and upsizing
1.04b	Aug 7, 2007	■ Added Reset Signalssection ■ Corrected mhresetn synchronous information
1.04b	June 14, 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DesignWare enhanced AHB-to-AHB bridge component (DW_ahb_eh2h) to the Advanced High-Performance Bus (AHB). This component 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.

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_eh2h.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_ahb_eh2h.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_ahb_eh2h signals.
- Chapter 5, Register Descriptions describes the programmable registers of the DW_ahb_eh2h.
- Chapter 6, "Programming the DW_ahb_eh2h" provides information needed to program the configured DW_ahb_eh2h.
- Chapter 7, "Verification" provides information on verifying the configured DW_ahb_eh2h.
- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_ahb_eh2h into your design.
- Appendix A, "Synchronizer Methods"documents the synchronizer methods (blocks of synchronizer functionality) used in DW_ahb_eh2h to cross clock boundaries.
- Appendix B, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals, Registers and Parameters chapters.
- Appendix C, "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*.

Web Resources

- DesignWare IP product information: http://www.designware.com
- Your custom DesignWare IP page: http://www.mydesignware.com
- Documentation through SolvNet: http://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product:

- First, prepare the following debug information, if applicable:
  - For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.
  - 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
- Then, contact Support Center, with a description of your question and supplying the requested information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

    Go to http://solvnet.synopsys.com/EnterACall and click the **Open A Support Case** to enter a call.
    Provide the requested information, including:
    - **Product:** DesignWare Library IP
    - **Sub Product:** AMBA
    - **Tool Version:** <product version number>

- **Problem Type:**
- **Priority:**
- **Title:** DW_ahb_eh2h
- **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

After creating the case, attach any debug files you created in the previous step.

- 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 name, Sub Product name, and Tool Version number in your e-mail (as identified earlier) 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 in the previous step.
- 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

Component Name	Description
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
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

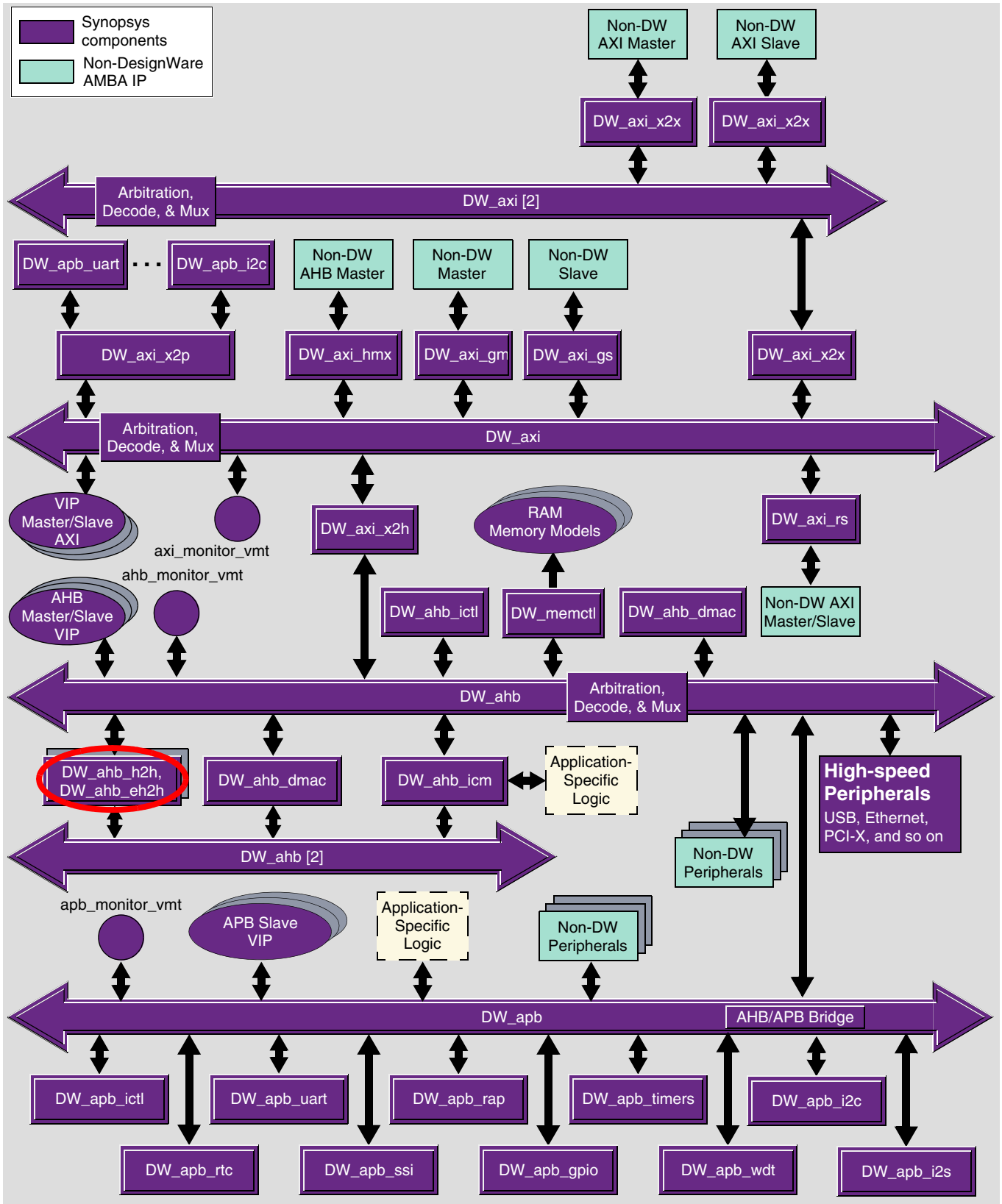
The DW_ahb_eh2h is an AHB-to-AHB bridge with a FIFO-based architecture, designed to achieve high throughput and high bus efficiency. This component is part of the DesignWare Synthesizable Components for AMBA 2.

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. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Figure 1-1 Example of DW_ahb_eh2h in a Complete System



Note

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

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

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

You can connect, configure, synthesize, and verify the DW_ahb_eh2h 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_eh2h component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

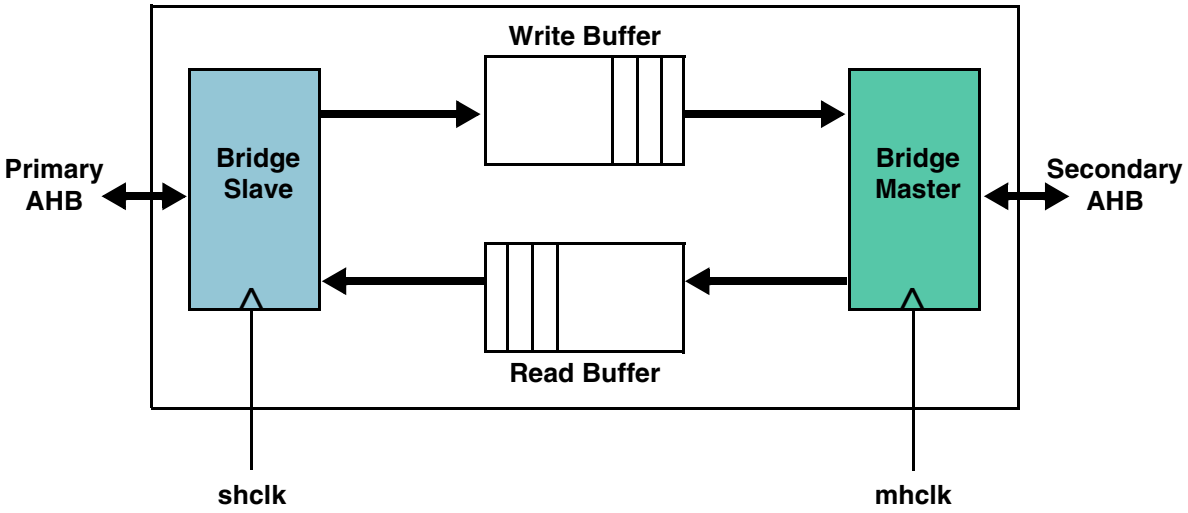
The Synopsys DW_ahb_eh2h consists of an AHB slave bus interface, a write buffer, a read buffer, and an AHB master bus interface. These components operate as follows:

1. The bridge slave accepts transfers (read/write requests, read/write addresses, and write data) from the primary AHB and sends them to the write buffer.
2. The write buffer transmits transfer attributes to the bridge master.
3. The bridge master executes received transfers into the secondary AHB.
4. The read buffer returns read data and read responses from the bridge master to the bridge slave.

1.2.1 DW_ahb_eh2h Block Diagram

The components of DW_ahb_eh2h are illustrated in Figure 1-2.

Figure 1-2 DW_ahb_eh2h Block Diagram



1.2.2 Functional Overview

The following topics provide overview information about the function of the DW_ahb_eh2h. For more information about the function and operation of the DW_ahb_eh2h, see "Functional Description" on page 25.

1.2.2.1 Write Transfers

Writes from the primary AHB to the bridge are "posted," meaning that the bridge slave responds with OKAY terminating the primary transfer before the write data actually reaches its final destination in the secondary AHB. Data is temporarily stored into local memory (the write buffer) until the bridge master is able to perform the corresponding bus transfer to the secondary AHB. If the buffer becomes full, the bridge slave splits the primary transfer. The split is cleared after the write buffer has been drained by the bridge master and is able to accept new data. The split mechanism allows the primary AHB to be free to handle other transfers.

1.2.2.2 Read Transfers

Reads from the primary AHB to the bridge are "split and prefetched". The bridge slave splits the first beat of a read burst. The read request is transmitted to the bridge master. Data is retrieved from the secondary AHB by the bridge master and temporarily stored into local memory (the read buffer). Once all burst data is available, the bridge slave clears the previously issued split response, signaling to the primary AHB that the read burst can be re-issued for completion.

1.2.2.3 Data Width Adaptation

The bridge can be used to connect AHB systems with arbitrary data widths. The primary AHB is allowed to perform full bandwidth[1] read and write accesses to the bridge slave independently of the width of the secondary AHB system. When the secondary data width is narrower than the primary data width, the

1. A full bandwidth transfer is a transfer with HSIZE = log2(bus width/8)

bridge master transforms wider primary transfers into narrower longer secondary bursts. This capability is called *transfer downsizing*.

When the secondary data width is wider than the primary data width, the bridge master does *not* transform narrower primary transfers into wider shorter secondary transfers. The HSIZE attribute for secondary transfers is limited by the width of the primary data bus. This feature, known as transfer upsizing, is not supported by the DW_ahb_eh2h bridge.

1.2.2.4 Software Interface and Interrupt

The bridge maintains two status registers to provide debug support and a minimal error recovery capability in case of write failure. A write failure occurs whenever a write transfer from the bridge master is terminated by an ERROR response. An interrupt line associated with write failure conditions is available at the pinout of the component. A third register allows software to acknowledge and reset the status registers and the interrupt signal.

1.2.2.5 Bus Performance

The bridge performance can be measured in terms of the number of bus clock cycles needed to execute a burst (or a sequence of bursts) of n consecutive data phases. For the bridge slave consider two contributes to the overall number of clock cycles: bus phase and split latency. The bus phase includes all the clock cycles consumed by the bridge slave responding to the accessing master. The split latency includes all of the clock cycles that last between a split response issued by the bridge and the retried transfer that follows after the bridge clears the split. For the bridge master, there is no split latency and only the bus phase is considered, which is defined as the number of clock cycles required for the master to generate n secondary data phases.

1.2.2.5.1 Bridge Slave Writes

When the write buffer has enough free locations to accept a full burst of length n, the split latency is zero and the bus phase lasts n+2 clock cycles (two wait states are inserted on the first NSEQ transfer). The average throughput on the bus phase is then n/n+2.

Bridge slave writes are affected by split latency when the write buffer operates close to the full condition. When a split is generated on a write transaction because the buffer is full, the latency depends on the capacity of the bridge master to drain the write buffer, which ultimately depends on the secondary system clock, data width, and the behavior of the secondary system in terms of wait states, responses, and arbitration.

Write performance deteriorates when the buffer depth is too short the secondary AHB is unavailable for long periods, or your write bandwidth is too large so that the write buffer is close to full or is full.

1.2.2.5.2 Bridge Master Writes

The bus phase lasts n clock cycles. Two idle cycles are required for arbitration; two idle cycles are required in between consecutive bursts. The average throughput is n/n+2.

1.2.2.5.3 Bridge Slave Reads

Bridge slave reads always are affected by split latency (assuming the write buffer is empty), consisting of:

- Synchronization delay to transmit the read request from the bridge slave clock domain to the bridge master clock domain (depends on the clock mode and pipe mode parameters, see "Parameter Descriptions" on page 39)
- Arbitration and bus phase of the bridge master, needed to prefetch read data
- Synchronization to transmit read completion back to the bridge slave (depends on the clock mode and pipe mode parameters, see "Parameter Descriptions" on page 39)
- Primary arbitration delay between the split clear generated by the bridge slave and the retried transfer

Split latency on reads is minimal when the clock mode parameter is set to synchronous and the pipe mode parameters are all set to 0 (see "Parameter Descriptions" on page 39).

The bus phase at the primary AHB lasts n+2 clock cycles (two extra cycles are needed for the initial split response). The average throughput on the primary bus phase is n/n+2

1.2.2.5.4 Bridge Master Reads

This has the same behavior as Bridge Master Writes.

1.2.2.5.5 Bus Decoupling

The write buffer helps to decouple the primary AHB from wait cycles, split/retry responses, and arbitration delay incurred when writing to the secondary AHB. The result is a peak throughput close to 1 seen by the primary master in write direction. The read buffer, in combination with a SPLIT response (issued by the bridge slave) and data prefetch (performed by the bridge master), allows for better bus utilization on reads. After a SPLIT response, the arbiter removes the master that is currently owning the bus so that the primary AHB can be used by other masters. The result of this mechanism is an improvement in the overall bus efficiency of the primary AHB, because only the originating master is affected by the latency of the read access. While the originating master is stalled by the SPLIT response, the primary AHB is fully available for other masters to operate.

1.2.2.5.6 Multi-Master Systems

Because both read and writes are non-blocking (writes are posted, reads are split), the bridge efficiently supports systems where multiple primary masters concurrently require access to the secondary bus. Writes from different masters are posted in the order in which they are received by the bridge slave, and executed in the same order by the bridge master. Reads are split, leaving the primary bus free for other masters to post a read or write access to the bridge slave or to any other primary slaves. Consecutive reads from different masters are executed by the bridge master in the same order in which they are received by the bridge slave.

1.2.2.6 Recommendations

You should take note of the following information regarding the DW_ahb_eh2h.

1.2.2.6.1 Locked Transfers

Read and write transfers are generally executed in the order in which they are received. The only exception is for locked transfers. Locked transfers are always completed before any other previously split read transfer

that is pending. When the bridge receives a locked transfer request and there are other reads (from other masters) still pending, the bridge aborts any operations for those reads and flushes the read buffer from any prefetched data to allow the locked transfer to complete first. Aborted transactions must be repeated after the locked transfer has completed.

