SYNOPSYS®

DesignWare® DW_ahb

Databook

DW_ahb - Product Code

2.15a
December 2020

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.
www.synopsys.com

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 2.06b onward.

Version	Date	Description
2.15a	December 2020	**Added:** ■ "AMBA 5 AHB Features" on page 36 ■ "Basic Core Module (BCM) Library" on page 125 **Updated**: ■ Version changed for 2020.12a release ■ "Performance" on page 120 ■ "Verification" on page 111 ■ Parameter Descriptions, Register Descriptions, Signal Descriptions, and Internal Parameter Descriptions are auto-extracted with change bars from the RTL **Removed**: ■ Index chapter
2.14a	July 2018	**Updated**: ■ Version changed for 2018.07a release ■ "Performance" on page 120 ■ Parameter Descriptions, Register Descriptions, Signal Descriptions, and Internal Parameter Descriptions are auto-extracted with change bars from the RTL **Removed**: ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.

Version	Date	Description
2.13a	October 2016	■ Version changed for 2016.10a release ■ Parameter Descriptions and Register Descriptions 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 ■ Added an entry for the xprop directory in and Table 2-4 ■ Added “Running VCS XPROP Analyzer” ■ Added“Non-Standard Master ID Sideband Signal” on page 33 ■ Updated, “Features” on page 18, Figure 1-2 on page 18, and Figure 7-1 on page 113 to update number of AHB slaves to 31 ■ Updated area and power numbers in Table 8-1, Table 8-2, Table 8-3, and Table 8-4 ■ Added a note in “Arbitration Schemes” on page 22 and “Master Priority Level Registers” on page 107 ■ Moved Internal Parameter Descriptions to Appendix
2.12a	June 2015	■ Added “Running SpyGlass® Lint and SpyGlass® CDC” ■ Added “Running Spyglass on Generated Code with coreAssembler” ■ Corrected description of AHB-Lite support. ■ “Signal Descriptions” on page 73 auto-extracted from the RTL. ■ Added “Internal Parameter Descriptions” on page 127 ■ Added a note in “Early Burst Termination” on page 28 to indicate the condition where arbiter can remove the grant within two cycles. ■ Updated “Weighted-Token Arbitration” on page 29 to clarify grant removal timing when all tokens from the master are consumed ■ Included Legal Value for 64-bit in “Parameter Descriptions” on page 41. ■ Register and field names for following modified: - PLi changed to AHB_PLi - EBTCOUNT changed to AHB_EBTCOUNT - EBT_EN changed to AHB_EBT_EN - EBT changed to AHB_EBT - DFT_MST changed to AHB_DFLT_MASTER - WTEN changed to AHB_WTEN - AHB_CL_M(i) changed to AHB_CCLM(i) - AHB_COMP_VERSION changed to AHB_VERSION_ID ■ Reset values of the registers AHB_EBT, AHB_DFLT_MASTER, AHB_TCL, and AHB_CCLM(i) updated to match with the actual values

Version	Date	Description
2.11a	June 2014	■ Version change for 2014.06a release. ■ Updated “Performance” section in the “Integration Considerations” chapter ■ Corrected Default Input/Output Delay in the following Signal groups: - Master Signals - Slave Signals - Control Signals - External Decoder Signals - Interrupt Signals
2.10c	May 2013	■ Version change for 2013.05a release ■ Updated the template.
2.10b	October 2012	Added the product code on the cover and in Table 1-1
2.10b	November 2011	Version change for 2011.11a release
2.10a	October 2011	Revised paragraph for Second Tier Arbitration explaining possibility of master being “starved” by bus
2.09a	June 2011	Updated system diagram in Figure 1-1 Enhanced “Related Documentation” section in Preface
2.09a	September 2010	Corrected names of include files and vcs command used for simulation
2.08a	December 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks.
2.08a	May 2009	Removed references to QuickStarts, as they are no longer supported.
2.08a	October 2008	■ Updated description of DFT_MST register ■ Version change for 2008.10a release
2.07a	June 2008	Version change for 2008.06a release
2.06b	June 2007	Version change for 2007.06a release

Preface

This databook provides information about the DesignWare® Advanced High-performance Bus (DW_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.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_ahb.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_ahb signals.
- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_ahb.
- Chapter 6, "Programming the DW_ahb" provides information needed to program the configured DW_ahb.
- Chapter 7, "Verification" provides information on verifying the configured DW_ahb.
- Chapter 8, "Integration Considerations" provides getting started information that allows you to walk through the process of using DW_ahb with coreConsultant.
- Appendix A, "DesignWare Synthesizable Component Constants" includes the contents of the DesignWare Synthesizable Components bus constants file.
- Appendix B, "Basic Core Module (BCM) Library" provides information about the BCMs used in DW_ahb.
- Appendix C, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.
- Appendix D, "Glossary" provides a glossary of general terms.

Related Documentation

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

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

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

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

    File > Build Debug Tar-file

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

  a. https://solvnetplus.synopsys.com

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

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

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

Ensure to include the following:

- **Product L1:** DesignWare Library IP
- **Product L2:** AMBA

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

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

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

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

Product Code

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

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

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

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

Component Name	Description
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI fabric
DW_axi_hmx	Configurable high performance interface from and AHB master to an AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI subsystems
DW_axi_x2h	Bridge from AMBA 3 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
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

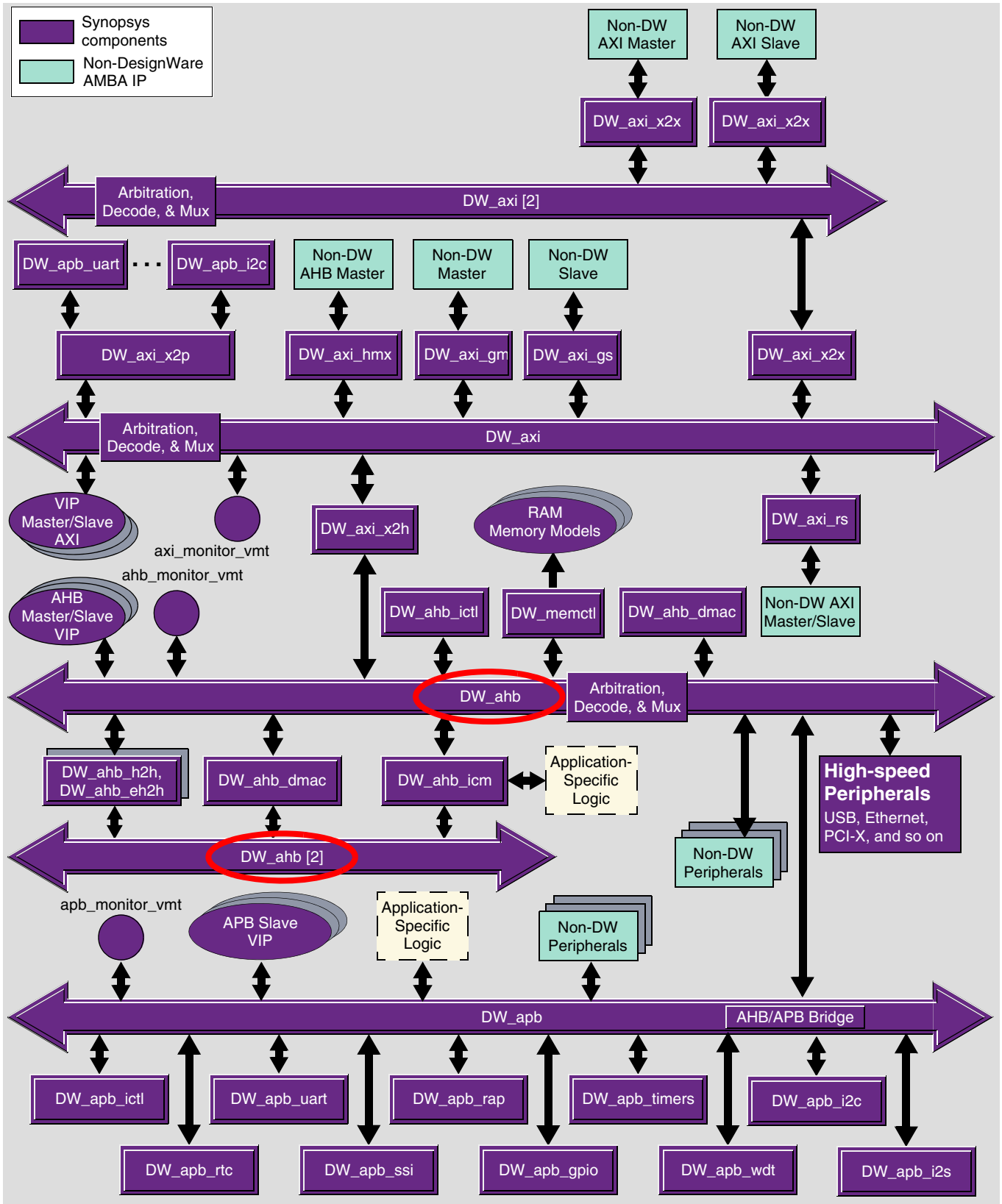
This chapter provides a basic overview of the DesignWare DW_ahb, which is a programmable Advanced High-performance Bus (AHB).

1.1 DesignWare System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA version 3.0-compliant AXI (Advanced eXtensible Interface) components.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. To access the product page and documentation for AMBA components, see the DesignWare IP Solutions for AMBA Interconnect page. (SolvNetPlus ID required)

Figure 1-1 Example of DW_ahb in a Complete System



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

1.2 General Product Description

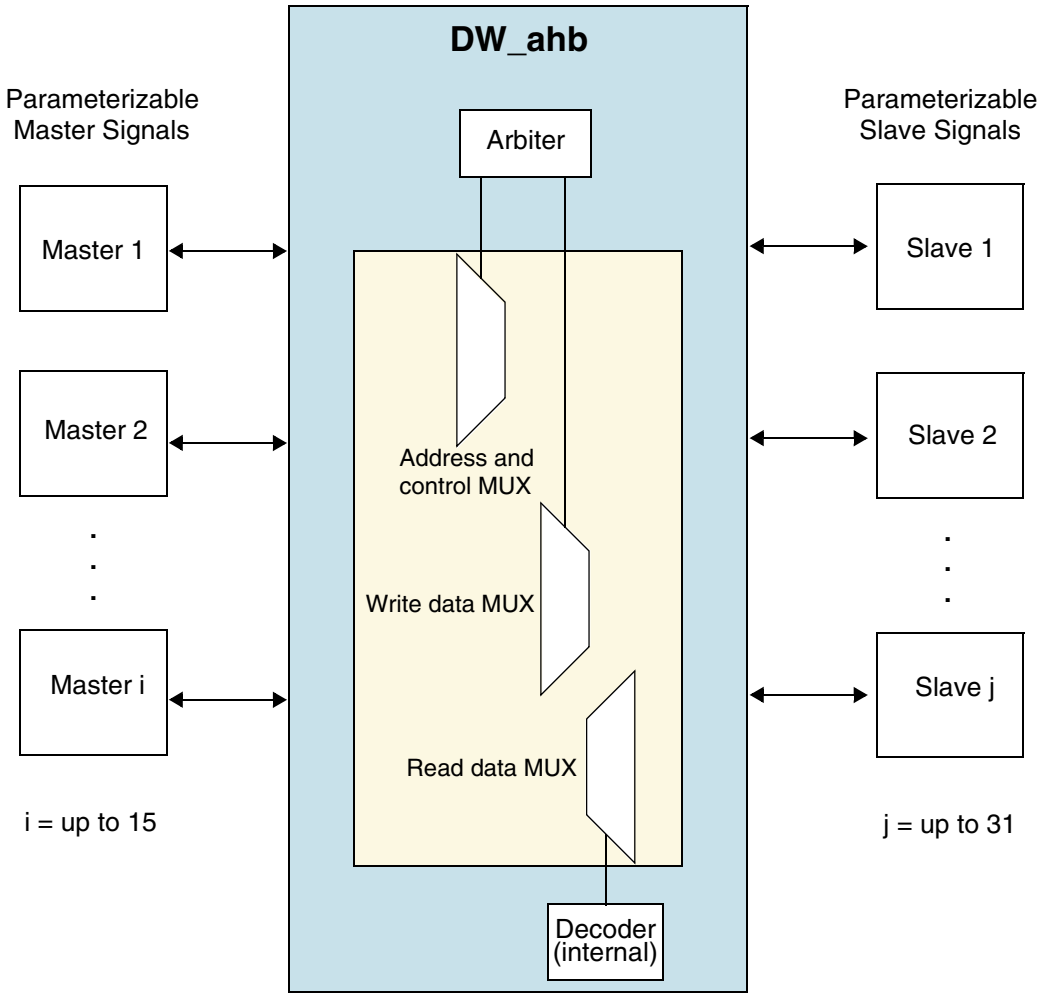
The Synopsys DW_ahb conforms to the *AMBA Specification, Revision 2.0* from Arm®.

1.2.1 DW_ahb Block Diagram

As shown in Figure 1-2, the DW_ahb consists of the following components:

- "Arbiter" – The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. The arbiter contains an optional programmable slave interface for allocating priorities, selecting the default master, allocating tokens to masters, and enabling early burst termination. If the slave interface is *not* included in the design, the following occurs:
  - Weighted-token arbitration is excluded
  - Early burst termination is disabled
  - Priorities are fixed and hardcoded
- "Optional Internal Decoder" – You can choose to include an AHB internal decoder, which is used to decode the address of each transfer and to generate a select signal for the slave that is involved in that transfer. By having the internal decoder, the DW_ahb needs to supply the addresses. Overlapping regions are checked when the decoder is being configured.
- "Optional External Decoder" – You can choose to use an external decoder. By having the decoder external to the DW_ahb, users can connect any decoder with any number of remap options.
- "Multiplexer" – All addresses, data, and control signals from each master are multiplexed.

Figure 1-2 DW_ahb Block Diagram



1.3 Features

The DW_ahb supports the following features:

- Compliance with the *AMBA Specification (Rev. 2.0)*
- Configuration of DesignWare AHB Lite system
- Configuration of up to 15 masters in a non-AMBA DesignWare Lite system
- Configuration of up to 31 slaves
- Configuration of data bus width of 8, 16, 32, 64, 128, or 256 bits
- System address width of 32 or 64 bits
- Configuration of system endianness — Big or little endian; can be controlled by external input or set during configuration of component
- Optional arbiter slave interface
- Optional internal decoder

- Programmable arbitration scheme:
  - Weighted token
  - Programmable or fixed priority
  - Fair-Among-Equals
- Arbitration for up to 15 masters
- Individual grant signals for each master
- Support for split, burst, and locked transfers
- Optional support for early-burst termination
- Optional support for non-standard Master ID sideband signal in AHB-Lite Mode
- Configurable support for termination of undefined length bursts by masters of equal or higher priority
- Configurable or programmable priority assignments to masters
- Disabling of masters and protection against self disable
- Optional support for memory remap feature
- Optional support for pausing of the system, immediately or when bus is IDLE
- Contiguous and non-contiguous memory allocation options for slaves
- External debug mode signals, giving visibility
- Complies with AMBA 5 AHB protocol specification
  - Extended memory types
  - Secure transfers
  - Exclusive transfers
  - Multiple Slave Select signals support on a single slave interface
  - User Signals support on each channel

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

1.4 Standards Compliance

The DW_ahb 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 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 111 chapter discusses the specific procedures for verifying the DW_ahb.

1.6 Licenses

Before you begin using the DW_ahb, 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 component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, refer to *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_ahb 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 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

The DW_ahb provides the AHB bus fabric to connect AHB masters and slaves to form part of a System-on-Chip (SoC) bus solution.

There is an option to configure AHB Lite, which is a subset of the full AHB design where only a single bus master is used. AHB Lite is the DesignWare implementation of AMBA 2.0 AHB Extensions AHB-Lite. The DesignWare AHB Lite configuration,

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

- AHB_LITE mode is not compliant with the AMBA 3 AHB-Lite Protocol.
- Instead of a HMASTLOCK signal, a HLOCK_M1 signal is used for locked transfers.

2.1 DW_ahb Components

This section describes the functions of the following DW_ahb components:

2.1.1 Arbiter

The DW_ahb allows only one bus master to have access to the bus at any given time. Each master can request control of the bus; however, you must decide which master is going to gain access first by specifying the priority level of each master through coreConsultant. The *AMBA Specification (Rev. 2.0)* does not specify an arbitration scheme.

2.1.1.1 Arbitration Schemes

If DW_ahb is specified as an AHB Lite system, then there is no arbitration scheme. There is only one master in an AHB Lite system, and it never has to request the bus as it is always granted. There is no need for priority levels or a default master.

The arbiter supports up to 16 priority levels (0 to 15)—0 is the disabled setting, 1 is the lowest priority, and 15 is the highest—so that each master can be assigned a separate priority. Masters can be masked from the arbitration scheme. Requests are masked according to protocol requirements, such as split-slave transactions or a programmed priority of 0.

Attention

Priority schemes for arbitration are opposite in DW_ahb and DW_ahb_icm:

- In DW_ahb, higher the priority value, higher the priority of the Master.
- In DW_ahb_icm, lower the priority value, higher the priority of the layer.

When DW_ahb is configured in coreConsultant, each master is assigned an initial priority. By default, each master is configured with a priority level synonymous with its master number. For instance, master 1 receives a priority of 1, master 2 has a priority of 2, and so on. Additionally, the priority level can be set so that it is programmable through a read/write register. For more information about the priority levels, see "Master Priority Level Registers" on page 107.

- **First Tier Arbitration** – When two or more masters request the bus at the same time, the requesting master with the highest priority is granted the bus.
- **Second Tier Arbitration** ("Fair Among Equals") – If two requesting masters have the same priority, then ownership is based on a "Fair-Among-Equals" algorithm. This algorithm compares each master at every clock cycle. When a master is granted access to the bus, DW_ahb holds the grant for two clock cycles (providing hready is high). Then it checks which master is requesting access and re-arbitration occurs.

  There is also a possibility that some wait states occur before the bus is granted to the next master. For instance, one master can be granted ownership of the bus for several accesses before the arbiter grants the bus to the next master. A fixed-length burst could continue into its SEQ portion by taking advantage of this extra grant cycle, allowing the burst to complete. Arbitration is locked during the fixed-length burst until beat 7 (assuming INCR8) and hready is high to allow back-to-back transfers on the bus.