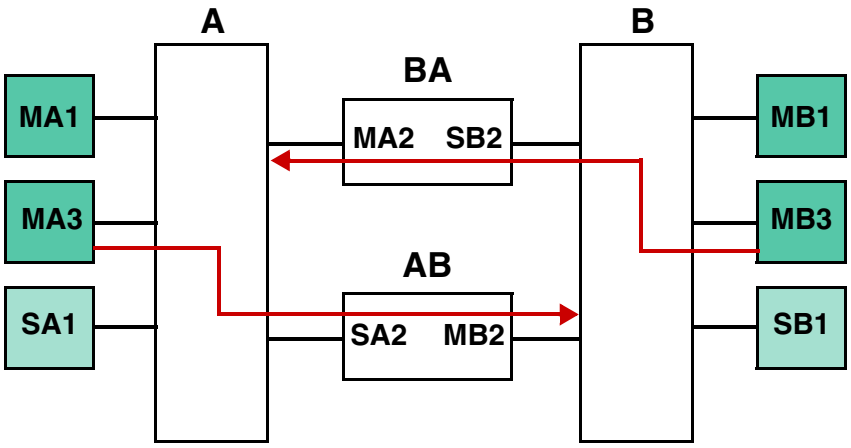
1.2.2.6.2 Read-Sensitive Locations

Care must be taken when accessing read-sensitive locations in the secondary AHB (for example, a device FIFO or a reset-on-read status register) through the bridge. There are situations (see "Read-Sensitive Locations" on page 29) where the bridge is forced to flush out the read buffer, which can cause data loss if the discarded buffer content is fetched from read-sensitive locations.

1.2.2.6.3 Deadlock Conditions

The split mechanism ensures adequate protection against deadlock conditions for bidirectional applications. Deadlock can still occur if locked transfer types are used in both directions. The following figure shows Master 3 in system A accessing the bridge AB with a locked transfer, while Master 3 in system B is performing a locked transfer to bridge BA. Both systems, A and B, are locked which results in an deadlock.

Figure 1-3 Deadlock Scenario for Simultaneous Locked Access



This deadlock scenario can be resolved by using the Alternative "HREADY Low" Response Mode with the timeout. Whenever locked transfers are requested, "HREADY Low" Response Mode can be enabled dynamically by qualifying the address phase with the sstall signal assertion, which inhibits the SPLIT response functionality. If the Master 3 (MA3) in system A and Master 3 (MB3) in system B simultaneously generate locked transfer as in Figure 1-3, then transfer cannot be completed as system A and B are already locked. Due to the existence of the timeout functionality (with "HREADY Low" Response Mode), an error is generated thus, bringing the system A and system B out of the deadlock. System A and system B can rebuild the locked transfer again.

1.3 Features

The following sections discuss the DW_ahb_eh2h features.

1.3.1 Clocks

- Asynchronous or synchronous clocks, any clock ratio
- Fully registered outputs
- Optional pipeline stages to reduce logic levels on bus inputs

1.3.2 Interfaces

The DW_ahb_eh2h has the following interfaces.

- AHB Slave
  - Data width: 32,64,128, or 256 bits
  - Address width: 32 or 64 bits
  - Big or little endian
  - Zero or two wait states OKAY response
  - ERROR response
  - No RETRY response
  - SPLIT response
  - HSPLIT generation
  - Handling of multiple, outstanding split transactions
  - Multiple HSELs
  - HREADY low (alternative to SPLIT response) operation mode
- AHB Master
  - Data width: 32,64,128, or 256 bits
  - Address width: 32 or 64 bits
  - Big or little endian
  - Lock and bus request generation
  - SINGLE, INCR burst type generation for writes
  - Any burst type generation for reads
  - Downsizing of wider transfers; note that upsizing is not supported

1.3.3 Operation

The DW_ahb_eh2h has the following features for read/write operation:

- Writes
  - Configurable depth write buffer
  - Posted writes (always, HPROT is don't care)
  - SPLIT response on write buffer full
  - Maximum of two wait states on non-sequential access
  - Zero wait states (full bandwidth) on sequential access
  - Zero BUSY cycles (full bandwidth), secondary burst generation
- Reads
  - Configurable depth read buffer
  - Prefetched reads
  - Non-prefetched reads
  - SPLIT response on non-sequential (non yet prefetched) access
  - Zero wait states (full bandwidth) on prefetched read data

1.3.4 Software Interface

- Interrupt signal on write errors
- Interrupt status/clear registers

1.3.5 Sideband Signals

- Input sstall pin to qualify an address phase for HREADY low operation mode
- Output sflush pin to monitor the flushing operation on the read buffer

1.4 Standards Compliance

The DW_ahb_eh2h 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_eh2h 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 79 section discusses the specific procedures for verifying the DW_ahb_eh2h.

1.6 Licenses

Before you begin using the DW_ahb_eh2h, 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_eh2h 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_eh2h 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_eh2h component within a DesignWare subsystem using coreAssembler, see "Overview of the coreAssembler Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

Like its predecessor, DW_ahb_h2h, DW_ahb_eh2h 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) see *DesignWare DW_ahb_h2h Databook*. The function of DW_ahb_eh2h 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

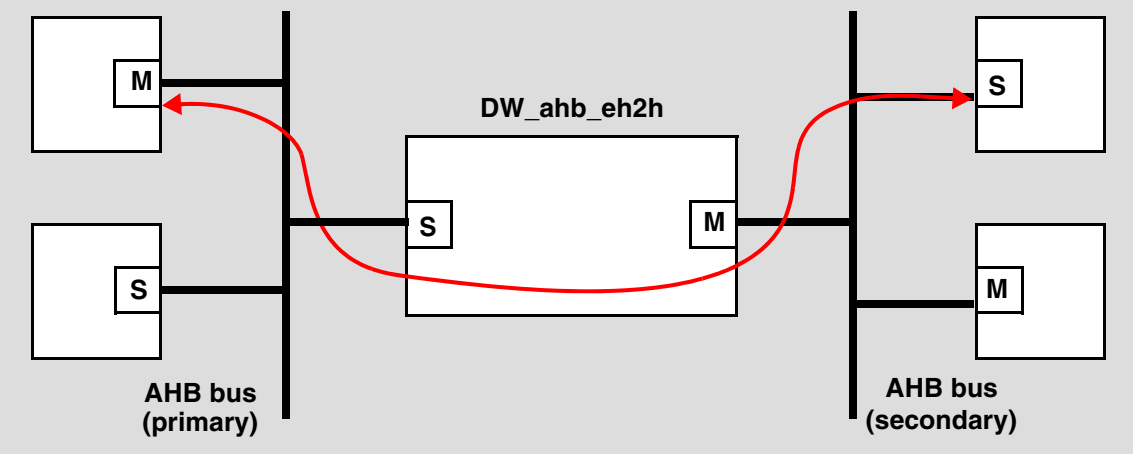
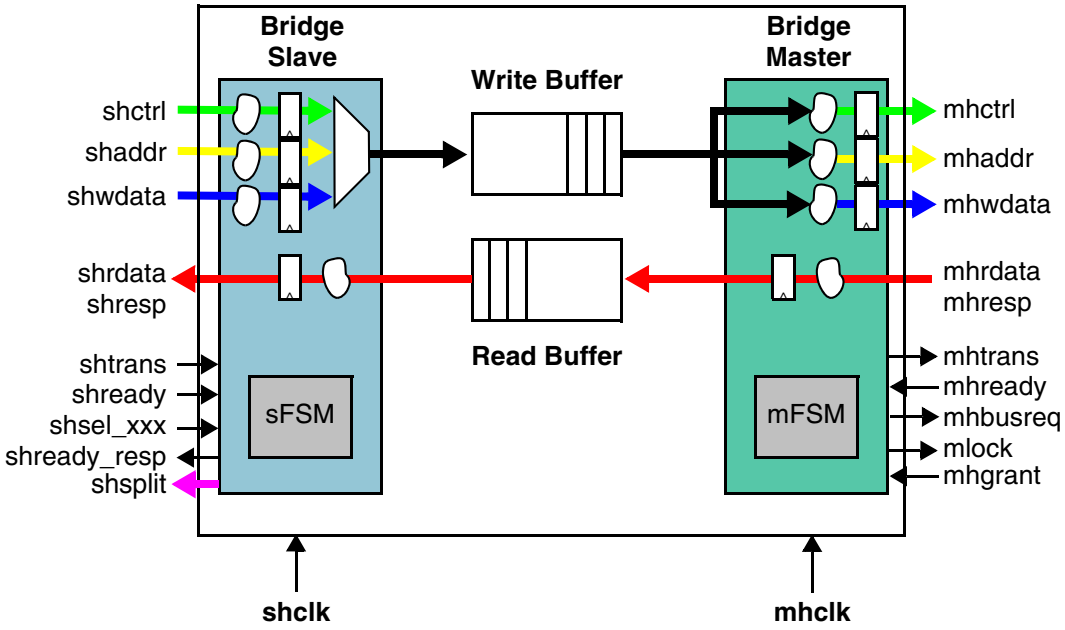


Figure 2-2 illustrates a more complex block diagram of the DW_ahb_eh2h.

Figure 2-2 DW_ahb_eh2h Block Diagram



2.1 Definitions

The following provides definitions for terms that are used throughout this chapter:

- **Transfer –** A transfer is a bus cycle where HREADY is high. Depending on the value of HTRANS, a transfer can be nonsequential (NSEQ), sequential (SEQ), IDLE, or BUSY. A transfer is issued by a bus master and responded by a bus slave.
- **Access –** An access to a given slave is a transfer qualified by one of the slave select lines.
- **Wait State –** A wait state is a bus cycle where HREADY is low and HRESP is OKAY.
- **OKAY Response –** A bus cycle where HREADY is high and HRESP is OKAY.
- **Non-OKAY Response –** Two consecutive bus cycles where HREADY is low and HRESP is ERROR, RETRY or SPLIT on the first cycle, HREADY is high and HRESP maintained to the previous value on the second cycle.
- **Response –** Sequence of bus cycles driven by a bus slave in response to a transfer issued by a bus master. The sequence consists of zero, one or more wait states followed either by OKAY or Non-OKAY response.
- **Transaction –** A sequence of transfers.
- **Throughput –** The average throughput for a bus transaction of length *n* is the ratio between the total number *n* of data phases in the transaction and the total number *n*+k of clock cycles required to execute them.

  Throughput = *n*/*n*+k

2.2 Bridge Slave Interface

The function of the bridge slave interface is described as follows:

2.2.1 Slave Selection

The bridge slave interface is sensitive to transfers qualified by either one of the three input select lines:

- shsel_reg

  When a transfer is qualified by shsel_reg, it is a "local access" — the bridge uses the lower 10 bits of shaddr to access local registers. In terms of bus protocol and bus response, a local access is treated the same way as any other bridge access. The only difference is that no corresponding transfer is produced by the bridge on the master side.

- **shsel_p** and **shsel_np**

  When a transfer is qualified by shsel_p or shsel_np, it is a "secondary access" — the bridge uses all the bits of shaddr to produce a corresponding transfer on the master side. When a read INCR transfer is qualified by shsel_p (*p* stands for "prefetch"), the bridge master prefetches read data for that transfer.

  When a read INCR transfer is qualified by shsel_np (*np* stands for "no prefetch"), the bridge master treats the request as a read SINGLE request, and no prefetch of read data occurs for sequential addresses following the first address of the burst.

  Selection with shsel_p or shsel_np only matters for read INCR accesses. For any other access types (write transfers any burst type, read transfers which are not INCR), shsel_p and shsel_np are equivalent.


Note When the bridge slave interface is not selected, the bridge drives 0 wait states OKAY response.

For more information about these signals, see "Signal Descriptions" on page 47.

2.2.2 Slave Response to Writes

The bridge response behavior on a write access depends on the value of shtrans and the availability[1] of the write buffer. The "minimum write buffer availability" required to accept a transfer with an OKAY response varies from 1 to 4 free entries, depending on the current and the previous transfer types.

When the write buffer availability is below the minimum required, the transfer is SPLIT. If the write buffer availability is equal or above the minimum required the following rules apply:

- NSEQ transfers are accepted with two wait states, OKAY response.
- SEQ transfers are normally accepted with 0 wait states, OKAY response.

1. The write buffer availability is defined as the number of free locations available on the next clock cycle. Because of the pipelined nature of the AHB bus, depending on the previous transfer seen by the bridge slave, the current clock cycle (on which the current transfer must be handled) may or may not require a write buffer push. The buffer availability can then be calculated as the number of free locations available in the current clock cycle (decreased by 1) when the current cycle requires a write buffer push.

- SEQ transfers can be occasionally accepted with two wait states, OKAY response. This can happen when the SEQ is preceded by BUSY cycles, and the write buffer has become full on the write data push of the previous SEQ/NSEQ write transfer.

If the write buffer availability is below the minimum:

- NSEQ/SEQ transfers are SPLIT in two cycles

Regardless the write buffer availability:

- BUSY and IDLE transfers are always responded with 0 wait states, OKAY response

2.2.3 Slave Response to Reads

The bridge response behavior on a read access depends on the value of shtrans and the read buffer status. When a read access is seen by the bridge slave for the first time, the transfer is split and a read request is forwarded to the bridge master. When the master receives the request, a read burst is executed onto the secondary AHB and prefetched data is stored into the read buffer. When all data beats for the read burst have been prefetched, the read buffer status becomes "ready."

Data stored into the read buffer is marked by a tag that identifies the primary master which originated the read transfer. When the buffer is ready and a NSEQ/SEQ read access occurs, a comparison is made between shmaster and the tag of the current read buffer data entry.

- A read access is marked by the bridge as "new" when either one of the following conditions occurs:
  - Read buffer is ready but shmaster does not match the tag.
  - The read buffer is "not ready".

    A new read access is typically a NSEQ access that is first seen by the bridge (was not previously split, and split cleared by the bridge) or a SEQ access that goes beyond the prefetched data availability into the buffer.
- A read access is marked by the bridge as "return" when the following condition is true:
  - The read buffer is ready and shmaster does match the tag.

    A return read access typically occurs when a previously split read transfer is cleared by the bridge itself and is retried by the primary master.
- NSEQ/SEQ "new" transfers are SPLIT in two cycles.
- NSEQ/SEQ "return" transfers are responded with 0 wait states, OKAY response or two cycle ERROR response, depending on the response received by the bridge master on the corresponding secondary transfer.
- BUSY and IDLE transfers are always responded with 0 wait states, OKAY response.

2.2.4 Read Buffer Flush

There are two situations where the bridge must flush out part of or all of the content of the read buffer:

- A locked transfer occurs when one or more previously split reads are still pending.
- A read "return" burst is early terminated before all data beats present in the buffer have been fetched by the primary master.

The read buffer is fully flushed when a locked transfer (read or write) is received and there are pending reads into the bridge. The read buffer is partially flushed when a read burst transfer terminates at the primary before all prefetched data entries stored into the buffer for that transfer have been popped out. The number of entries to be flushed in this case is given by the number of prefetched entries that are still in the buffer for that transfer.

The bridge has only one read buffer, which is used to queue in a first-in-first-out fashion sequential data beats associated with read bursts requested by different primary masters. Under normal conditions, a sequence of "new" read transfers (issued by different primary masters) is cleared and returned by the bridge in the exact order in which the masters were split. If the last transfer is locked, pending reads of the other master cannot be returned before the locked transfer has completed because the bus is locked and cannot be handed over to another master. To avoid deadlock, the bridge must flush any non-locked read data emerging from the first-in-first-out buffer to allow the last locked transfer to advance in the queue and complete first.