  With this scheme, there is the slight possibility that a master can get "starved" from the bus, such as when there are three masters vying for access to the bus. Assuming there are no wait states, the bus grants ownership as follows: M1-M3-M2-M1. In this example, the arbiter does not have a way of stopping the updating of the Fair-Among-Equals arbitration even when DW_ahb does not take the result from the arbiter—it still cycles through its Fair-Among-Equals algorithm. In this type of scenario, there is a possibility that a master (for example, M2) does not get granted ownership if the number of clients and wait states is of a combination that the Fair-Among-Equals algorithm (which is

running free) is not arbitrating because every time the arbiter is polled to produce the grant, the same master (for example, M1) surfaces again.

- **Weighted Token Arbitration**– If DW_ahb is specified with the weighted-token arbitration scheme, then each master in the system is allocated an initial number of clock tokens. This scheme is an extension of the default arbitration scheme (first and second tiers). Additionally, the overall length of the arbitration period needs to be defined. Masters use tokens to compete for the bus, based on priorities using the upper tier arbitration.

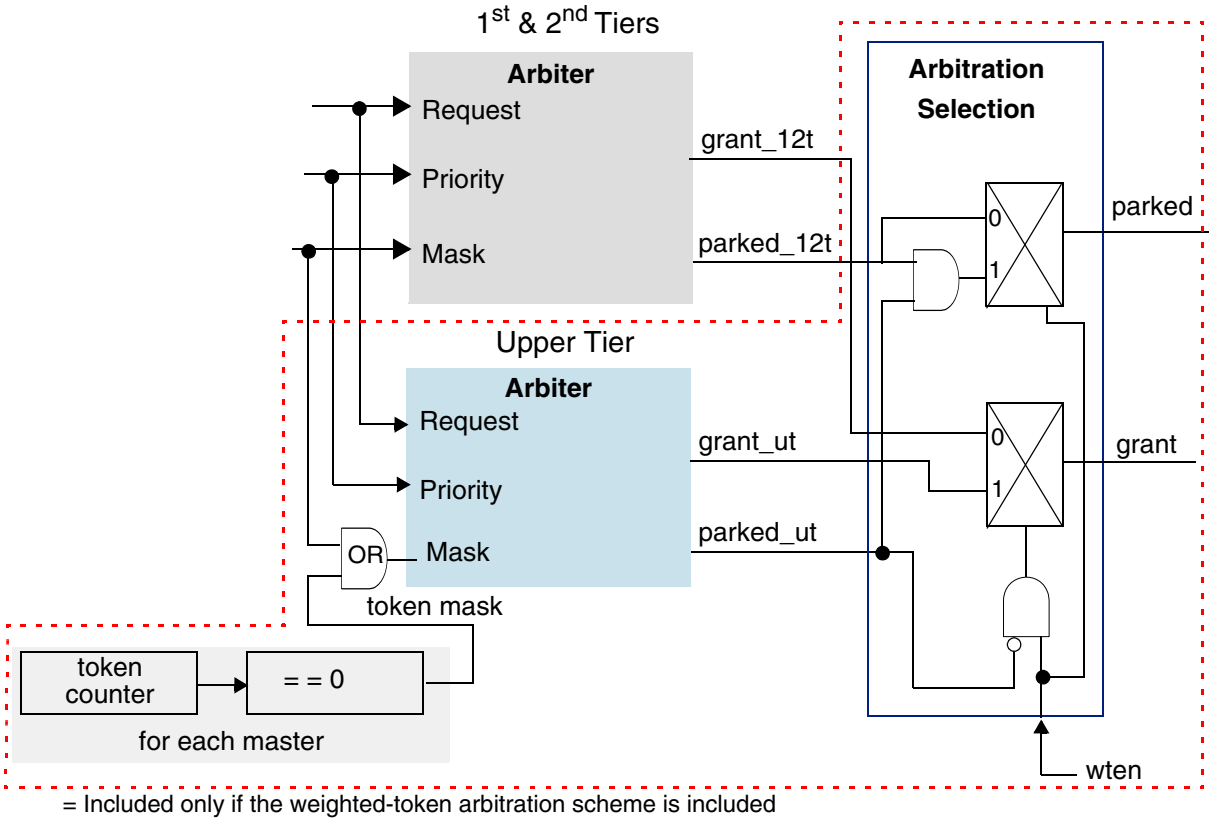
  A master can have both zero and infinite tokens at the same time. Each master is assigned a number of clock tokens that it can use and be guaranteed to get this number of cycles over an arbitration period. Masters with remaining tokens have priority over masters that have used all of their tokens. User-configured token values are summed—by adding 16 to the priority of a master when it has tokens left—to ensure that they do not exceed the total allocated number of tokens.

  The adjusted priority reduces by 16 or reverts to the configured priority when the tokens are used up. A master with all of its tokens used is granted the bus when there is no master with tokens requesting the bus. You can specify any number of tokens for a master. The larger the value, the more the number of tokens. To facilitate an infinite number of tokens, the value of 0 represents infinite tokens. For more information, see the configuration parameters for weighted-token arbitration in "Parameter Descriptions" on page 41.

Note It is not possible to configure a priority of 0 in coreConsultant. But when the master priorities are not hardcoded, they can be programmed through software. It is recommended that masters be configured with different priorities so that only the first tier of arbitration is required, plus the upper tier when the weighted-token arbitration scheme is enabled. The second tier, Fair-Among-Equals, is a cycle-by-cycle comparison of the requesting masters.

These arbitration schemes are illustrated in Figure 2-1.

Figure 2-1 Arbitration Scheme in DW_ahb



1st tier =	Master with highest priority wins ownership of bus if two or more masters are requesting access to bus.
2nd tier =	If two requesting masters have the same priority, then ownership is based on a Fair-Among-Equals scheme.
Upper tier =	If weighted-token arbitration is enabled, each master in the system is allocated a number of clock tokens that are required to gain access to the bus. When competing masters have unused tokens, they compete based on priority level.

The granting of masters for fixed-length burst masters is not every cycle. Therefore, the fairness depends on the number of masters and wait states, and on each cycle calculating the next master. If the system is not ready, then another master is granted in the following cycle. The master that wins ownership is the master that is granted when the bus is ready. All masters will eventually be granted ownership.

Figure 2-2 shows the timing for fixed-length bursts with multiple masters requesting ownership. When masters have different priorities, the bus ownership is highest down to lowest. When masters have equal priorities, the bus ownership is random.

Figure 2-2 Bursts with Specified Length

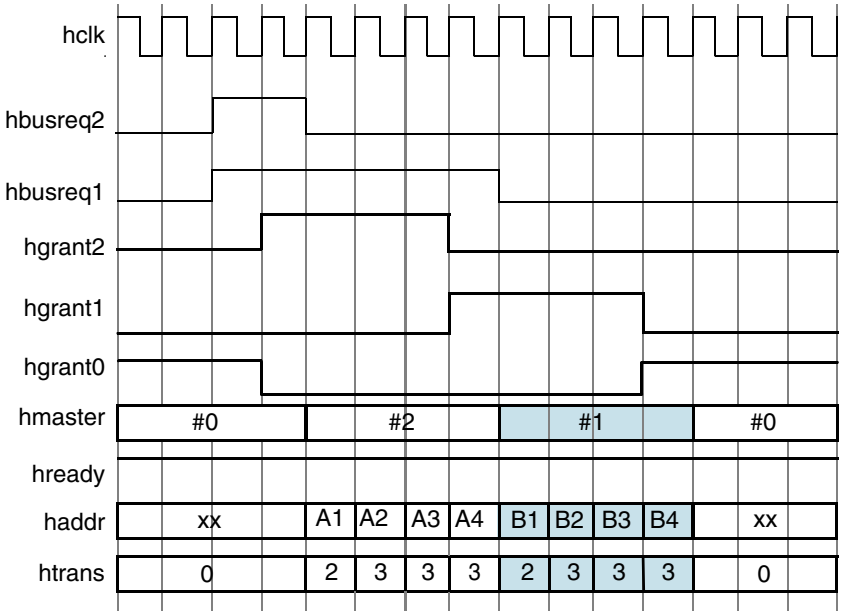
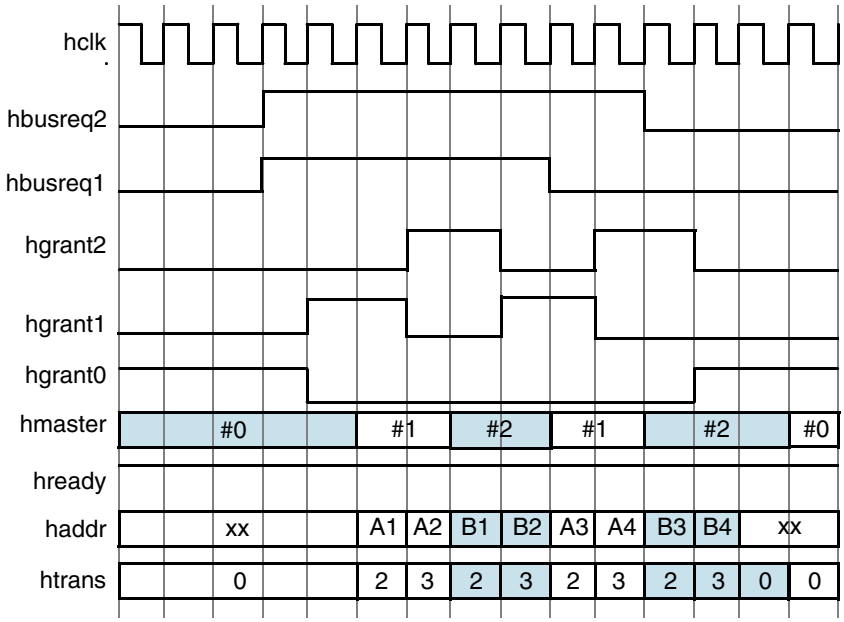


Figure 2-3 shows the timing that occurs when masters with undefined length bursts and the same priority level are early terminated by masters of equal or higher priority.