Similarly, when a "return" read burst is early terminated, the bridge must flush out any remaining sequential data beat (prefetched "in excess") from the buffer, which would otherwise prevent any subsequent read transfer to be returned and completed.

The output signal sflush is high whenever a data word is flushed out from the read buffer. The signal is provided to allow external monitoring on the flush functionality.

2.2.5 Read-Sensitive Locations

Typically, a read buffer flush is not a problem when data is prefetched from true memory. If the flushed data is still needed, the transfer is eventually repeated by the master. There is a loss in performance, but not a loss of data. When data is prefetched from read-sensitive locations, a read buffer flush produces a data loss.

To avoid this scenario, the following access rules must be followed when using the bridge to access read-sensitive locations:

1. Always use a locked access type
2. Always use a SINGLE burst type

Rule1 can be dropped if no master in the primary AHB ever requests locked access to the bridge.

2.2.6 Deadlock

Two bridges can be used to connect two AHB subsystems in a bidirectional fashion. The split response mechanisms automatically protect the system from deadlock. However, the case must be avoided where both bridges are accessed with locked access type at almost the same time. In this scenario, unrecoverable deadlock is produced. For more an illustration of a deadlock condition, see Figure 1-3.

2.2.7 Performance Impact of Locked Transfers

The bridge is designed to efficiently support traffic from multiple masters, where only a small percentage of the transfers are locked and the distribution of the requests is such that only a small fraction of the locked transfers cause the read buffer to flush. Handling of unlocked transfers becomes inefficient when intense read traffic from multiple masters occurs where a high percentage of the requests is locked. For this reason, locked access to the bridge must be used with care.

2.2.8 Performance Impact of Read Incremental Bursts

The bridge is designed to efficiently support read-defined length bursts, where the prefetch depth is known. Using undefined length read bursts becomes inefficient when a big fraction of the prefetched data is not needed by the master and is flushed by the bridge. For this reason, it is preferable to use defined-length burst types whenever possible.

2.2.9 Local Access

A local access must always be 32 bits in size and SINGLE burst type. The behavior of the bridge is undefined for other access types.

2.2.10 Prefetch Depth

The prefetch depth used by the bridge for a given read transfer depends on the following factors:

- Select line used to qualify the transfer: shsel_p/shsel_np
- Value of shburst on the read transfer
- Distance of shaddr from the 1 KB boundary
- Value of the EH2H_READ_PREFETCH_DEPTH configuration parameter
- Number of free locations in the read buffer

Under normal conditions, the bridge always attempts to prefetch all data beats for the read burst. If this is not possible because, for example, the number of free locations in the read buffer is too low, the bridge reduces the prefetch depth to the value that fills the buffer.

2.2.11 SPLIT Response and SPLIT Clear

A SPLIT response is only generated after:

- Write access or “new” read access when write buffer is full

  A SPLIT that occurs in this situation is said to be a “write” SPLIT because it is caused by the write buffer being unavailable.

- New read access when write buffer is not full

  A SPLIT that occurs in this situation is said to be a “read” split.

Both write and read SPLITs are always followed by a split clear.

A write SPLIT is cleared when the write buffer contains at least four free entries. The latency between the SPLIT response and the SPLIT clear depends on the capacity of the master interface to drain the write buffer.

A read SPLIT is cleared when the read buffer contains all prefetched read data associated with the SPLIT read transaction. The latency depends on the number of reads and writes queued into the write buffer before the current transaction, and the capacity of the master to execute queued transactions in the secondary AHB.

2.2.12 Alternative “HREADY Low” Response Mode

The SPLIT response functionality can be inhibited in two ways: (1) qualifying the address phase with the sstall signal driven to 1 or (2) configuring the bridge with EH2H_IS_SSPLIT_CAPABLE = 0. When the SPLIT functionality is inhibited, the response sequence SPLIT-response/SPLIT-clear is replaced by shready_resp low. Because hready low stalls the primary bus, transfers qualified by sstall = 1 are said to be “stalling” transfers. Like locked transfers, stalling transfers require the read buffer to be flushed.

2.2.13 Timeout on “HREADY Low” Response Mode

A timeout counter counts sttick transitions while shready_resp is low. If two transitions are counted while shready_resp is low, the transfer data phase is terminated with an ERROR response. The sttick signal is expected to be an externally driven periodic waveform. If the timeout functionality is not required, sttick can be hardwired to 1 or 0 on the bridge I/O.

2.3 Bridge Master Interface

The function of the master interface is described as follows:

2.3.1 Generation of Secondary Writes

The bridge master is sensitive to the status of the write buffer. As soon as write data is present in the buffer, the master requests the bus and setup address and control information for the new transfer. The transfer starts when the master is granted the bus; data is taken from the buffer and driven onto the bus.

Note

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

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

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

In the case of a burst, the first NSEQ transfer is followed by one or more SEQ transfers. Once the last data beat has been driven, burst execution terminates, mhtrans is driven to IDLE, and mhbusreq is driven low. Write bursts can only be of type SINGLE or INCR. If the primary requests a wrapped burst type, it is executed as an INCR burst, with mhtrans set to NSEQ on the address phase that follows the wrap.

2.3.2 Generation of Secondary Reads

The bridge master is sensitive to the status of the write buffer. As soon as a read request is present in the write buffer, the master checks the availability of the read buffer. If the read buffer is available, the master requests the bus and address and control information is setup for the new transfer. Otherwise, the master waits for the read buffer to be drained by the bridge slave. The transfer starts when the master is granted the

bus; data is sampled from the bus and put onto the read buffer. In case of a burst, the first NSEQ transfer is followed by one or more SEQ transfers. Once the last data beat has been pushed onto the buffer, burst execution terminates, mhtrans is driven to IDLE, and mhbusreq is driven low. Read bursts can be of any burst type according to the requested burst type and the data-width ratio between the slave interface and the master.

2.3.3 Behavior of ERROR Response

ERROR responses from a secondary slave are ignored. The bridge master does not insert IDLE on the second cycle of the two-cycle response, and the burst is continued as normal. In case of a read, the error response is returned to the primary master through the read buffer. In case of a write, an interrupt is generated.

2.3.4 Behavior of SPLIT/RETRY Response

SPLIT and RETRY are handled in the same way. The bridge master cancels the next transfer, inserting IDLE on the second cycle of the two-cycle response and repeats the retried/split transfer, reissuing address and controls with HBURST equal to INCR. This continues until an ERROR or OKAY response is received.

2.3.5 Behavior of Early Burst Termination

If mhgrant is removed during burst execution, the bridge master continues requesting and waits for the bus to be granted ownership again. Once bus ownership is granted, the burst is continued with HBURST equal to INCR.

When a transaction has to be rebuilt due to a SPLIT, RETRY, De-grant Event or Early Burst Termination, the bridge master always rebuilds the transfer with hburst equal to INCR. A De-grant Event is defined as “the bridge master's hbusreq is active while hready is active and hgrant is removed.”

2.3.6 BUSY Cycles

There are cases where the bridge masters insert BUSY cycles during the execution of write bursts. This can happen as a result of mhclk being much faster than shclk, or because of BUSY cycles inserted by the originating primary master. BUSY cycles are never inserted during execution of read bursts.

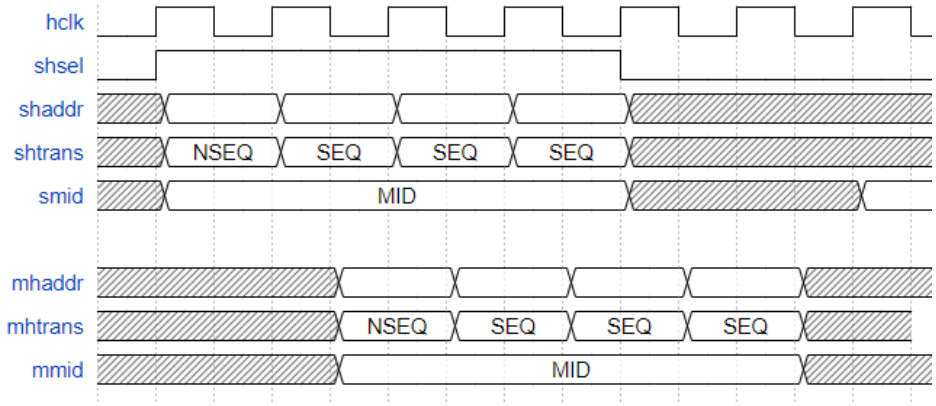
2.3.7 HLOCK

Transfers that are requested by the primary with shmastlock asserted are handled by the bridge master as locked. Bus arbitration is performed by the bridge master with mhbusreq and mhlock asserted.

2.4 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 EH2H_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 2-3 is an example AHB transaction in DW_ahb_eh2h.

Figure 2-3 AHB Transaction in DW_ahb_eh2h With MID Sideband Signal



2.5 Write Buffer

The write buffer is used to transmit read and write requests from the bridge slave to the bridge master.

In write direction, the buffer is pushed by the bridge slave with the following information:

- A control word containing burst type, master number, protection, locked attributes for the burst
- An address word containing the nonsequential start address for the burst
- One or more consecutive data words containing the write data for the burst

In read direction, the buffer is pushed by the bridge slave with the following information:

- A control word containing burst type, master number, maximum prefetch depth to be applied
- An address word containing the nonsequential start address for the burst

The two wait states inserted on NSEQ writes are required to push address and controls. The depth of the buffer is under user control (minimum is four). The bigger the depth of the buffer, the more the capacity of the bridge to smooth out peaks in the write bandwidth.

2.6 Read Buffer

The read buffer is used to transmit prefetched read data from the bridge master to the bridge slave.

The buffer is pushed by the bridge master with the following information:

- A control word containing burst type, master number, prefetch depth actually applied by the bridge master
- One or more consecutive data words containing read data for burst and response

The depth of the buffer is under user control (minimum is four). The bigger the depth of the buffer, the more the capacity of the bridge to store prefetched data for longer bursts or for multiple bursts requested in succession by different masters.

The guideline formula to set the read buffer depth to allow prefetch of *k* consecutive read bursts of length *N* is:

```
READ_BUFFER_DEPTH=k*(N+1)
```

For example, to prefetch one burst of length 16, READ_BUFFER_DEPTH must be set to 17. To allow up to 2 INCR8 burst, READ_BUFFER_DEPTH must be set to 18.

2.7 Pipelines

To support high clock speed with tight input delay constraints, the read and write buffers are equipped with retiming pipeline stages placed in front of push and pop interfaces.

The scope of these pipelines is to reduce critical timing paths between unregistered bus inputs and the buffer's push/pop logic. You can control instantiation of the pipelines using four independent "pipe-mode" configuration parameters:

EH2H_WRITE_BUFFER_PUSH_PIPE_MODE
EH2H_READ_BUFFER_PUSH_PIPE_MODE
EH2H_WRITE_BUFFER_POP_PIPE_MODE
EH2H_READ_BUFFER_POP_PIPE_MODE

The pipelines are transparent in terms of functionality (apart from a small extra latency effect). For applications where timing is not critical, you can avoid the instantiation of the pipelines setting these pipe-mode parameters to 0.

For applications where timing is critical, you can determine if timing requirements are met instantiating all the pipelines and synthesizing the component. At a later stage, a selective removal of the pipelines from paths that are not critical can be performed to optimize for area.

For more information about setting these configuration parameters, see "Parameter Descriptions" on page 39.

2.8 Interrupt and Software Registers Interface

The following local registers are available within the bridge:

- EWS: Errored Write Status register
- MEWS: Multiple Errored Write Status register
- EWSC: Errored Write Status Clear register
- Bit *K* of EWS is set when a write transfer (originated by the primary master *K*) executed by the bridge master receives an ERROR response.
- Bit *K* of MEWS is set when a write transfer (originated by the primary master *K*) receives an ERROR response and the corresponding bit *K* in EWS is already set.
- Writing to EWSC, a vector with bit *K* set to 1 causes bit *K* to be reset in EWS and MEWS registers.

After issuing a block of writes, the master can check if the writes are successful issuing a read to EWS. This methodology can be used to facilitate software debugging. The interrupt line is generated from the OR of EWS bits. The line is synchronous to mhclk.

For more information about these registers, see "Register Descriptions" on page 59.

2.9 Clocking

The following section discuss clocking details

2.9.1 Clock Adaptation

The bridge can be used to connect AHB systems running with arbitrary clocks. The bridge slave is a single clock design operating with the primary AHB clock; the bridge master is a single clock design operating with the secondary AHB clock. Clock adaptation is performed by the DesignWare Dual-Clock FIFO Controllers (DW_fifoctl_s2_sf), internally used to implement FIFO management of the read and write memory buffers.

If the EH2H_CLK_MODE parameter is set to "Synchronous," a synchronization depth of 1 is used between the push and pop clock domains of the internal dual clock FIFOs. Although the synchronization has a depth of 1—that is, a signal can be synchronized in a single clock cycle—it is actually comprised of two synchronization register stages. The first stage is performed on the negative edge of the destination clock, and the second stage is performed on the positive edge of the destination clock.

You can use the EH2H_CLK_MODE parameter, described in "Parameter Descriptions" on page 39, to specify whether the clocks are synchronous or asynchronous during configuration of the DW_ahb_eh2h. The synchronous clock mode configuration does not affect the overall functionality of DW_ahb_eh2h but allows for a reduction in the synchronization latency between the operations of the bridge slave and the bridge master.

For more information about the DW Dual-Clock FIFO controllers, go to the following Synopsys web page:

http://www.synopsys.com/dw/ipdir.php?c=DW_fifoctl_s2_sf

If EH2H_CLK_MODE is set to asynchronous, you can individually choose the synchronization depths for signals traversing from master interface to slave interface (EH2H_SIF_SYNC_DEPTH), and from slave interface to master interface (EH2H_MIF_SYNC_DEPTH).

For both parameters, there is a choice of two- or three-stage positive-edge synchronization.

The DW_ahb_eh2h performs all clock domain crossing through the internal dual clock FIFOs. Within these FIFOs, the pointer values are synchronized across clock domains, using the synchronization depth specified by the user. The data from the FIFO is sampled from push clock domain registers to the pop clock domain through a read multiplexor. Control logic within the DW_ahb_eh2h ensures that the currently selected read data from the FIFO is not sampled in the same cycle that it is being updated by the push clock domain.

2.9.2 Reset Signals

To avoid serious operational failures, both sides of the DW_ahb_eh2h should be reset before any system traffic reaches it; that is, it is an illegal operation to reset just one side of the DW_ahb_eh2h without resetting the other side. It is not necessary to activate the mhresetn and shresetn signals simultaneously; they can be reset at different times, but nothing should try to access the DW_ahb_eh2h between the time when one side is reset and the other is reset.

Each reset signal can be asserted asynchronously, but it must be de-asserted synchronously with respect to its own clock; that is mhresetn is de-asserted synchronously with respect to mhclk, and shresetn is de-asserted synchronously with respect to shclk.

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

No transaction should be driven to the DW_ahb_eh2h under these combined conditions:

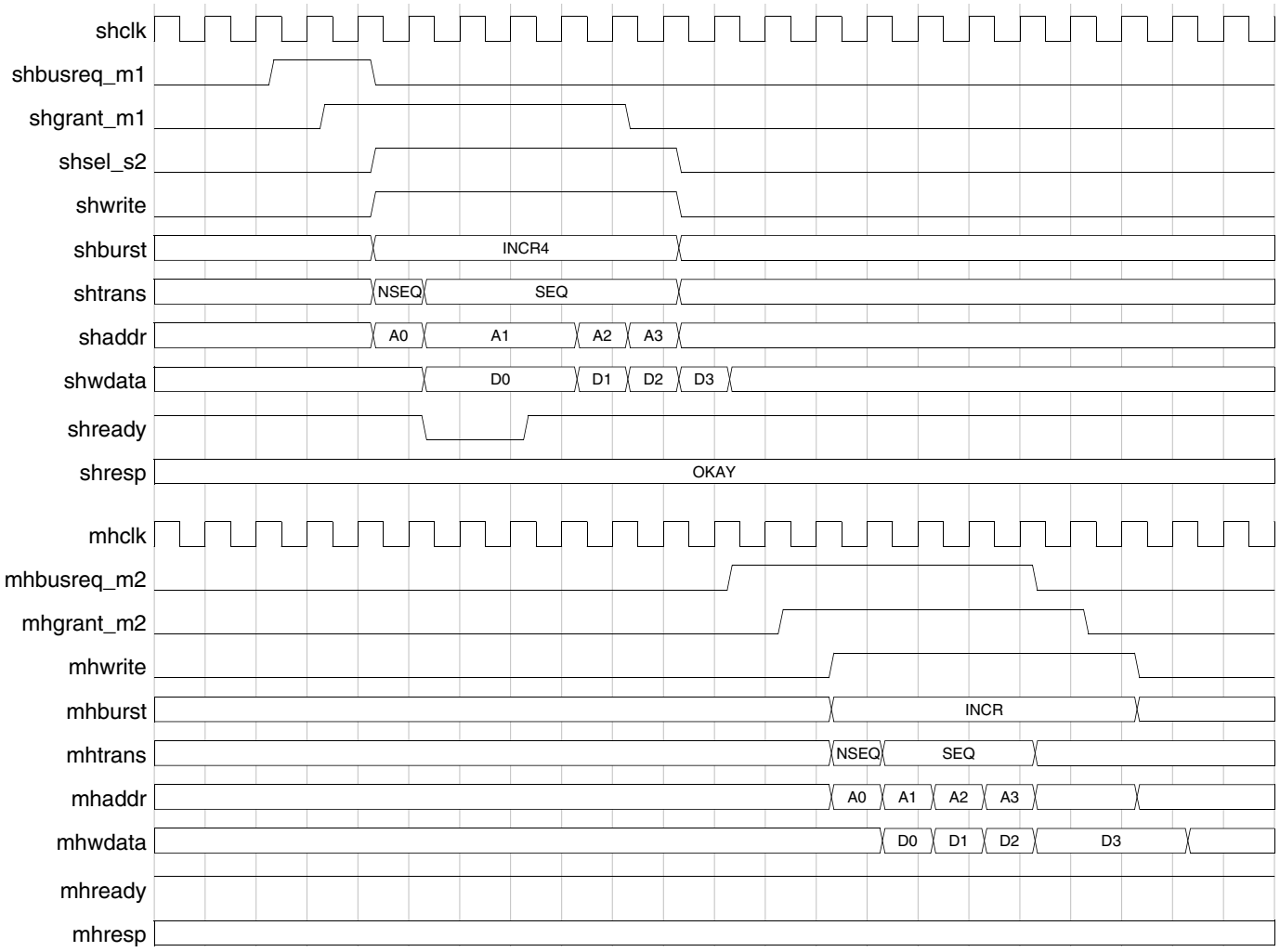
1. During assertion of mhresetn and shresetn.
2. For one additional cycle of shclk for the slave interface.

2.10 Timing Diagrams

This section provides timing diagrams for the DW_ahb_eh2h.

The timing diagram in Figure 2-4 shows the execution of a write INCR4 burst performed by the primary master 1 (m1) to the bridge. Two wait states are inserted by the bridge slave on the first NSEQ burst beat, as can be seen by the transitions on shready. The remaining SEQ beats have no wait states. After several clock cycles of latency, the bridge master requests the bus, the arbiter grants ownership to master 2 (m2, the bridge), and the bridge master executes the burst in four clock cycles.

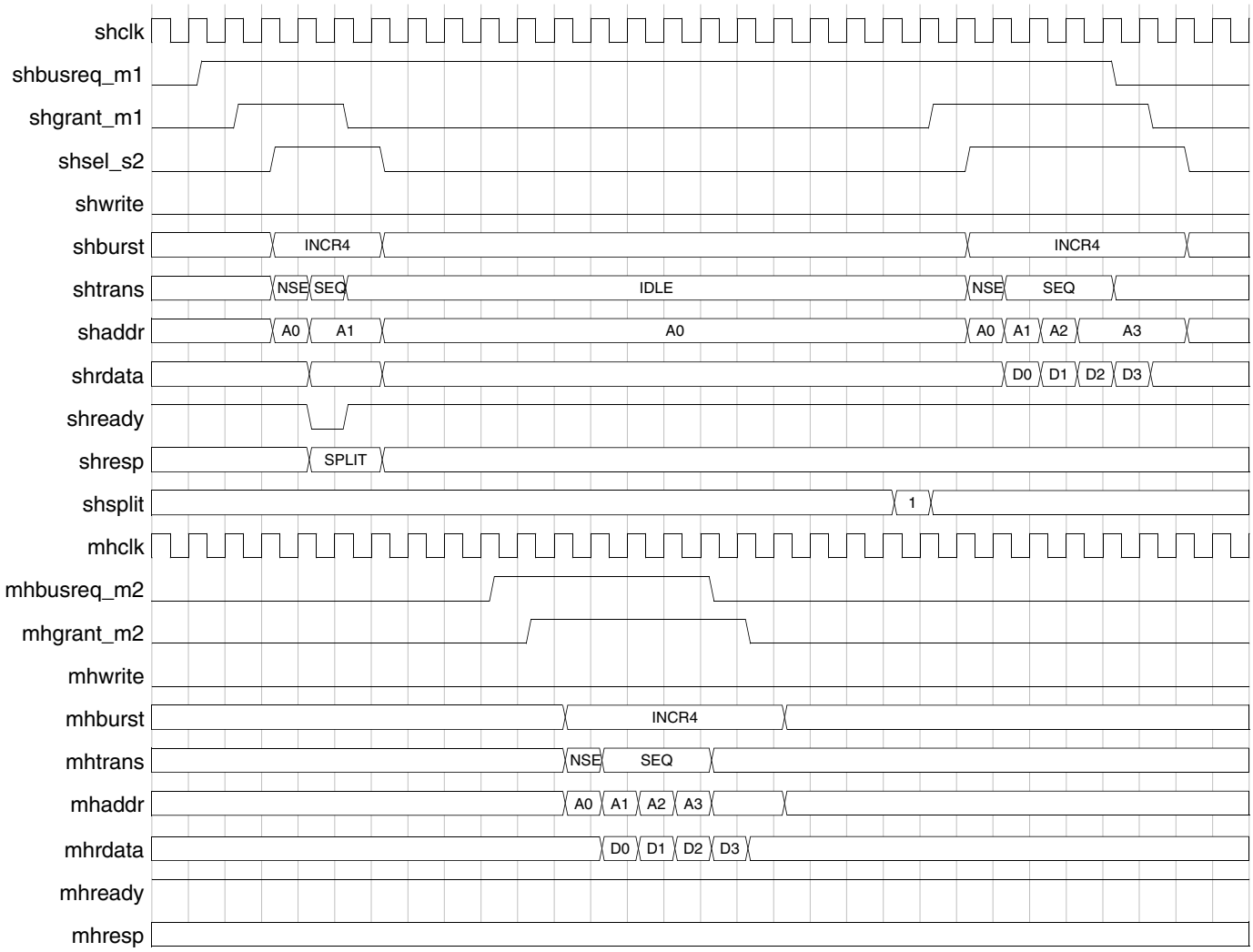
Figure 2-4 Write Burst



The timing diagram in Figure 2-5 shows the execution of a read INCR4 burst performed by master 1 (m1) to the bridge. The bridge uses split response operation mode. The bridge slave splits the transfer on the first NSEQ burst beat. This can be seen by the transitions on shresp. At a later stage, the split is cleared as

indicated by the transition on the shsplit bus. The split master then retries the split transfer and completes the burst in four clock cycles. In between the split response and the split clear events, the bridge master prefetches read data from the secondary bus, as indicated by transitions on mh* signals.

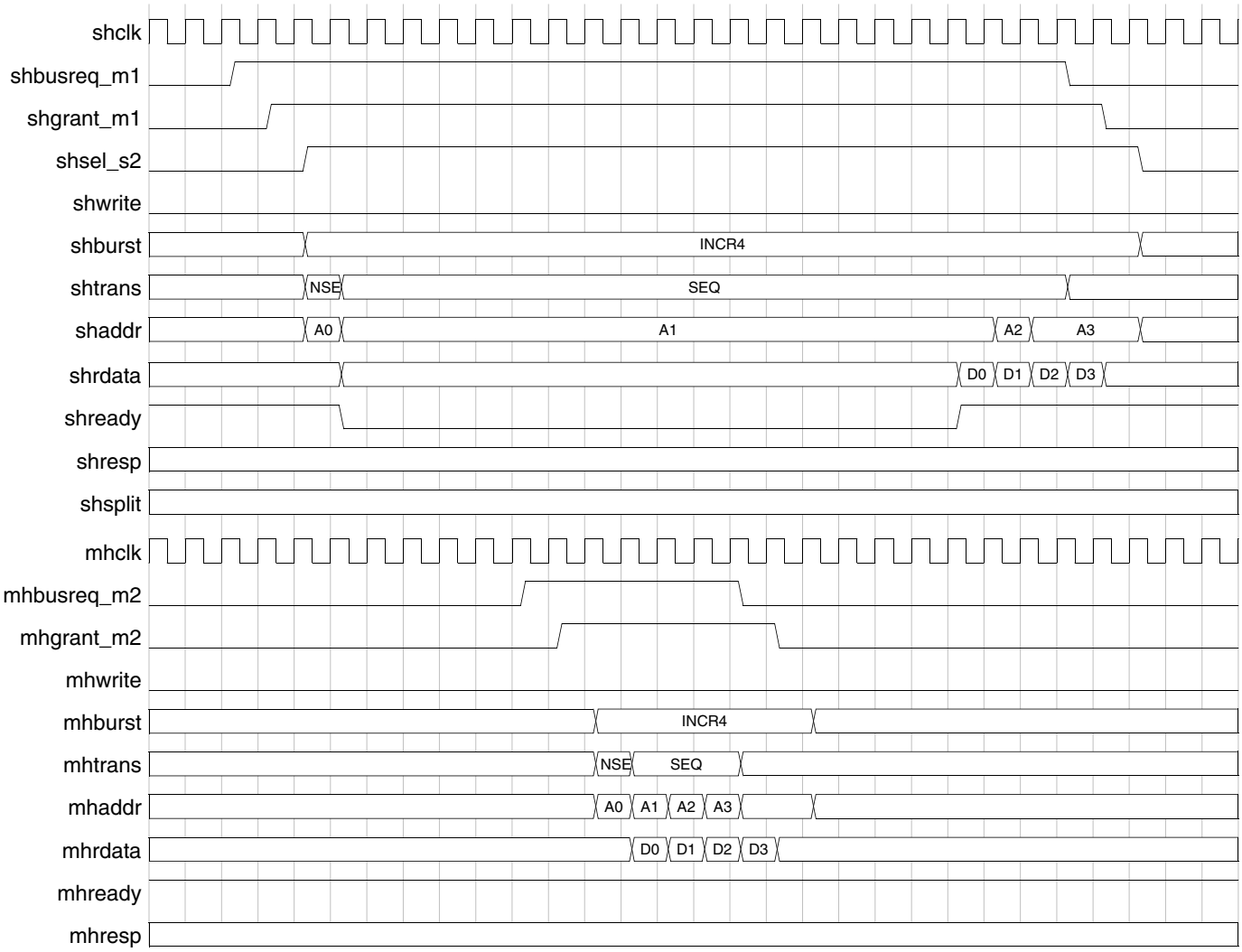
Figure 2-5 Read Burst With Split Response Operation Mode



The timing diagram in Figure 2-6 shows the execution of a read INCR4 burst performed by master 1 (m1) to the bridge. The bridge uses hready low operation mode. The bridge slave drives shready low on the first NSEQ burst beat. At a later stage, shready is driven high, allowing the master to complete the burst in four

clock cycles. While hready is low, the bridge master prefetches read data from the secondary bus, as indicated by transitions on mh* signals.

Figure 2-6 Read Burst With hready Low Operation Mode



3
Parameter Descriptions

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

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

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

3.1 Parameters

Table 3-1 Parameters

Label	Description
Address bus width	Specifies the address bus width of the primary AHB system to which the bridge is attached as an AHB slave. The address and write data are sequentially pushed into the write buffer. The write buffer width is determined by the data width. If the data is less than 64 bits, the address width must be restricted to match the data width. Therefore, this parameter can be set to 64 only when the data width is greater than 32 bits. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** EH2H_PHY_SDATA_WIDTH>32 **Parameter Name:** EH2H_PHY_SADDR_WIDTH
Data bus width	Specifies the read and write data bus width of the primary AHB system to which the bridge is attached as an AHB slave. **Values:** ■ 32 (32) ■ 64 (64) ■ 128 (128) ■ 256 (256) **Default Value:** 32 **Enabled:** Always **Parameter Name:** EH2H_PHY_SDATA_WIDTH
Data bus endianness	Specifies the data bus endianness of the primary AHB system to which the bridge is attached as an AHB slave. **Values:** ■ Little-Endian (0) ■ Big-Endian (1) **Default Value:** Little-Endian **Enabled:** Always **Parameter Name:** EH2H_PHY_SBIG_ENDIAN
Number of masters in primary AHB	Specifies the number of masters in the primary AHB. This parameter determines how many bits are valid in the shsplit output bus and in the interrupt status registers. **Values:** 1, ..., 15 **Default Value:** 3 **Enabled:** Always **Parameter Name:** EH2H_PHY_NUM_PRIMARY_MASTERS

Table 3-1 Parameters (Continued)

Label	Description
Address bus width	Specifies the address bus width of the secondary AHB system to which the bridge is attached as an AHB master. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** EH2H_PHY_MADDR_WIDTH
Data bus width	Specifies the read and write data bus width of the secondary AHB system to which the bridge is attached as an AHB master. **Values:** ■ 32 (32) ■ 64 (64) ■ 128 (128) ■ 256 (256) **Default Value:** 32 **Enabled:** Always **Parameter Name:** EH2H_PHY_MDATA_WIDTH
Data bus endianness	Specifies the Data bus endianness of the secondary AHB system to which the bridge is attached as an AHB master. **Values:** ■ Little-Endian (0) ■ Big-Endian (1) **Default Value:** Little-Endian **Enabled:** Always **Parameter Name:** EH2H_PHY_MBIG_ENDIAN
Write buffer depth	Specifies the number of locations in the write buffer. The write buffer transfers controls, addresses, and write data from the bridge slave to the bridge master. **Values:** 4, ..., 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** EH2H_WRITE_BUFFER_DEPTH
Read buffer depth	Specifies the number of locations in the read buffer. The read buffer transfers controls, addresses, and read data from the bridge master to the bridge slave. **Values:** 4, ..., 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** EH2H_READ_BUFFER_DEPTH