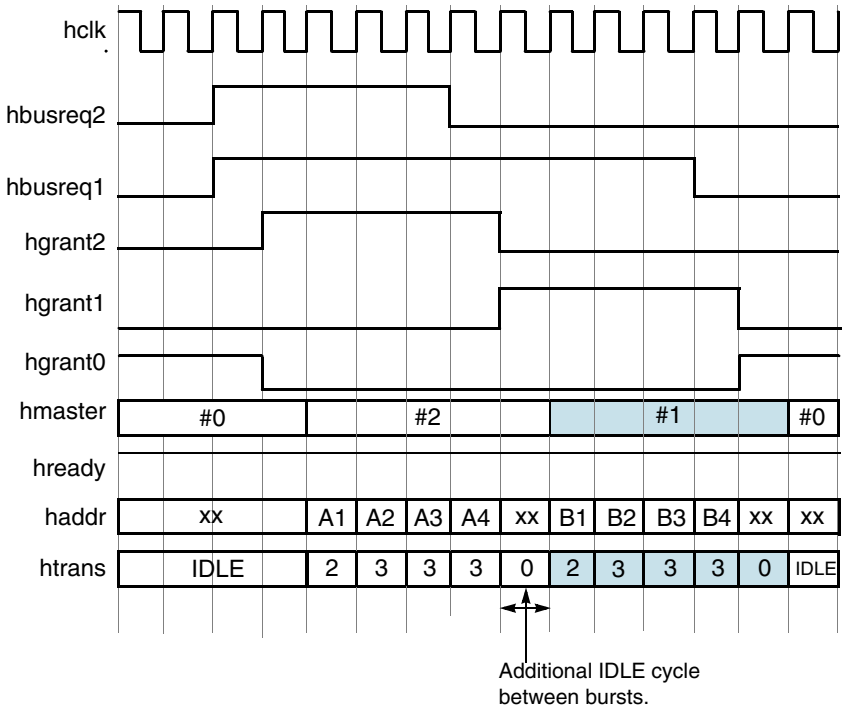
Figure 2-3 Bursts with Unspecified Length and Equal Priority Level



In Figure 2-3, there are two masters requesting ownership of the bus with the same priority. When a master is granted access to the bus, it is for a minimum of two clock cycles. The arbiter then selects one of the masters, and bus ownership can change. Burst transfers that are of an unspecified length may be early burst terminated by grants to requests from masters of equal priority or greater priority. For bursts of this type, changing grants is calculated at every cycle.

Figure 2-4 shows the timing for bursts of an unspecified length from masters with different priority levels. The highest priority wins ownership of the bus each time. There is an additional IDLE cycle between bursts, because the request line has to be held until the last transfer has started.

Figure 2-4 Bursts with Unspecified Length and Different Priority Level



2.1.1.2 Transfers

The DW_ahb supports many types of transfers, including split and locked transfers. If a split-capable slave is not able to complete a transfer as requested, it can issue a split response to its master. In this case, the arbiter may grant ownership of the bus to another requesting bus master through the normal method of arbitration. The master who received the split response will not be granted bus access until the slave indicates it is ready to resume the split transfer.

A master that has been split cannot begin another transfer on the bus until the original transfer has been completed because it is masked from arbitration. When a slave indicates to the arbiter that the split transfer can resume, the master is allowed to compete under the normal arbitration procedure. The master must complete the same transfer.

A slave may issue a retry response if it is not capable of completing a transfer. In this case, the arbiter begins arbitration again to grant access to the bus, but masters with a lower priority than the current master are masked out. This is in contrast to the split response where all other requesting masters may compete for access to the bus.

By asserting its hlock signal, the master indicates to the arbiter that the current transfer is locked, meaning that no other master can be allowed access to the bus until the current transfer has completed.

For an AHB Lite configuration, slaves must not produce SPLIT/RETRY responses. The lock functionality is still required because a master might be performing a transfer to a multi-port slave. The slave must be given an indication that no other transfer should be accepted by the slave when the master is given locked access.

2.1.1.3 Arbiter Slave Interface

The arbiter slave interface is an optional AHB slave over which the internal registers of DW_ahb may be read from and written to by any master in the system. The arbiter slave interface is activated by enabling the AHB_HAS_ARBIF parameter.

The arbiter supports little- or big-endian systems. Access to the registers can be 8, 16, or 32 bits for a 32-bit system; 8 or 16 for a 16-bit system; or 8 bits for an 8-bit system. All other transfer sizes are illegal. The internal registers are:

- Master priorities
- Default master
- Early burst termination
- Weighted token registers
- coreKit version ID

For more information on these registers, see "Arbiter Slave Interface Registers" on page 107.

As mentioned previously, it is possible to disable a master by programming its priority to zero. To protect a master from disabling itself, the DW_ahb prevents the master from writing zero into its own priority register.

When the system is configured as AHB Lite, the arbiter slave interface is not available.

2.1.1.4 Default Master versus Dummy Master

The *default master* is the master that is granted ownership of the bus when no masters are requesting it. The *dummy master* is the master that takes ownership of the bus when none of the masters can. The default master can be programmed to be any of the masters within the system, including the dummy master. When weighted-token arbitration is enabled, the default master is the dummy master.

There are two cases when the dummy master is granted ownership of the bus: (1) when no master can be granted ownership (for example, when the default master has been split) or (2) if any master is subject to Early Burst Termination. The dummy master, designated as master 0, never requests ownership of the bus. The dummy master drives IDLE cycles when it is granted ownership.

When the system is configured as AHB Lite, the default master number is changed to 1.

2.1.1.5 Hard-coded Default Master

Optional Feature. The ID number (index) of the default master can be hard-coded through coreConsultant. In this case, the register for the default master is read-only and cannot be changed. For more information, see "Default Master Register" on page 109. If there is no arbiter slave interface, the ID number of default master is hardcoded.

2.1.1.6 Pause Mode

Optional Feature. When the system is in pause mode, the dummy master owns the bus. Requests from masters are ignored until the system exits pause mode, which is whenever an interrupt occurs. This requires the DW_apb_rap module or similar functionality to be implemented in order to exit Pause Mode. By default, the arbiter Pause Mode feature is enabled in coreConsultant, which means that the design includes this functionality. If Pause Mode is not enabled, you have no method for pausing the system arbitration other than ensuring that all masters are idle. When the system is configured as AHB Lite, pause mode is disabled.

2.1.1.7 Delayed Pause Action

Optional Feature. When the delayed pause action is set during configuration, the pause signal, when set, does not take effect until hready is high and htrans is IDLE. If this action is not set, the DW_ahb enters pause mode at the next hclk edge once the pause signal is set. By delaying the action on pause, any other transfers on the bus can be completed before the system is paused.

2.1.1.8 Early Burst Termination

Optional Feature. This feature is enabled when the arbiter slave interface has been included in the design. Early-burst termination (EBT) makes it possible to terminate a burst early when a master holds onto the bus for too many cycles. When this feature is enabled using the EBTEN parameter, it forces the master to begin arbitration of the bus again and to rebuild the burst to complete the transfer. When a programmable cycle count is exceeded, bus control is handed over to the dummy master. The dummy master owns the bus for one cycle, then normal arbitration continues. When this feature is enabled, the DW_ahb includes the Early Burst Termination (EBT) registers, which can be programmed to determine the number of cycles a transfer can take.

When a burst is early-burst terminated, an interrupt (ahbarbint) is set and a status register needs to be read to clear it. Furthermore, locked transfers cannot be early-burst terminated.

The EBTEN parameter inserts logic so that no master can hold the bus for a long period of time during a burst; EBTEN is controlled by software using a counter. For example, a low-priority master might perform a burst on the bus, which takes more cycles to complete than the counter in the EBTEN logic allows; this is like an accumulation of wait states for each beat going over a maximum threshold. At this time, the burst is terminated, and the bus is given to the dummy master for one cycle. Normal arbitration then resumes, allowing a higher-priority master to gain access. If this logic is not included in the DW_ahb, then no bursts are terminated in this manner.

The AMBA specification uses command pipelining in order to increase bus utilization. Pipelining sometimes requires an EBT. For example, suppose a low-priority master starts a fixed-length, 16-beat burst, and a higher-priority master requests the bus during this burst. Before the end of the 16-beat burst, the change in grant from the lower to the higher-priority master must occur. If this grant change does not occur before the end of the burst, there is a period of time when there is an IDLE cycle on the bus, which decreases the utilization of the bus. The AMBA specification requires this early change in grant to efficiently use all cycles on the bus. If the 16-beat burst finishes correctly, there is no problem and no EBT. However, if the master performing the 16-beat burst inserts busy cycles between the last two beats of the burst, the transfer is early-terminated because the grants have changed; the newly granted master is already starting its next transfer.