Table 3-1 Parameters (Continued)

Label	Description
Read prefetch depth	Specifies the number of locations considered for prefetch by the bridge master when a read undefined-length incremental burst operation is requested by the bridge slave. **Values:** 1, ..., 16 **Default Value:** 1 **Enabled:** Always **Parameter Name:** EH2H_READ_PREFETCH_DEPTH
Write buffer push pipe mode	Reduces the critical timing path length between the bridge slave bus inputs and the write buffer push logic, allowing the instantiation of a pipeline stage in front of the push interface of the write buffer. **Values:** ■ No-Pipe (0) ■ Pipe (1) **Default Value:** Pipe **Enabled:** Always **Parameter Name:** EH2H_WRITE_BUFFER_PUSH_PIPE_MODE
Read buffer push pipe mode	Reduces the critical timing path length between the bridge master bus inputs and the read buffer push logic, allowing the instantiation of a pipeline stage in front of the push interface of the read buffer. **Values:** ■ No-Pipe (0) ■ Pipe (1) **Default Value:** Pipe **Enabled:** Always **Parameter Name:** EH2H_READ_BUFFER_PUSH_PIPE_MODE
Write buffer pop pipe mode	Reduces the critical timing path length between the bridge master bus inputs and the write buffer pop logic, allowing the instantiation of a pipeline stage in front of the pop interface of the write buffer. **Values:** ■ No-Pipe (0) ■ Pipe (1) **Default Value:** Pipe **Enabled:** Always **Parameter Name:** EH2H_WRITE_BUFFER_POP_PIPE_MODE

Table 3-1 Parameters (Continued)

Label	Description
Read buffer pop pipe mode	Reduces the critical timing path length between the bridge slave bus inputs and the read buffer pop logic, allowing the instantiation of a pipeline stage in front of the pop interface of the read buffer. **Values:** ■ No-Pipe (0) ■ Pipe (1) **Default Value:** Pipe **Enabled:** Always **Parameter Name:** EH2H_READ_BUFFER_POP_PIPE_MODE
Clocking mode	Determines how two clock domains must be synchronized. The bridge slave is always clocked by shclk and reset by shresetn. The bridge master is always clocked by mhclk and reset by mhresetn. **Values:** ■ Asynchronous (0) ■ Synchronous (1) **Default Value:** Asynchronous **Enabled:** Always **Parameter Name:** EH2H_CLK_MODE
Slave Interface Synchronisation Depth ?	Determines the amount of synchronization to be placed on signals crossing from the master interface to the slave interface. This controls the depth of synchronization added to the push pointer of the read buffer and the pop pointer of the write buffer. **Values:** ■ 2-stage posedge (2) ■ 3 stage posedge (3) **Default Value:** 3 stage posedge **Enabled:** EH2H_CLK_MODE==0 **Parameter Name:** EH2H_SIF_SYNC_DEPTH
Master Interface Synchronisation Depth ?	Determines the amount of synchronization to be placed on signals crossing from the slave interface to the master interface. This controls the depth of synchronization added to the pop pointer of the read buffer and the push pointer of the write buffer. **Values:** ■ 2-stage posedge (2) ■ 3 stage posedge (3) **Default Value:** 3 stage posedge **Enabled:** EH2H_CLK_MODE==0 **Parameter Name:** EH2H_MIF_SYNC_DEPTH

Table 3-1 Parameters (Continued)

Label	Description
SPLIT capable slave	Enables generation of a SPLIT response and SPLIT clear from the slave interface. When this parameter is False, the slave interface never generated s SPLIT. When True, the slave interface SPLIT behavior can be controlled dynamically using the sstall input signal. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** EH2H_IS_SSPLIT_CAPABLE
Non Standard Master ID Sideband 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. **Values:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 0 **Enabled:** Always **Parameter Name:** EH2H_MID_WIDTH
Connect shsel_p	Controls the connection of the shsel_p signal to the AHB fabric. If this value is set to 1, the shsel_p signal is included in the connection to the target interface. If this value is set to 0, the shsel_p input signal is tied to logic 0. The connection order for hsel is: ■ shsel_p ■ shsel_np ■ shsel_reg **Values:** 0, 1 **Default Value:** 1 **Enabled:** This parameter is used in the coreAssembler. **Parameter Name:** EH2H_CONNECT_HSEL_P
Connect shsel_np	Controls the connection of the shsel_np signal to the AHB fabric. If this value is set to 1, the shsel_np signal is included in the connection to the target interface. If this value is set to 0, the shsel_np input signal is tied to logic 0. If EH2H_CONNECT_HSEL_P is enabled, shsel_np connects to the slave index after shsel_p. The connection order for hsel is: ■ shsel_p ■ shsel_np ■ shsel_reg **Values:** 0, 1 **Default Value:** 0 **Enabled:** This parameter is used in the coreAssembler. **Parameter Name:** EH2H_CONNECT_HSEL_NP

Table 3-1 Parameters (Continued)

Label	Description
Connect shsel_reg	Controls the connection of the shsel_reg signal to the AHB fabric. If this value is set to 1, the shsel_reg signal is included in the connection to the target interface. If this value is set to 0, the shsel_reg input signal is tied to logic 0. If EH2H_CONNECT_HSEL_P and EH2H_CONNECT_HSEL_NP are enabled, the shsel_reg signal connects to the slave index after the shsel_np signal. If EH2H_CONNECT_HSEL_NP is not enabled, shsel_reg connects to the slave index after shsel_p. The connection order for hsel is: ■ shsel_p ■ shsel_np ■ shsel_reg **Values:** 0, 1 **Default Value:** 0 **Enabled:** This parameter is used in the coreAssembler. **Parameter Name:** EH2H_CONNECT_HSEL_REG

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 clock(s) 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: Name of configuration parameter(s) that populates this signal in your configuration.

Validated by: Assertion or de-assertion of signal(s) 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:

- Slave Interface on page 49
- Master Interface on page 54
- Miscellaneous Signals on page 57

4.1 Slave Interface Signals

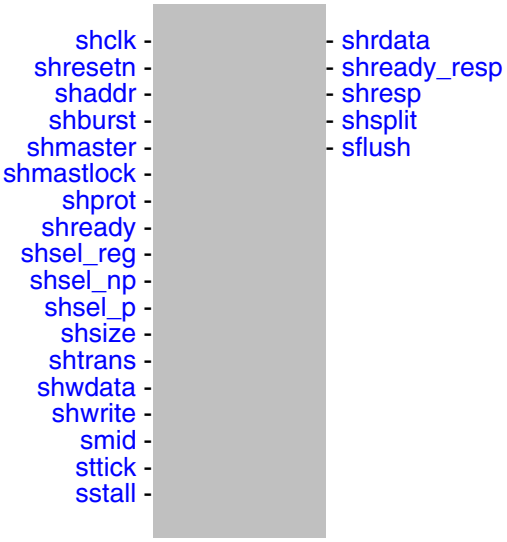


Table 4-1 Slave Interface Signals

Port Name	I/O	Description
shrdata[(SDW-1):0]	O	Slave read data **Exists:** Always **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shready_resp	O	Slave current data phase complete **Exists:** Always **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
shresp[1:0]	O	Slave response control **Exists:** Always **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 Slave Interface Signals (Continued)

Port Name	I/O	Description
shsplit[15:0]	O	Slave split clear indication for the arbiter **Exists:** Always **Synchronous To:** EH2H_IS_SSPLIT_CAPABLE==1 ? "shclk" : "None" **Registered:** EH2H_IS_SSPLIT_CAPABLE == 1 ? "0:0=No;15:1=Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
sflush	O	External monitoring on the flush functionality.This signal is high whenever a data word is flushed out from the read buffer. **Exists:** Always **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
shclk	I	Slave clock **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shresetn	I	Slave reset. Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of slave port clock. DW_ahb_eh2h 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 address **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 Slave Interface Signals (Continued)

Port Name	I/O	Description
shburst[2:0]	I	Slave burst control **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shmaster[3:0]	I	Master which owns address bus **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shmastlock	I	Slave lock control **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shprot[3:0]	I	Slave protection control **Exists:** Always **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shready	I	Slave previous data phase complete **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shsel_reg	I	Slave select used for local register access **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 Slave Interface Signals (Continued)

Port Name	I/O	Description
shsel_np	I	Slave select used for secondary access when prefetch on INCR reads can be enabled **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shsel_p	I	Slave select used for secondary access when prefetch on INCR reads can be disabled **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
shsize[2:0]	I	Slave size control **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shtrans[1:0]	I	Slave transfer control **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shwdata[(SDW-1):0]	I	Slave write data **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
shwrite	I	Slave direction control **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 Slave Interface Signals (Continued)

Port Name	I/O	Description
smid[(EH2H_MID_WIDTH-1):0]	I	Optional. Non standard Master ID sideband signal input. **Exists:** EH2H_MID_WIDTH!=0 **Synchronous To:** shclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
sttick	I	Slave periodic waveform used to timeout HREADY low responses **Exists:** Always **Synchronous To:** shclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
sstall	I	Slave address phase qualifier for transfers. ■ 1: HREADY low response mode ■ 0: SPLIT response mode **Exists:** Always **Synchronous To:** (EH2H_IS_SSPLIT_CAPABLE == 1) ? "shclk" : "None" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.2 Master Interface Signals

Table 4-2 Master Interface Signals

Port Name	I/O	Description
mhaddr[(MAW-1):0]	O	Master address **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhburst[2:0]	O	Master burst control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhbusreq	O	Master bus request status **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhlock	O	Master lock control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2 Master Interface Signals (Continued)

Port Name	I/O	Description
mhprot[3:0]	O	Master protection control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhsize[2:0]	O	Master size control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhtrans[1:0]	O	Master transfer control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwdata[(MDW-1):0]	O	Master write data **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhwrite	O	Master direction control **Exists:** Always **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
mmid[(EH2H_MID_WIDTH-1):0]	O	Optional. Non standard Master ID sideband signal output. **Exists:** EH2H_MID_WIDTH!=0 **Synchronous To:** mhclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Master Interface Signals (Continued)

Port Name	I/O	Description
mhclk	I	Master clock **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhresetn	I	Master reset. Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of master port clock. DW_ahb_eh2h 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	Master bus grant status **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhrdata[(MDW-1):0]	I	Master read data **Exists:** Always **Synchronous To:** mhclk **Registered:** EH2H_PHY_MDATA_WIDTH > EH2H_PHY_SDATA_WIDTH ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
mhready	I	Master data phase complete **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mhresp[1:0]	I	Master response control **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 Miscellaneous Signals

Table 4-3 Miscellaneous Signals

Port Name	I/O	Description
minterrupt	O	Master interrupt generated from the OR of the Error on Write Status register bits **Exists:** Always **Synchronous To:** mhclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Register Descriptions

This chapter details all possible registers in the core. They are arranged hierarchically into maps and blocks (banks). For configurable IP titles, your actual configuration might not contain all of these registers.

Attention: For configurable IP titles, do not use this document to determine the exact attributes of your register map. It is for reference purposes only.

When you configure the core in coreConsultant, you must access the register attributes for your actual configuration at workspace/report/ComponentRegisters.html or workspace/report/ComponentRegisters.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the registers 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 Offset and Memory Access values 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 core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Exists Expressions

The Exist expressions indicate the combination of configuration parameters required for a register, field, or block to exist in the memory map. The expression is only valid in the local context and does not indicate the conditions for existence of the parent. For example, the Exists expression for a bit field in a register assumes that the register exists and does not include the conditions for existence of the register.

Offset

The term *Offset* is synonymous with *Address.*

Memory Access Attributes

The Memory Access attribute is defined as <ReadBehavior>/<WriteBehavior> which are defined in the following table.

Table 5-1 Possible Read and Write Behaviors

Read (or Write) Behavior	Description
RC	A read clears this register field.
RS	A read sets this register field.
RM	A read modifies the contents of this register field.
Wo	You can only write to this register once field.
W1C	A write of 1 clears this register field.
W1S	A write of 1 sets this register field.
W1T	A write of 1 toggles this register field.
W0C	A write of 0 clears this register field.
W0S	A write of 0 sets this register field.
W0T	A write of 0 toggles this register field.
WC	Any write clears this register field.
WS	Any write sets this register field.
WM	Any write toggles this register field.
no Read Behavior attribute	You cannot read this register. It is Write-Only.
no Write Behavior attribute	You cannot write to this register. It is Read-Only.

Table 5-2 Memory Access Examples

Memory Access	Description
R	Read-only register field.
W	Write-only register field.
R/W	Read/write register field.
R/W1C	You can read this register field. Writing 1 clears it.
RC/W1C	Reading this register field clears it. Writing 1 clears it.
R/Wo	You can read this register field. You can only write to it once.

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

Attribute	Description
Volatile	As defined by the IP-XACT specification. If true, indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. For example, when the core updates the register field contents.
Testable	As defined by the IP-XACT specification. Possible values are unconstrained, untestable, readOnly, writeAsRead, restore. Untestable means that this field is untestable by a simple automated register test. For example, the read-write access of the register is controlled by a pin or another register. readOnly means that you should not write to this register; only read from it. This might apply for a register that modifies the contents of another register.
Reset Mask	As defined by the IP-XACT specification. Indicates that this register field has an unknown reset value. For example, the reset value is set by another register or an input pin; or the register is implemented using RAM.
* Varies	Indicates that the memory access (or reset) attribute (read, write behavior) is not fixed. For example, the read-write access of the register is controlled by a pin or another register. Or when the access depends on some configuration parameter; in this case the post-configuration report in coreConsultant gives the actual access value.

Component Banks/Blocks

The following table shows the address blocks for each memory map. Follow the link for an address block to see a table of its registers.

Table 5-4 Address Banks/Blocks for Memory Map: DW_ahb_eh2h_mem_map

Address Block	Description
DW_ahb_eh2h_addr_block1 on page 62	DW_ahb_eh2h address block **Exists:** Always

5.1 DW_ahb_eh2h_mem_map/DW_ahb_eh2h_addr_block1 Registers

DW_ahb_eh2h address block Follow the link for the register to see a detailed description of the register.

Table 5-5 Registers for Address Block: DW_ahb_eh2h_mem_map/DW_ahb_eh2h_addr_block1

Register	Offset	Description
EH2H_EWSC on page 63	0x0	This register resets the EH2H_EWS[K] and EH2H_MEWS[K] bits.
EH2H_EWS on page 65	0x4	This register is set when the bridge master receives an ERROR response on a write transfer. The...
EH2H_MEWS on page 67	0x8	This register is set when the bridge master receives an ERROR response on a write transfer and the...
EH2H_COMP_PARM_1 on page 69	0x3f0	This register provides the value of the configured parameters, namely EH2H_IS_SSPLIT_CAPABLE, EH2H_CLK_MODE,...
EH2H_COMP_PARM_2 on page 73	0x3f4	This register provides the value of the configured parameters namely, EH2H_WRITE_BUFFER_DEPTH, EH2H_READ_BUFFER...
EH2H_COMP_VERSION on page 74	0x3f8	This register provides the EH2H Component Version ID.
EH2H_COMP_TYPE on page 75	0x3fc	This register provides information about the Component Type.

5.1.1 EH2H_EWSC

- **Name:** Error on Write Status Clear
- **Description:** This register resets the EH2H_EWS[K] and EH2H_MEWS[K] bits.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

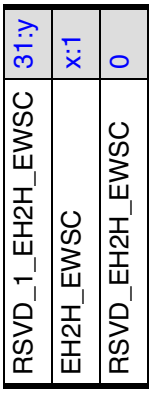


Table 5-6 Fields for Register: EH2H_EWSC

Bits	Name	Memory Access	Description
31:y	RSVD_1_EH2H_EWSC	W	(RSVD_1_EH2H_EWSC) These bits of the EH2H_EWSC register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** EH2H_PHY_NUM_PRIMARY_MASTERS + 1
x:1	EH2H_EWSC	W	This bit denotes whether the status is cleared or not. **Values:** ■ 0x0 (NO_EFFECT): No effect ■ 0x1 (CLEAR_STATUS): Clear the status **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** EH2H_PHY_NUM_PRIMARY_MASTERS

Table 5-6 Fields for Register: EH2H_EWSC (Continued)

Bits	Name	Memory Access	Description
0	RSVD_EH2H_EWSC	W	These bits of the EH2H_EWSC register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always

5.1.2 EH2H_EWS

- **Name:** Error on Write Status
- **Description:** This register is set when the bridge master receives an ERROR response on a write transfer. The index K corresponds to the primary master which originated the transfer. After issuing a block of writes, the primary master can check if the writes were successful by issuing a read to EWS. This methodology can be used to facilitate software debugging.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** Always

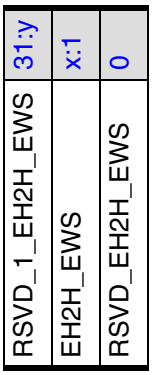


Table 5-7 Fields for Register: EH2H_EWS

Bits	Name	Memory Access	Description
31:y	RSVD_1_EH2H_EWS	R	This bit of the EH2H_EWS register is reserved. It always returns 0. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** EH2H_PHY_NUM_PRIMARY_MASTERS + 1

Table 5-7 Fields for Register: EH2H_EWS (Continued)

Bits	Name	Memory Access	Description
x:1	EH2H_EWS	R	These bits denote the Interrupt status. The interrupt line (minterrupt) is generated from the OR of the EWS bits and is synchronous to mhclk. **Values:** ■ 0x0 (INACTIVE): Interrupt is Inactive ■ 0x1 (ACTIVE): Interrupt is Active **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** EH2H_PHY_NUM_PRIMARY_MASTERS
0	RSVD_EH2H_EWS	R	These bits of the EH2H_EWS register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.3 EH2H_MEWS

- **Name:** Multiple Errors on Write Status
- **Description:** This register is set when the bridge master receives an ERROR response on a write transfer and the EH2H_EWS[K] bit is already set. The index K corresponds to the primary master that originated the transfer.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

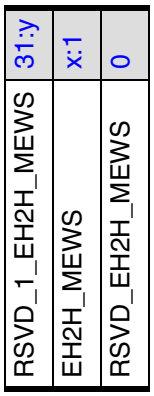


Table 5-8 Fields for Register: EH2H_MEWS

Bits	Name	Memory Access	Description
31:y	RSVD_1_EH2H_MEWS	R	These bits (RSVD_1_EH2H_MEWS) of the EH2H_MEWS register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** EH2H_PHY_NUM_PRIMARY_MASTERS + 1
x:1	EH2H_MEWS	R	Interrupt status. **Values:** ■ 0x0 (INACTIVE): Interrupt is Inactive ■ 0x1 (ACTIVE): Interrupt is Active **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** EH2H_PHY_NUM_PRIMARY_MASTERS

Table 5-8 Fields for Register: EH2H_MEWS (Continued)

Bits	Name	Memory Access	Description
0	RSVD_EH2H_MEWS	R	This bit (RSVD_EH2H_MEWS) of the EH2H_MEWS register is reserved. It always returns 0. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.4 EH2H_COMP_PARM_1

- **Name:** Component Parameter Register 1
- **Description:** This register provides the value of the configured parameters, namely EH2H_IS_SSPLIT_CAPABLE, EH2H_CLK_MODE, EH2H_NUM_PRIMARY_MASTERS, EH2H_PHY_SBIG_ENDIAN, EH2H_PHY_SADDR_WIDTH, EH2H_PHY_SDATA_WIDTH, EH2H_PHY_MBIG_ENDIAN, EH2H_PHY_MADDR_WIDTH, EH2H_PHY_MDATA_WIDTH.
- **Size:** 32 bits
- **Offset:** 0x3f0
- **Exists:** Always

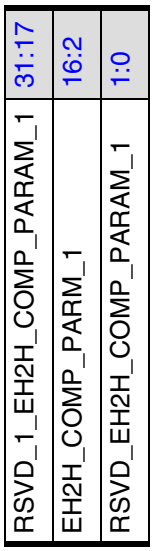


Table 5-9 Fields for Register: EH2H_COMP_PARM_1

Bits	Name	Memory Access	Description
31:17	RSVD_1_EH2H_COMP_PARAM_1	R	RSVD_1_EH2H_COMP_PARAM_1 Reserved field read-only **Value After Reset:** 0x0 **Exists:** Always

Table 5-9 Fields for Register: EH2H_COMP_PARM_1 (Continued)

Bits	Name	Memory Access	Description
16:2	EH2H_COMP_PARM_1	R	These bits specify the following values: ■ 2nd bit for IS_SSPLIT_CAPABLE - 0x0 (FALSE): EH2H is not Split Capable - 0x1 (TRUE): EH2H is Split Capable ■ 3rd bit for CLK_MODE - 0x0 (ASYNC_CLOCK_MODE): Asynchronous Clock Mode - 0x1 (SYNC_CLOCK_MODE): Synchronous Clock Mode ■ [7:4] for NUM_PRIMARY_MASTERS - 0x1 (PRIMARY_MASTERS_1): Number of primary masters is equal to 1 - 0x2 (PRIMARY_MASTERS_2): Number of primary masters is equal to 2 - 0x3 (PRIMARY_MASTERS_3): Number of primary masters is equal to 3 - 0x4 (PRIMARY_MASTERS_4): Number of primary masters is equal to 4 - 0x5 (PRIMARY_MASTERS_5): Number of primary masters is equal to 5 - 0x6 (PRIMARY_MASTERS_6): Number of primary masters is equal to 6 - 0x7 (PRIMARY_MASTERS_7): Number of primary masters is equal to 7 - 0x8 (PRIMARY_MASTERS_8): Number of primary masters is equal to 8 - 0x9 (PRIMARY_MASTERS_9): Number of primary masters is equal to 9 - 0xa (PRIMARY_MASTERS_10): Number of primary masters is equal to 10 - 0xb (PRIMARY_MASTERS_11): Number of primary masters is equal to 11 - 0xc (PRIMARY_MASTERS_12): Number of primary masters is equal to 12 - 0xd (PRIMARY_MASTERS_13): Number of primary masters is equal to 13 - 0xe (PRIMARY_MASTERS_14): Number of primary masters is equal to 14 - 0xf (PRIMARY_MASTERS_15): Number of primary masters is equal to 15 ■ 8th bit for PHY_SBIG_ENDIAN

Table 5-9 Fields for Register: EH2H_COMP_PARM_1 (Continued)

Bits	Name	Memory Access	Description
			- 0x0 (SLAVE_LITTLE_ENDIAN): Slave is Little Endian - 0x1 (SLAVE_BIG_ENDIAN): Slave is Big Endian ■ 9th bit for PHY_SADDR_WIDTH - 0x0 (SLAVE_ADDR_WIDTH_32): AHB slave address bus width is equal to 32 - 0x1 (SLAVE_ADDR_WIDTH_64): AHB slave address bus width is equal to 64 ■ [11:10] for PHY_SDATA_WIDTH - 0x0 (SLAVE_DATA_WIDTH_32): AHB slave data bus width is equal to 32 - 0x1 (SLAVE_DATA_WIDTH_64): AHB slave data bus width is equal to 64 - 0x2 (SLAVE_DATA_WIDTH_128): AHB slave data bus width is equal to 128 - 0x3 (SLAVE_DATA_WIDTH_256): AHB slave data bus width is equal to 256 ■ 12th bit for PHY_MBIG_ENDIAN - 0x0 (MASTER_LITTLE_ENDIAN): Master is Little Endian - 0x1 (MASTER_BIG_ENDIAN): Master is Big Endian ■ 13th bit for PHY_MADDR_WIDTH - 0x0 (MASTER_ADDR_WIDTH_32): AHB master address bus width is equal to 32 - 0x1 (MASTER_ADDR_WIDTH_64): AHB master address bus width is equal to 64 ■ [16:14] for PHY_MDATA_WIDTH - 0x0 (MASTER_DATA_WIDTH_8): AHB slave data bus width is equal to 8 - 0x1 (MASTER_DATA_WIDTH_16): AHB slave data bus width is equal to 16 - 0x2 (MASTER_DATA_WIDTH_32): AHB slave data bus width is equal to 32 - 0x3 (MASTER_DATA_WIDTH_64): AHB slave data bus width is equal to 64 - 0x4 (MASTER_DATA_WIDTH_128): AHB slave data bus width is equal to 128 - 0x5 (MASTER_DATA_WIDTH_256): AHB slave data bus width is equal to 256 **Value After Reset:** COMP_P_1_RESET_VAL **Exists:** Always

Table 5-9 Fields for Register: EH2H_COMP_PARM_1 (Continued)

Bits	Name	Memory Access	Description
1:0	RSVD_EH2H_COMP_PARAM_1	R	These bits (RSVD_EH2H_COMP_PARAM_1) of the EH2H_COMP_PARM_1 register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always

5.1.5 EH2H_COMP_PARM_2

- **Name:** Component Parameter Register 2
- **Description:** This register provides the value of the configured parameters namely, EH2H_WRITE_BUFFER_DEPTH, EH2H_READ_BUFFER DEPTH, and EH2H_READ_PREFETCH_DEPTH.
- **Size:** 32 bits
- **Offset:** 0x3f4
- **Exists:** Always

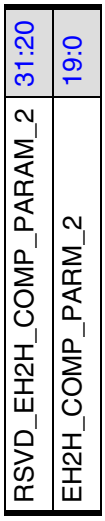


Table 5-10 Fields for Register: EH2H_COMP_PARM_2

Bits	Name	Memory Access	Description
31:20	RSVD_EH2H_COMP_PARAM_2	R	These bits (RSVD_EH2H_COMP_PARAM_2) of the EH2H_COMP_PARM_2 register are reserved. They always return 0. **Value After Reset:** 0x0 **Exists:** Always
19:0	EH2H_COMP_PARM_2	R	These bits specify the value for WRITE_BUFFER_DEPTH, READ_BUFFER_DEPTH, and READ_PREFETCH_DEPTH. ■ [7:0] for WRITE_BUFFER DEPTH ■ [15:8] for READ_BUFFER DEPTH ■ [19:16] for READ_PREFETCH_DEPTH **Value After Reset:** COMP_P_2_RESET_VAL **Exists:** Always

5.1.6 EH2H_COMP_VERSION

- **Name:** EH2H Component Version Register
- **Description:** This register provides the EH2H Component Version ID.
- **Size:** 32 bits
- **Offset:** 0x3f8
- **Exists:** Always

Table 5-11 Fields for Register: EH2H_COMP_VERSION

Bits	Name	Memory Access	Description
31:0	EH2H_COMP_VERSION	R	These bits specify the component version ID. **Value After Reset:** EH2H_COMP_VERSION **Exists:** Always

5.1.7 EH2H_COMP_TYPE

- **Name:** EH2H Component Type Register
- **Description:** This register provides information about the Component Type.
- **Size:** 32 bits
- **Offset:** 0x3fc
- **Exists:** Always

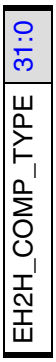


Table 5-12 Fields for Register: EH2H_COMP_TYPE

Bits	Name	Memory Access	Description
31:0	EH2H_COMP_TYPE	R	These bits specify the DesignWare Component Type number (0x44571130). This assigned unique hexadecimal value is constant and is derived from the two ASCII letters "DW" followed by a 16-bit unsigned number. **Value After Reset:** EH2H_COMP_TYPE **Exists:** Always

6 Programming the DW_ahb_eh2h

This chapter describes the programmable features of the DW_ahb_eh2h.

6.1 Programming Considerations

You should note the following guidelines when programming the three software registers: Error on Write Status Clear (EWSC), Error on Write Status (EWS), Multiple Errors on Write Status (MEWS).

- Access to local registers must always be 32 bits wide.
- Bridge slave interface does not check the size of the local register access.
- Bridge slave interface does not perform a direction check on local register access.
- Execution and ordering of local accesses is always consistent with any other read and write operation performed by the bridge. The only difference is that the transfer is terminated within the bridge and does not affect the secondary bus.

7 Verification

This chapter provides an overview of the testbench available for DW_ahb_eh2h verification. Once you have configured the DW_ahb_eh2h in coreConsultant and have set up the verification environment, you can run simulations automatically.

Note The DW_ahb_eh2h 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, refer to the following web page:
https://www.synopsys.com/dw/doc.php/doc/amba/latest/dw_amba_install.pdf

7.1 Overview of Vera Tests

The DW_ahb_eh2h peripheral incorporates numerous operational features. Many of these features, having related operational characteristics, are combined into one test to reduce simulation time. Some of the tests listed in this chapter do have some overlap.

A detailed description of each test, outlining specific transactions, appears in the README file located in the *workspace*/sim/ directory.

The DW_ahb_eh2h verification testbench performs the following set of tests.

7.1.1 test_01_random

This test verifies the overall bridge functionality with randomized stimuli. Monitors are switched off to allow simulation of longer random sequences. Up to three primary masters are used to generate random bus traffic on the primary AHB. The bridge slave is accessed in both read and write direction by all the masters. Other slaves are also present in the system. Two secondary masters are used to generate random bus traffic on the secondary AHB. The bridge master competes with those masters to access in both read and write direction secondary slaves.

Randomized parameters for the bus masters are:

- Requesting/non-requesting behavior of each master
- Locked transfers versus non-locked transfers
- Transfer attributes: address, protection, direction, burst type, size

- Transfer data (algorithmically correlated with the random address)
- Busy cycle insertion
- Incremental burst length

Randomized parameters for the bus slaves are:

- Response type (no ERROR response in the secondary AHB)
- Wait states
- Split-response/split-clear delay

This test checks:

- Data consistency across the bridge– If a secondary location has been written through the bridge, it must always be possible to read consistent data from that location.
- Deadlock – Timeout for transfer completion is implemented in each master to ensure any transfer completes within a predefined amount of clock cycles.
- AMBA protocol – Bridge slave responses are monitored with a dedicated checker.

7.1.2 test_02_random_m

This test verifies the protocol behavior of the bridge master and uses similar stimuli and checking as in the test_01_random test. It also verifies the bridge master protocol behavior using the AHB bus monitor.

7.1.3 test_03_random_c

This test is used to cover scenarios with more than three masters in the primary AHB. Up to EH2H_NUM_PRIMARY_MASTERS are used to generate random bus traffic on the primary AHB. The bridge slave is accessed in both read and write direction by all the masters. Other slaves are also present in the system. The test proceeds similarly to test_01_random and test_02_random_m.

7.1.4 test_04_random_e

This test is used to cover ERROR response scenarios and uses stimuli similar to test_01_random except that slaves in the secondary AHB are configured to generate exclusively OKAY and ERROR responses. This test runs similar checks as test_01_random, however, data consistency checks are inhibited.