When early-burst termination is not enabled—that is, when EBTEN is set to 0—it is possible that an EBT may still occur. Examples are:

- Unspecified-length INCR transfers when a higher priority master requests the bus (only when configuration parameter AHB_FULL_INCR is false).
- Busy cycles between the last two beats of a fixed-length burst.

When designing the arbiter within the DW_ahb, attempts were made to reduce these EBTs. However, eliminating all EBTs would have reduced the bus utilization. The DW_ahb arbiter grants a master the bus for a minimum of two cycles whenever it is granted the bus, provided hready is active. This allows a master to start a transfer, and in the case of a burst transfer, get to the SEQ part of the burst. As arbitration is broken over an IDLE or an NSEQ, this makes the arbiter less noisy and eliminates some initial EBT conditions. If a higher-priority master requested the bus one cycle after the lower-priority master, then the lower-priority master would be allowed to complete its burst before the higher-priority master would be given access.

Note DW_ahb can de-assert the grant assigned to a master before two cycles when the master that is granted does not initiate any transfer and a higher priority master is requested at the same time.
The arbiter removes the grant for the current master and assign it to the requested higher priority master.

Synopsys recommends that all AHB masters should be capable of rebuilding an early-terminated burst, regardless of the reason for which it occurred.

For more information about programming the early burst termination registers, see "Early Burst Termination Registers" on page 108.

2.1.1.9 Full Incrementing Bursts

Optional Feature. When a burst of unspecified length is issued from a master, you can control the updating of the internal arbiter. By supporting full incrementing bursts, the arbiter does not "early terminate" a burst transfer of unspecified length. The entire burst is allowed to complete. By not supporting full incrementing bursts, which is the default mode of operation, then when a master issues a burst transfer of unspecified length, the arbiter is free to update the grants to the highest priority master. This, in effect, early terminates the transfer.

2.1.1.10 Weighted-Token Arbitration

Optional Feature. In weighted-token arbitration, each master's clock token value is equivalent to the following:

[(maximum number of cycles) – two cycles]

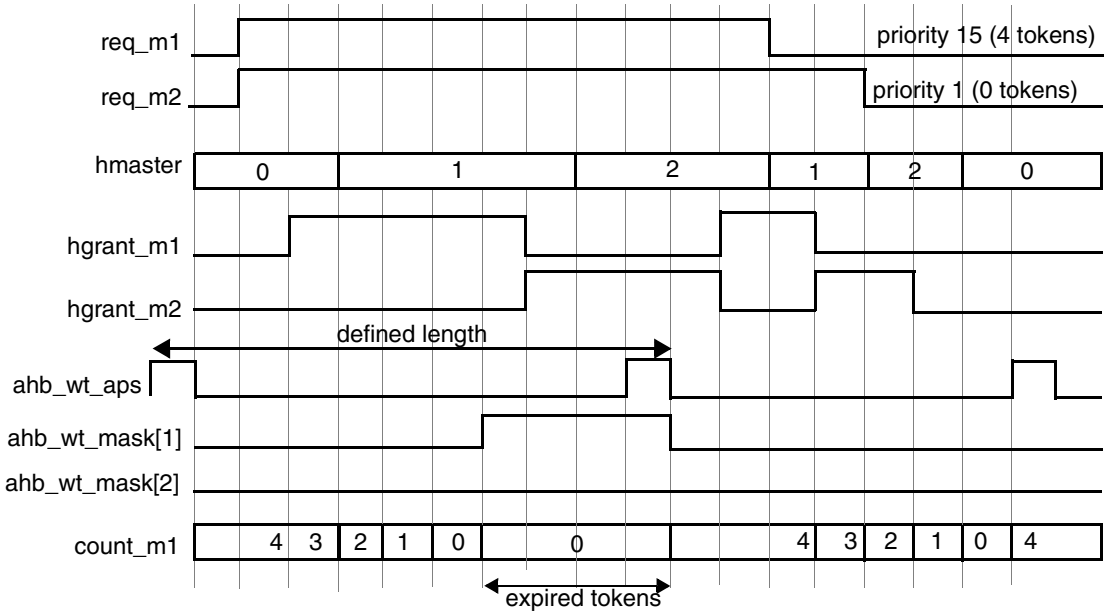
If the maximum number of cycles required is 100, then you should enter 98. The counter has a configured value of 98 to zero (99 cycles), after which the grant is removed that is, after the completion of the current (non-locked or locked) transfer, the bus is handed over to other competing masters.

The minimum arbitration period is the sum of all the tokens, taking into account the extra two cycles for each master. If any master is masked because it ran out of tokens each time a new arbitration period starts, it is unmasked. Granting operates on the upper tier arbitration scheme until all tokens are used. This feature requires a limitation where masters must be configured to different priorities. After configuration, you can program equal priorities, which functionally operates the same. You need to include an arbiter slave interface in the configuration. The initial token values can be hardcoded or left programmable. When the

values are programmable, you must ensure that the total arbitration period is long enough to count all the tokens, as there is no hardware check; otherwise, lower priority masters will be locked out.

Figure 2-5 illustrates two competing masters in a weighted-token arbitration scheme, which is set for a defined length of time as indicated by the ahb_wt_aps signal (for more information, see "Weighted-Token Arbitration Registers" on page 109).

Figure 2-5 Masters Competing for Bus Ownership with Weighted Tokens



Master 1 has a priority level of 15 (the highest) and has four tokens, while master 2 has a priority level of 1 (the lowest) and is allocated zero tokens. Having zero tokens is the same as having an infinite number of tokens, which means that master 2 always operates at the upper tier and never runs out of tokens.

When master 1 is granted ownership (hgrant_m1), it owns the bus with hmaster = 1. At this time, the master 1 token counter begins (count_m1). When master 1 is out of tokens, master 2 is granted ownership (hgrant_m2) until the arbitration period ends because it is still requesting ownership of the bus. Then the next arbitration period starts, at which point the master 1 token counter is reset and master 1 is granted ownership again because it has the highest priority and has tokens to use.

2.1.2 Optional Internal Decoder

The DW_ahb internal decoder generates the peripheral select lines for the slaves on the AHB by decoding the system address bus. During configuration, you can choose to include the internal decoder or a decoder external to the DW_ahb (for more information, see "Optional External Decoder" on page 33). Each system slave can be specified with a starting and ending address that must be aligned to 1 KB boundaries. Different slaves may have multiple start and end addresses. The Decoder is responsible for the following features:

- "Remap Operation"
- "Multiple Address Regions" on page 32
- "Slave Aliasing" on page 32
- "Slave Visibility" on page 33

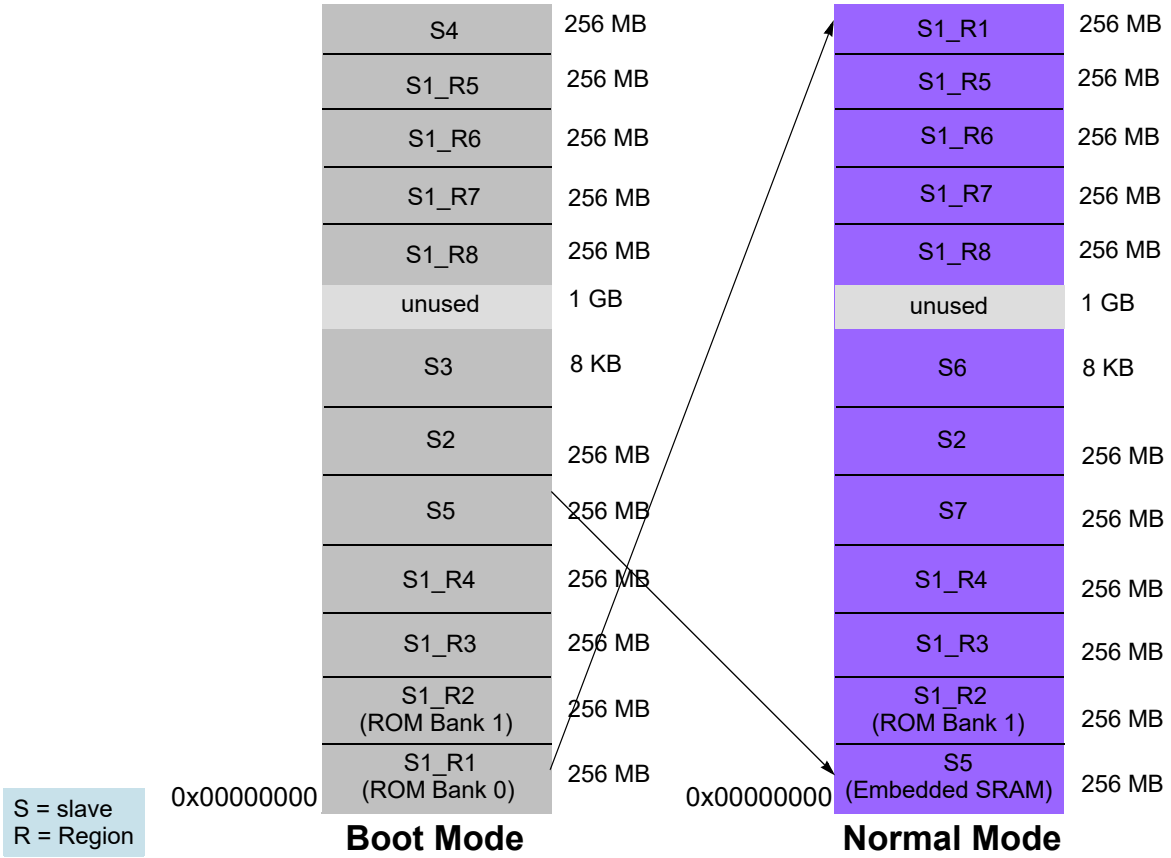
- "Default Slave" on page 33

2.1.2.1 Remap Operation

Processors may boot from one memory and then run from another. Often this means that address 0x00000000 needs to be remapped from one memory to another. The DW_ahb has two modes of operation to allow for this type of scenario: Boot Mode and Normal Mode. Each mode has its own memory map, which can be identical to or different from the other, depending on your configuration. This functionality is referred to as the DesignWare Remap Feature which, when enabled, allows you to configure the slave for both modes.