7.1.5 test_20_version

This test verifies the consistency between the version ID value read from the corresponding register within the bridge, the version ID constant in the cc_constants file, and the coreKit version ID.

7.1.6 test_21_regfile

This test reads and writes all the register within the software interface of the bridge.

7.1.7 test_22_timeout

This test covers timeout scenarios where the bridge terminates a waited transfer with an ERROR response.

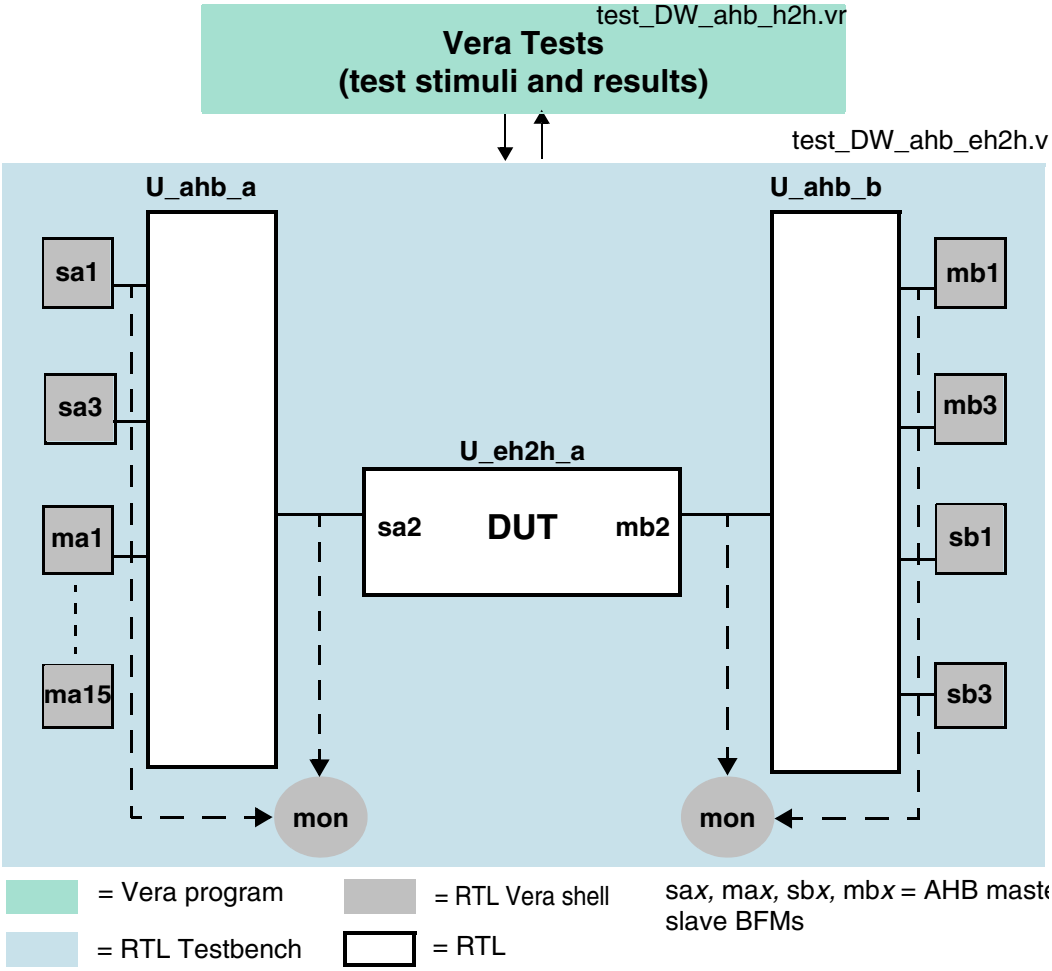
7.1.8 test_23_demo

This test produces waveforms for simple read and write transfers to show the basic behavior of the bridge master and the bridge slave.

7.2 Overview of DW_ahb_eh2h Testbench

The following diagram is a illustrates the DW_ahb_eh2h testbench.

Figure 7-1 DW_ahb_eh2h Testbench



7.2.1 Running Simulations from the Command Line

To run simulations from a UNIX command line, a simulation model must be generated through the coreConsultant GUI. In addition, all tests and test options must be configured in the Verification tab of the GUI. Then, simulations can be run as follows:

To run all tests selected in the GUI, change your working directory to DW_ahb_eh2h/sim and then execute the following command:

```
runtest.sh
```

To run single tests, change the working directory to DW_ahb_eh2h/sim and run the following:

```
runtest --simulator selected_simulator --test test_name
```

The *selected_simulator* is the one chosen in the GUI (does not work if not configured in the GUI). The *test_name* is the name of the selected test and the sub directory where the test is located. For example, to run the simple register write/read test using vcs, run the following:

```
runtest --simulator vcs --test test_reg_wr_rd
```

The results of running tests through the command line are available only in each test directory, in the test.log file.

7.2.2 Command Line Output Files

The runtest.log file is generated in *workspace*/sim/ only as a result of running simulations from coreConsultant or coreAssembler. The runtest.log file provides a pass/fail result for the particular simulation, as well as some detailed information. The test.log file located in *workspace*/sim/test_*name* is generated when tests are run from the GUI or command line, and provides more detail on each specific test simulation, in addition to the pass/fail status. The waveforms are also written to this directory, when enabled. To enable waveform generation from the command line, the switch DumpEnabled must be set as follows:

```
runtest --simulator vcs --DumpEnabled 1 --test test_reg_wr_rd
```

If the simulation results match expected results, the simulation completes successfully and the simulation status in the test.log file is PASSED. If the simulation results do not match expected results, the simulation terminates and the simulation status in the test.log file is FAILED.

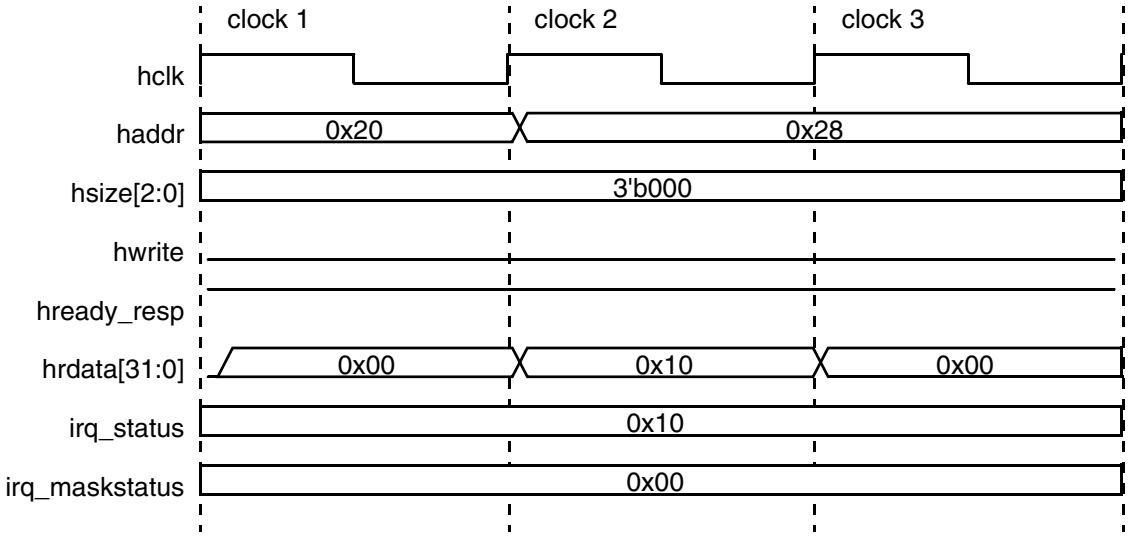
8 Integration Considerations

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

8.1 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 following figure, 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 8-1 AHB Read

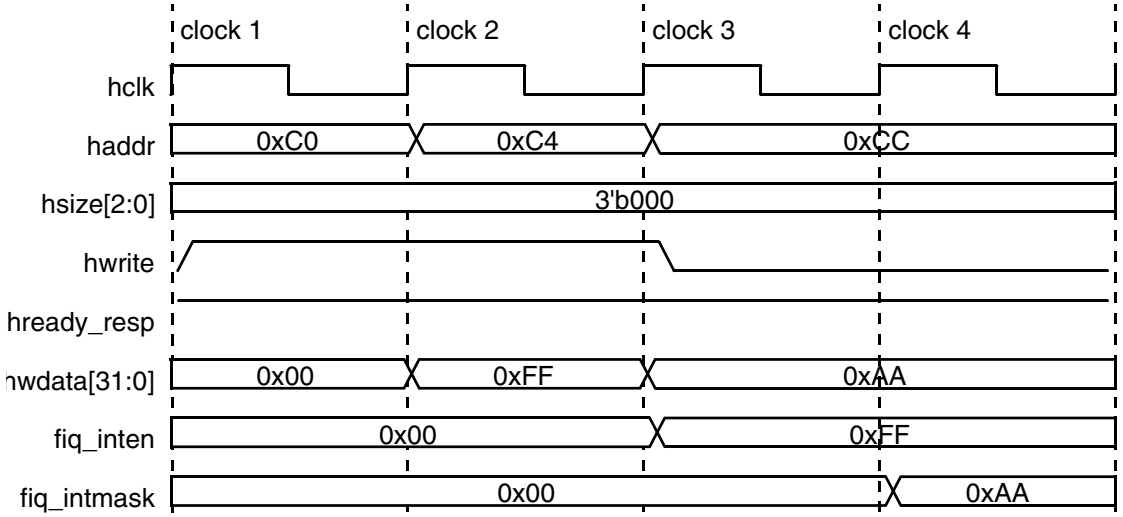


8.2 Write Accesses

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 the following figure, 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 8-2 AHB Write

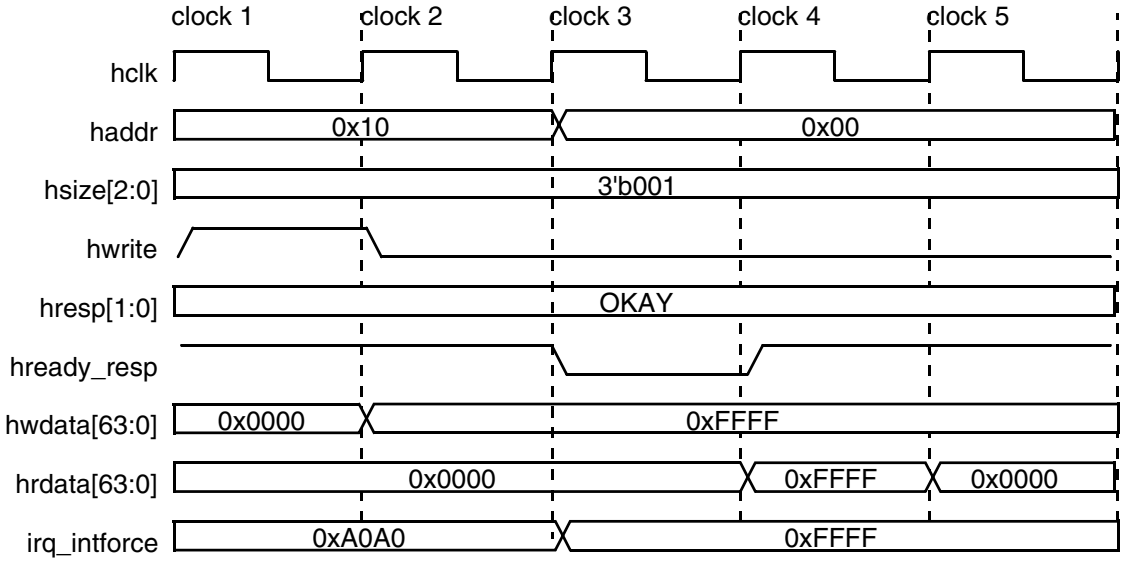


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

8.3 Consecutive Write-Read

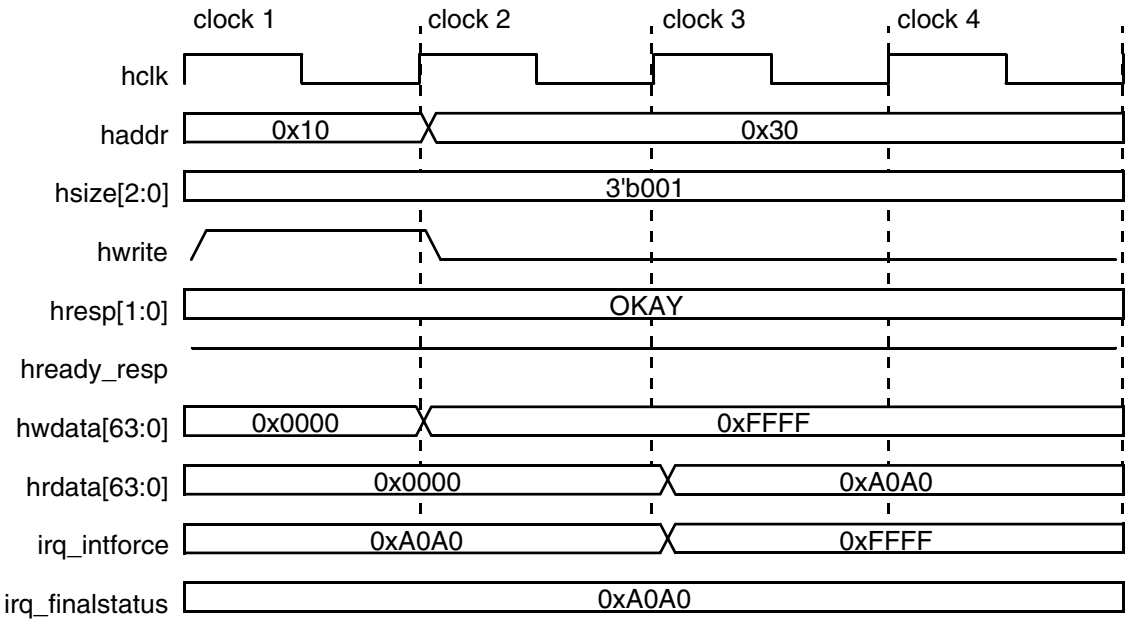
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_eh2h inserts a “wait state” when it detects a read immediately after a write to the same address. As shown in the following figure, 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 8-3 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. The following figure shows a normal consecutive write-read access.