If this feature is not enabled, configuration options are available for only Normal Mode. Figure 2-6 illustrates memory maps for both Boot and Normal modes.

Figure 2-6 Example DW_ahb Memory Map – Boot Mode and Normal Mode



In the Boot Mode example, a ROM (slave 1) occupies the base address 0x00000000, whereas an embedded RAM (slave 5) is remapped to occupy the address location of 0x00000000 in Normal Mode.

Note When there is no arbiter slave interface, there is no slave 0.

Selection of the different memory maps is under the control of the input signal remap_n. In Boot Mode, the remap_n signal is logic zero, whereas it is logic one in Normal Mode. If you do not enable the Remap feature, remap_n is not included as a top-level signal; instead, it is internally tied off to 1 (the default setting of Normal Mode).

2.1.2.2 Multiple Address Regions

A memory controller with a single select line has a region for internal registers and a region for external memory. From the memory map's perspective, the internal registers could be at a different location than the external memory. These can be treated as separate regions without having to assign the entire memory space between the two regions to the memory controller.

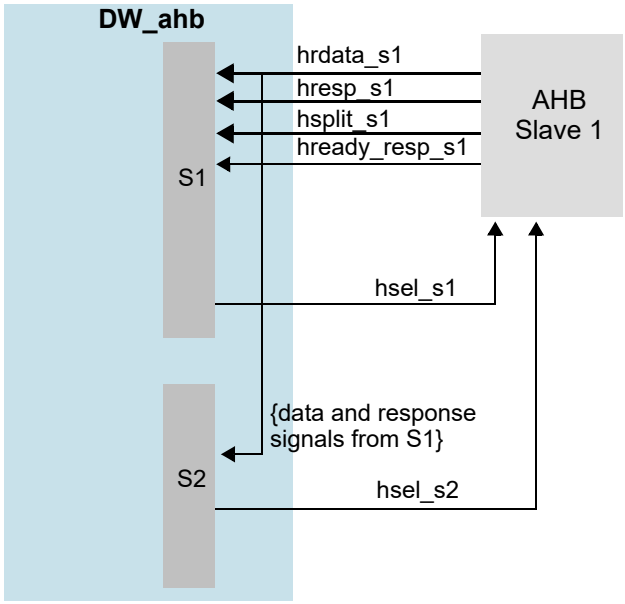
A slave does not have to occupy a continuous area of the memory map. DW_ahb is designed so that slave 1 can have up to eight regions (refer to Figure 2-6) in both Boot and Normal modes. It does not need the same number of regions in both modes. This allows a memory controller, for instance, to have banks at non-contiguous memory locations. For instance, in Figure 2-6 slave 1 is split into eight regions (S1_R1 through S1_R8) in both Boot and Normal modes. Other slaves can be assigned up to two regions of addresses in both modes.

2.1.2.3 Slave Aliasing

The DW_ahb also includes an alias feature, shown in Figure 2-7, that allows the data from one slave to be returned as data for another slave. This feature is designed for slaves with multiple hsel signals. For instance, a memory controller may have two hsel signals—one for internal registers and one for external registers—but may have only one set of data, response, split, and ready signals. The alias feature allows the same hrdata, hresp, hsplit, and hready_resp *lines of the current slave to be those of its aliased slave*.

Internally, each slave is treated individually as illustrated in Figure 2-7. When a second select line is used, then there is no need for the data and response signals to be connected at the top-level because they come from the original slave.

Figure 2-7 DW_ahb with Aliased Slave



2.1.2.4 Slave Visibility

A given slave can be visible (enabled) in either Boot Mode, Normal Mode, or both modes, so the number of slaves visible in the system may vary depending on the operating mode. For instance, in Figure 2-6, slave 2 has been configured to be visible in both modes, slaves 3 and 4 are visible in only Boot Mode, and slaves 6 and 7 are visible in only Normal Mode. Additionally, a peripheral does not have to occupy the same address space in both modes (for example, slave 1, region 1 occupies a different location in Normal Mode) than it does in Boot Mode.

2.1.2.5 Default Slave

When a master attempts to access an unassigned address, the DW_ahb returns an error response. The agent in the DW_ahb that provides this response is referred to as the *default slave*. The select line for the default slave is internal to the DW_ahb and is not a top-level I/O signal, unless the decoder is external.

2.1.3 Optional External Decoder

During configuration of the DW_ahb, you can choose to have an external decoder. By having the decoder external to the DW_ahb, you can connect any decoder with any number of remap options. When this option is chosen, the internal decoder is not included. There are inputs for the peripheral selects for controlling the return data from slaves.

2.1.4 Multiplexer

All address and control signals from each master are multiplexed, depending on which master owns the system address and control bus. The write data from each master is multiplexed, depending on which master owns the system data bus. All data from each slave is multiplexed, depending on which slave is addressed in the previous cycle.

2.1.5 Non-Standard Master ID Sideband Signal

The Non-Standard Master ID Sideband signal is available only in the AHB-Lite mode. The sideband signals (mid_in and mid_out) are available when the MID_WIDTH parameter is non zero (in AHB-Lite mode). These signals enable the propagation of the ID through DW_ahb. The timing of these signals is similar to other AHB control signals, such as hsize and hprot.

2.2 Timing Diagrams

For timing, refer to the following diagrams:

- Bursts with specified length (Figure 2-2)
- Bursts with unspecified length and equal priority levels (Figure 2-3)
- Bursts with unspecified length and different priority levels (Figure 2-4)

The following diagrams are from the *AMBA Specification (Rev. 2.0)*:

- Simple transfer (Figure 2-8)
- Transfer with wait states (Figure 2-9)
- Multiple transfers (Figure 2-10)
- Granting access with no wait states (Figure 2-11)

- Granting access with wait states (Figure 2-12)
- Data bus ownership (Figure 2-13)
- Hand-over after burst (Figure 2-14)

Figure 2-8 Simple Transfer

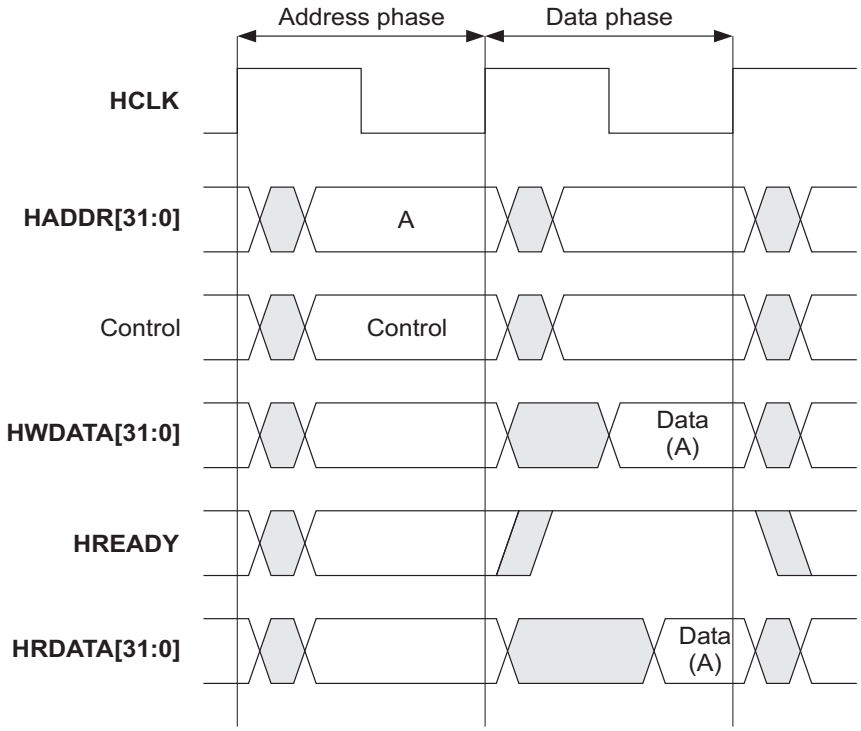


Figure 2-9 Transfer with Wait States

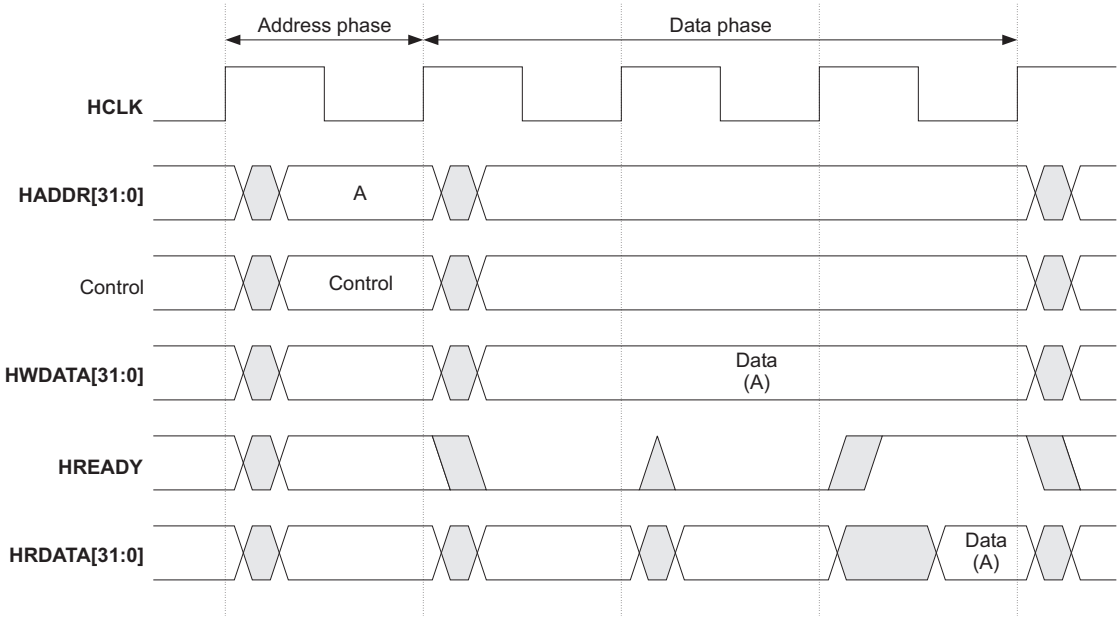


Figure 2-10 Multiple Transfers

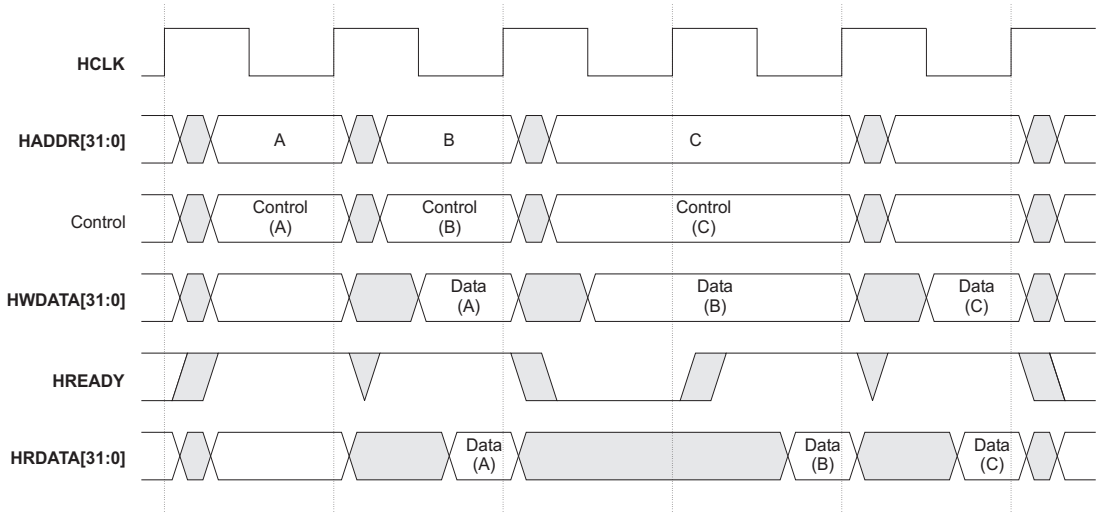


Figure 2-11 Granting Access with No Wait States

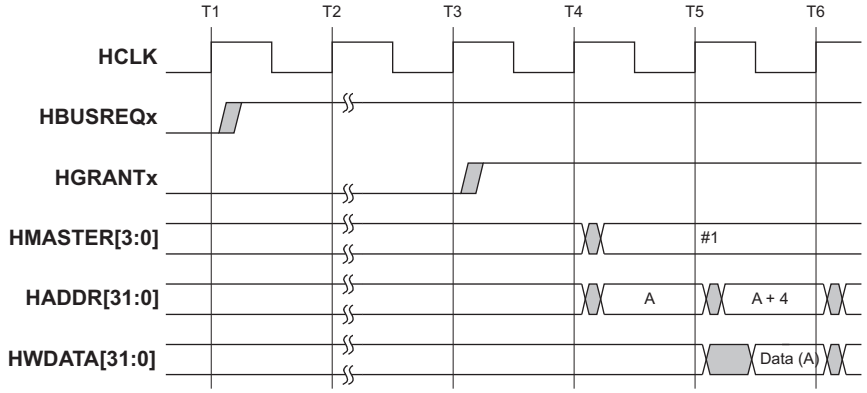


Figure 2-12 Granting Access with Wait States

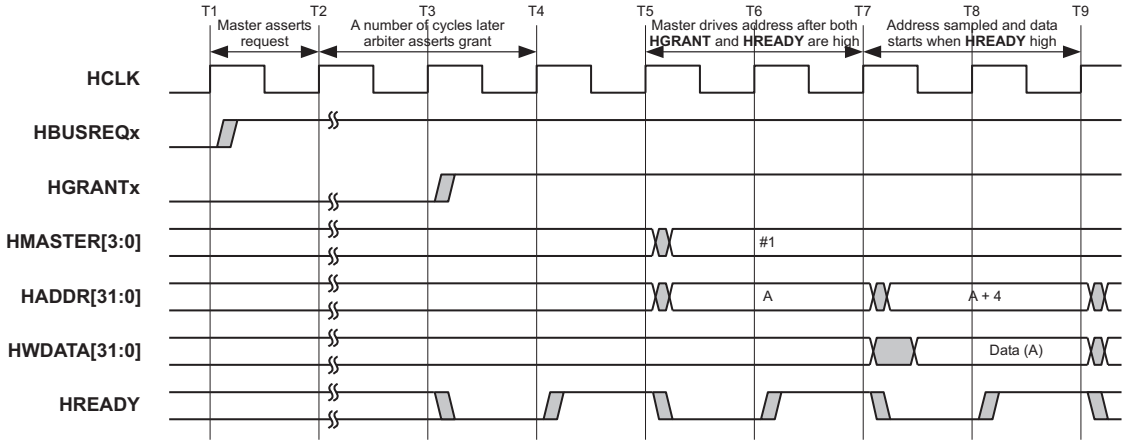


Figure 2-13 Data Bus Ownership

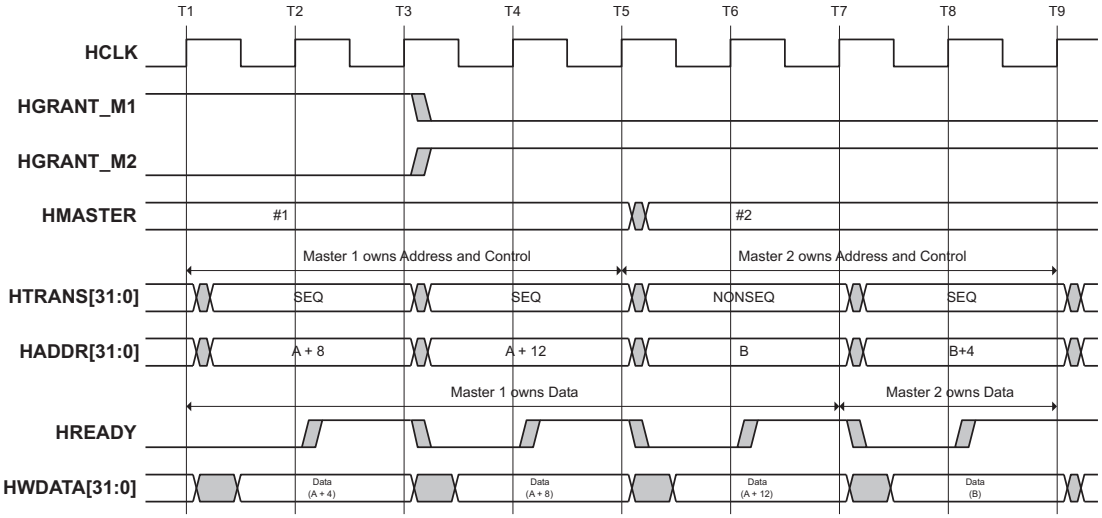
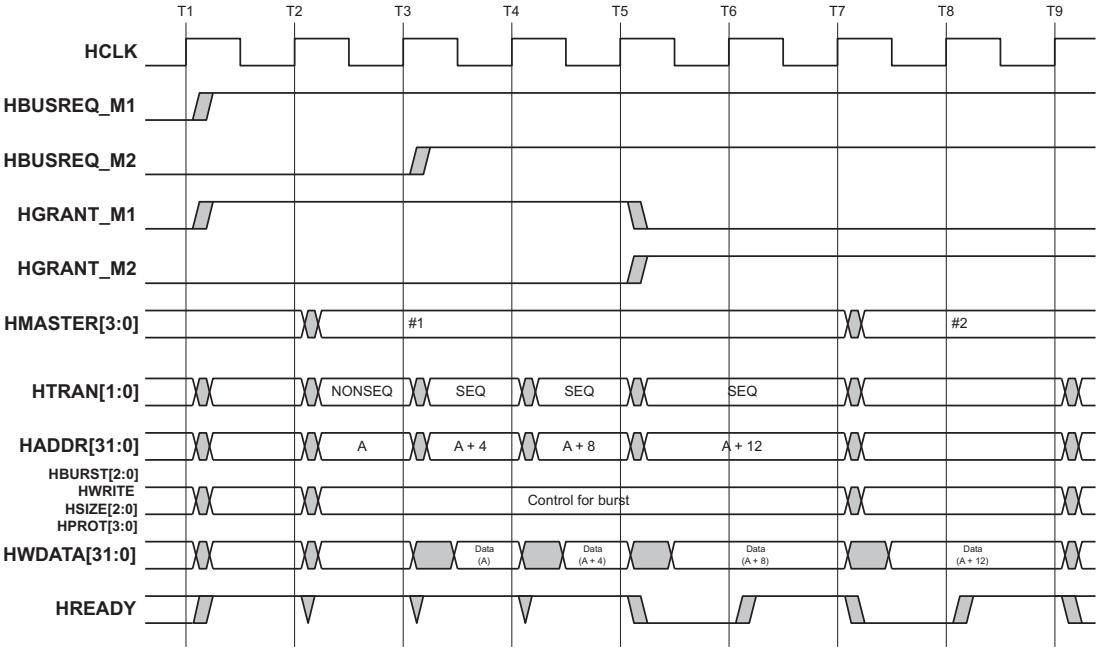