Figure 8-4 AHB Consecutive Read/Write



8.4 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>
   ```

8.5 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_ahb_eh2h.

8.5.1 Power Consumption, Frequency, and Area Results

Table 8-1 provides information about the synthesis results (power consumption, frequency, and area) of the DW_ahb_eh2h using the industry standard 28nm technology library and how it affects performance.

Table 8-1 Power Consumption, Frequency, and Area Results for DW_ahb_eh2h Using Industry Standard 28nm Technology Library

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Default Configuration	mhclk: 300 MHz shclk: 300 MHz	10194 gates	0.172uW	35.8uW
Minimum Configuration: EH2H_PHY_SADDR_WIDTH=32 EH2H_PHY_SDATA_WIDTH=32 EH2H_PHY_SBIG_ENDIAN=0 EH2H_PHY_MADDR_WIDTH=32 EH2H_PHY_MDATA_WIDTH=32 EH2H_PHY_MBIG_ENDIAN=0 EH2H_PHY_NUM_PRIMARY_MASTERS=2 EH2H_WRITE_BUFFER_DEPTH=4 EH2H_READ_BUFFER_DEPTH=4 EH2H_READ_PREFETCH_DEPTH=1 EH2H_CLK_MODE=0 EH2H_IS_SSPLIT_CAPABLE=1 EH2H_WRITE_BUFFER_PUSH_PIPE_MODE=0 EH2H_WRITE_BUFFER_POP_PIPE_MODE=0 EH2H_READ_BUFFER_PUSH_PIPE_MODE=0 EH2H_READ_BUFFER_POP_PIPE_MODE=0	mhclk: 300 MHz shclk: 300 MHz	6218 gates	0.103uW	20.9 uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Maximum Configuration: EH2H_PHY_SADDR_WIDTH=64 EH2H_PHY_SDATA_WIDTH=128 EH2H_PHY_SBIG_ENDIAN=0 EH2H_PHY_MADDR_WIDTH=64 EH2H_PHY_MDATA_WIDTH=128 EH2H_PHY_MBIG_ENDIAN=0 EH2H_PHY_NUM_PRIMARY_MASTERS=15 EH2H_WRITE_BUFFER_DEPTH=4 EH2H_READ_BUFFER_DEPTH=4 EH2H_READ_PREFETCH_DEPTH=1 EH2H_CLK_MODE=0 EH2H_IS_SSPLIT_CAPABLE=1 EH2H_WRITE_BUFFER_PUSH_PIPE_MODE=0 EH2H_WRITE_BUFFER_POP_PIPE_MODE=0 EH2H_READ_BUFFER_PUSH_PIPE_MODE=0 EH2H_READ_BUFFER_POP_PIPE_MODE=0 EH2H_MID_WIDTH=12	mhclk: 300 MHz shclk: 300 MHz	17118 gates	0.289uW	60.7uW

A
Synchronizer Methods

This appendix describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_ahb_eh2h to cross clock boundaries.

This appendix contains the following sections:

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

A.1 Synchronizers Used in DW_ahb_eh2h

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_eh2h are listed and cross referenced to the synchronizer type in Table A-1. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-1 Synchronizers Used in DW_ahb_eh2h

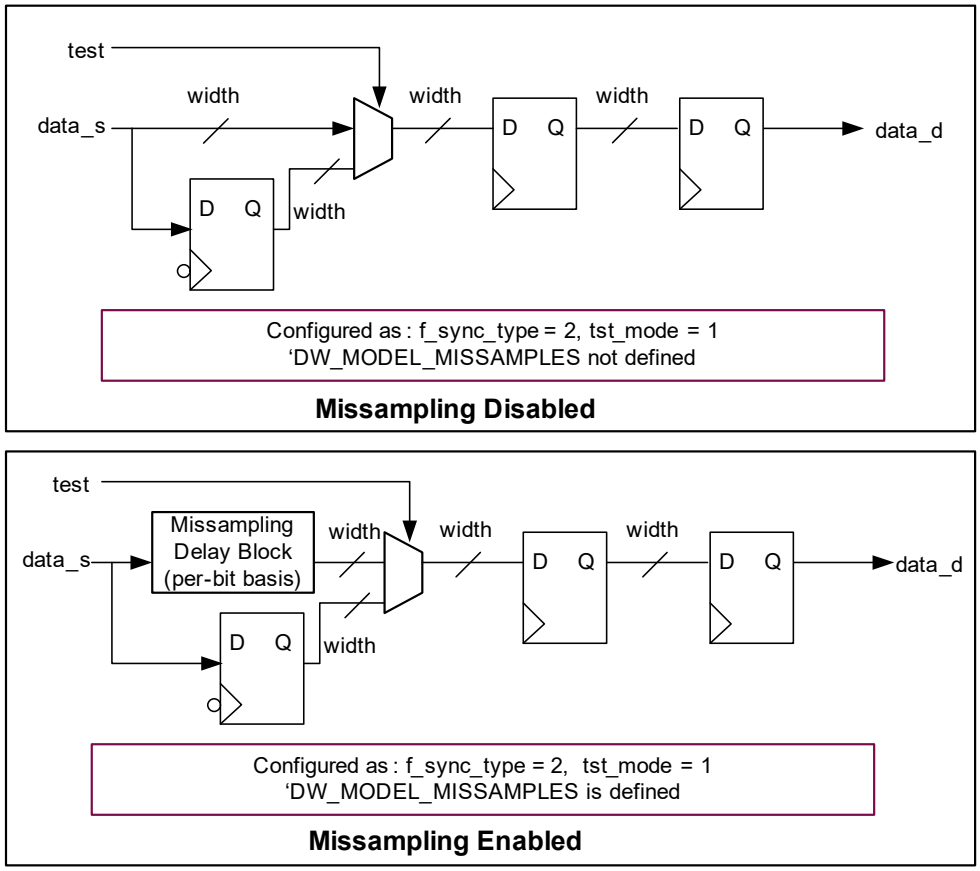
Synchronizer Module File	Sub Module File	Synchronizer Type and Number
DW_ahb_eh2h_bcm21.v		Synchronizer 1: Simple Multiple Register Synchronizer
DW_ahb_eh2h_bcm07.v	DW_ahb_eh2h_bcm05.v DW_ahb_eh2h_bcm21.v	Synchronizer 2: Synchronous dual clock FIFO Controller with Static Flags

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 Synchronizer 1: Simple Double Register Synchronizer (DW_ahb_eh2h)

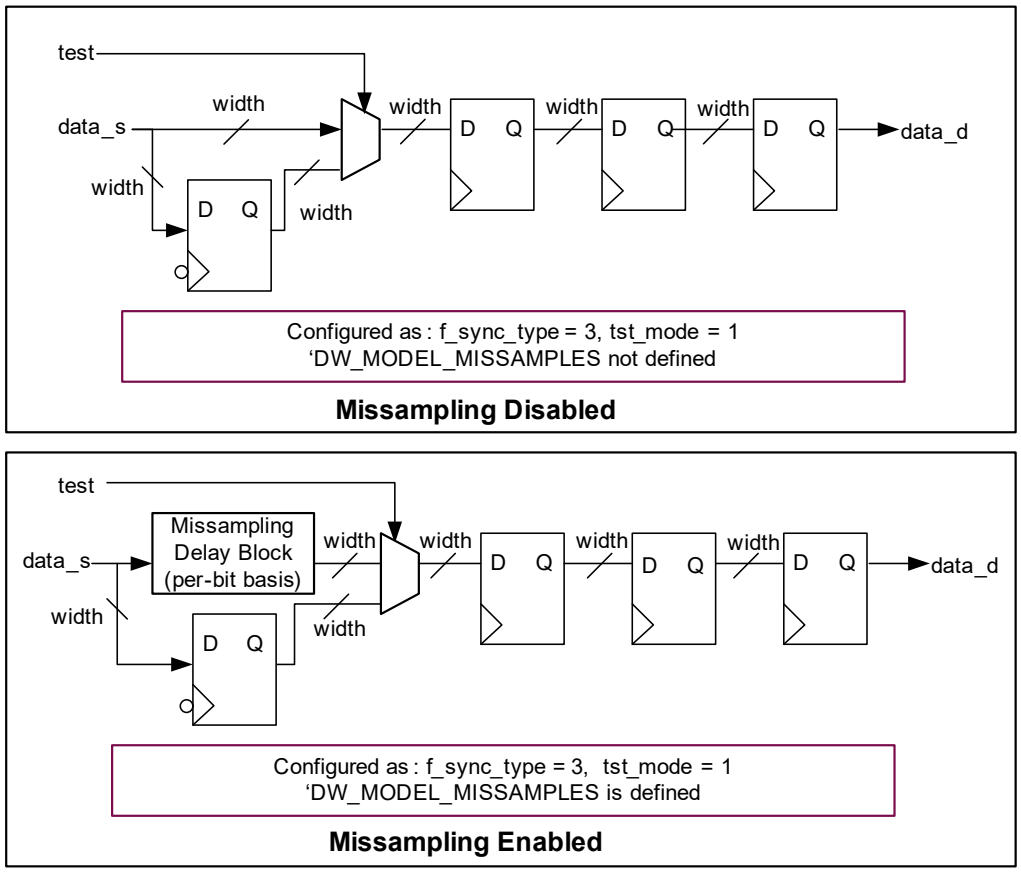
This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If shclk and mhclk are asynchronous (EH2H_CLK_MODE =0) then DW_ahb_eh2h_bcm21 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameter EH2H_SIF_SYNC_DEPTH. The following example shows the two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block Diagram of Synchronizer 1 with Two Stage Synchronization (Both Positive Edge)



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

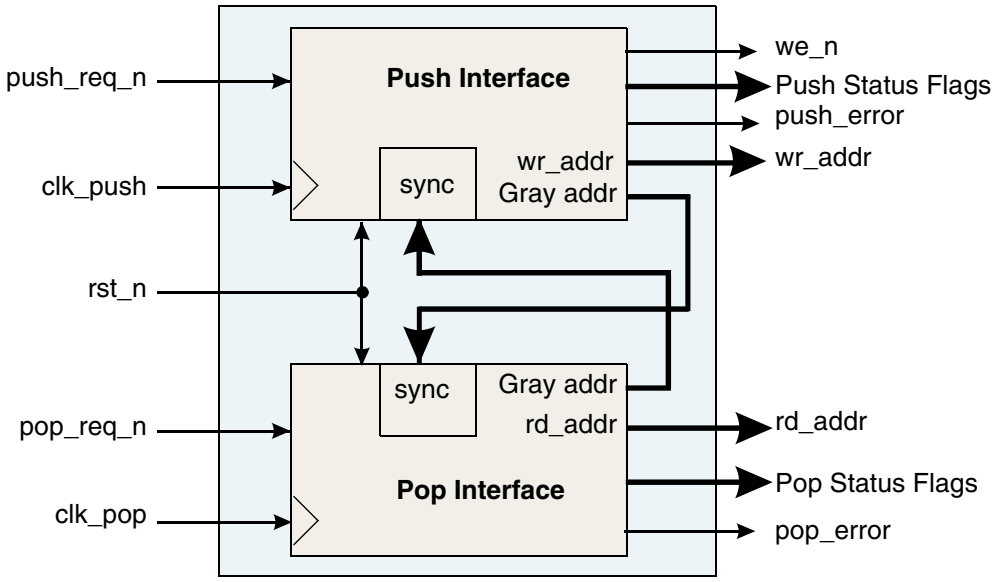
Figure A-2 Block Diagram of Synchronizer 1 with Three Stage Synchronization (Both Positive Edge)



A.3 Synchronizer 2: Synchronous (Dual-clock) FIFO Controller with Static Flags (DW_ahb_eh2h)

DW_ahb_eh2h_bcm07 is a dual independent clock FIFO RAM controller. It is designed to interface with a dual-port synchronous RAM. The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Figure A-3 shows the block diagram of Synchronizer 2.

Figure A-3 Synchronizer 2 Block Diagram



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-2 Internal Parameters

Parameter Name	Equals To
CLK_MODE	(EH2H_CLK_MODE ? 1'b1 : 1'b0)
COMP_P_1_RESET_VAL	{MDATA_WIDTH, MADDR_WIDTH, MBIG_ENDIAN, SDATA_WIDTH, SADDR_WIDTH, SBIG_ENDIAN, NUM_PRIMARY_MASTERS, CLK_MODE, SSPLIT_CAPABLE}
COMP_P_2_RESET_VAL	{(READ_PREFETCH_DEPTH-1), (READ_BUFFER_DEPTH-1), (WRITE_BUFFER_DEPTH-1)}
EH2H_COMP_TYPE	32'h44571130
EH2H_COMP_VERSION	EH2H_VERSION_ID
EH2H_VERSION_ID	32'h3131312a
MADDR_W_D_32_LOG	lg(EH2H_PHY_MADDR_WIDTH/32)
MADDR_WIDTH	MADDR_W_D_32_LOG
MAW	EH2H_PHY_MADDR_WIDTH
MBIG_ENDIAN	(EH2H_PHY_MBIG_ENDIAN ? 1'b1 : 1'b0)
MDATA_W_D_8_LOG	lg(EH2H_PHY_MDATA_WIDTH/8)

Table B-2 Internal Parameters (Continued)

Parameter Name	Equals To
MDATA_WIDTH	MDATA_W_D_8_LOG
MDW	EH2H_PHY_MDATA_WIDTH
NUM_PRIMARY_MASTERS	EH2H_PHY_NUM_PRIMARY_MASTERS
READ_BUFFER_DEPTH	EH2H_READ_BUFFER_DEPTH
READ_PREFETCH_DEPTH	EH2H_READ_PREFETCH_DEPTH
SADDR_W_D_32_LOG	lg(EH2H_PHY_SADDR_WIDTH/32)
SADDR_WIDTH	SADDR_W_D_32_LOG
SAW	EH2H_PHY_SADDR_WIDTH
SBIG_ENDIAN	(EH2H_PHY_SBIG_ENDIAN ? 1'b1 : 1'b0)
SDATA_W_D_32_LOG	lg(EH2H_PHY_SDATA_WIDTH/32)
SDATA_WIDTH	SDATA_W_D_32_LOG
SDW	EH2H_PHY_SDATA_WIDTH
SSPLIT_CAPABLE	(EH2H_IS_SSPLIT_CAPABLE ? 1'b1 : 1'b0)
WRITE_BUFFER_DEPTH	EH2H_WRITE_BUFFER_DEPTH

C
Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
AHB	Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (Arm® Limited specification).
AMBA	Advanced Microcontroller Bus Architecture — a trademarked name by Arm® Limited that defines an on-chip communication standard for high speed microcontrollers.
APB	Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (Arm® Limited specification).
APB bridge	DW_apb submodule that converts protocol between the AHB bus and APB bus.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
arbiter	AMBA bus submodule that arbitrates bus activity between masters and slaves.
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.

bus bridge	Logic that handles the interface and transactions between two bus standards, such as AHB and APB. See APB bridge.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.
core	Any configurable block of synthesizable IP that can be instantiated as a single entity (VHDL) or module (Verilog) in a design. Core is the preferred term for a big piece of IIP. Anything that requires coreConsultant for configuration, as well as anything in the DesignWare Cores library, is a core.
core developer	Person or company who creates or packages a reusable core. All the cores in the DesignWare Library are developed by Synopsys.
core integrator	Person who uses coreConsultant or coreAssembler to incorporate reusable cores into a system-level design.
coreAssembler	Synopsys product that enables automatic connection of a group of cores into a subsystem. Generates RTL and gate-level views of the entire subsystem.
coreConsultant	A Synopsys product that lets you configure a core and generate the design views and synthesis views you need to integrate the core into your design. Can also synthesize the core and run the unit-level testbench supplied with the core.
coreKit	An unconfigured core and associated files, including the core itself, a specified synthesis methodology, interfaces definitions, and optional items such as verification environment files and core-specific documentation.
cycle command	A command that executes and causes HDL simulation time to advance.
decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
Design View	A simulation model for a core generated by coreConsultant.
DesignWare Synthesizable Components	The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs.

DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the 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.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
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.
subsystem	In relation to coreAssembler, highest level of RTL that is automatically generated.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
workspace	A network location that contains a personal copy of a component or subsystem. After you configure the component or subsystem (using coreConsultant or coreAssembler), the workspace contains the configured component/subsystem and generated views needed for integration of the component/subsystem at the top level.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index

V

W

Z