Figure 2-14 Handover after Burst



2.3 AMBA 5 AHB Features

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

This section describes the following features supported by DW_ahb to comply with AMBA 5 AHB protocol specification.

- "Extended Memory Types" on page 37
- "Secure Transfers" on page 37
- "Exclusive Transfers" on page 38
- "Multiple Slave Select" on page 38
- "User Signals" on page 38

These features are enabled by setting the configuration parameter AHB_INTERFACE_TYPE to 1.

Note In AHB5 configuration (AHB_INTERFACE_TYPE=1) Multiple slave select feature is present to support slaves with multiple select signals. Hence, Slave Aliasing feature is not supported in AHB5 configuration.

2.3.1 Extended Memory Types

DW_ahb supports Extended Memory Types feature of the AMBA 5 AHB protocol when the configuration parameter AHB_HAS_EXTENDED_MEMTYPE is set to 1. To support this feature, DW_ahb includes addidtional signals hprot_m1[6:4] and hprot[6:4] to its interface. The extended memory type information is propagated across the DW_ahb unmodified. For more information on these signals refer to the "Signal Descriptions" chapter.

2.3.2 Secure Transfers

DW_ahb supports secure transfers feature of AMBA 5 AHB protocol by adding hnonsec_m1 and hnonsec signals to its master and slave interface respectively. This feature is enabled by setting the configuration parameter AHB_HAS_SECURE_XFER to 1.

2.3.2.1 Trustzone Support

This is an optional feature that can be enabled by setting the Trustzone Support (AHB_HAS_TZ_SUPPORT) configuration parameter to 1. Each slave port has an input pin (tz_secure_s(j)) indicating whether the attached slave is a secure slave. If a non-secure read or write command access is addressed to a secure slave, as indicated by the hnonsec_m1 signal, then this command is routed to the default slave. The default slave generates an ERROR response for such transactions. The default slave gives ERROR response for SEQUENTIAL and NON-SEQUENTIAL transaction types and OKAY response for IDLE and BUSY transaction types.

When the configuration parameter AHB_HAS_TZ_SUPPORT is set to False, the input port tz_secure_s(j) does not exist and Trustzone feature is not supported in DW_ahb. Hence, DW_ahb does not take any action on secure or non-secure transfer. It is slave's responsibility to act and give proper response for secure or non-secure transfer.

Note The input signal tz_secure_s(j) must be stable once the corresponding slave is selected. It can change only when the slave is not selected.

Trustzone feature is supported only if the decoder is internal to the DW_ahb that is, AHB_HAS_XDCDR = 0.

2.3.3 Exclusive Transfers

DW_ahb supports Exclusive transfers feature of AMBA 5 AHB protocol when the configuration parameter AHB_HAS_EXC_XFER is set to 1. An exclusive transfer is indicated to the DW_ahb by hexcl_m1 master signal. DW_ahb intimates to the slave that the current transfer is part of an Exclusive access sequence through the hexcl signal. It is slave's responsibility to monitor the Exclusive access sequence and indicate the success or failure of an Exclusive transfer through the hexokay_s(j) signal. The signals associated with Exclusive transfer (hexcl_m1 and hexokay_s(j)) are transported across the DW_ahb unmodified.

DW_ahb adds master identifier signals, hmaster_m1, to its master interface when the parameter AHB5_HMASTER_WIDTH is configured to a non-zero value. hmaster_m1 signals are mapped to hmaster. When Exclusive transfers feature is enabled, these signals can be used to differentiate between multiple exclusive threads of masters.

2.3.4 Multiple Slave Select

DW_ahb supports multiple slave select signals on a single slave interface. This feature is enabled by setting the configuration parameter AHB_MULTI_SLV_SEL to 1. When the parameter AHB_MULTI_SLV_SEL is set to 1, width of the slave select signal, hsel_s(j), is equal to the number of address regions in jth slave (where, j <= NUM_IAHB_SLAVES). This feature supports slaves with multiple hsel signals. With multiple hsel signals a slave interface is capable of providing multiple logical interfaces, each addressing to a different location in the system address map.

2.3.5 User Signals

DW_ahb supports user signals on each channel. The configuration parameters AHB_A_UBW, AHB_W_UBW and AHB_R_UBW determines the existence and width of the user signals on address channel, write data channel and read data channel respectively. The User signals associated with a transfer are transported unmodified across the DW_ahb.

For timing of user signals, refer to the following timing diagrams:

Figure 2-15 shows the simple transfer with user signals.

Figure 2-16 shows the extended transfer with user signals.

Figure 2-17 shows the multiple transfers with user signals.

Figure 2-15 Simple Transfer with User Signals

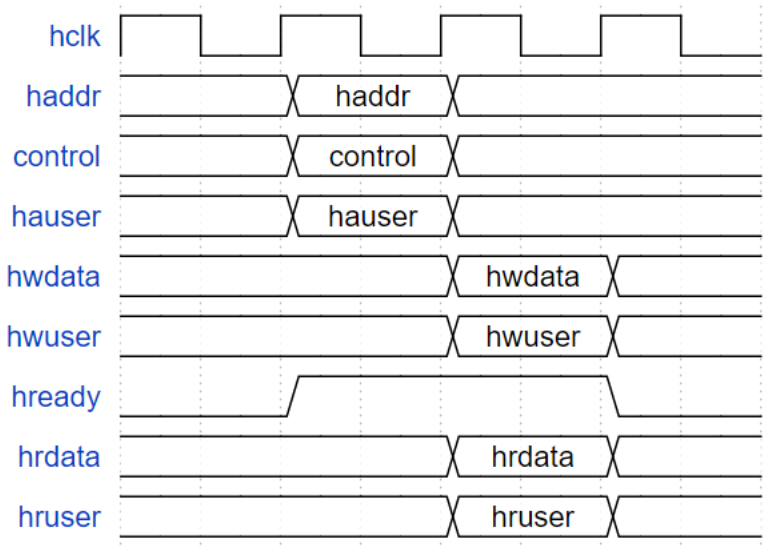


Figure 2-16 Extended Transfer with User Signals

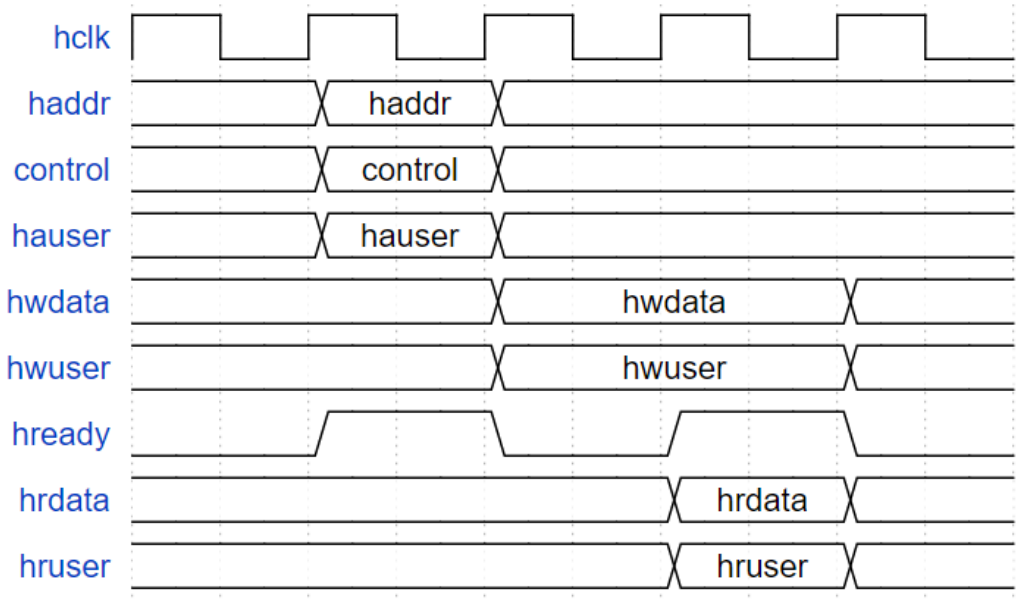
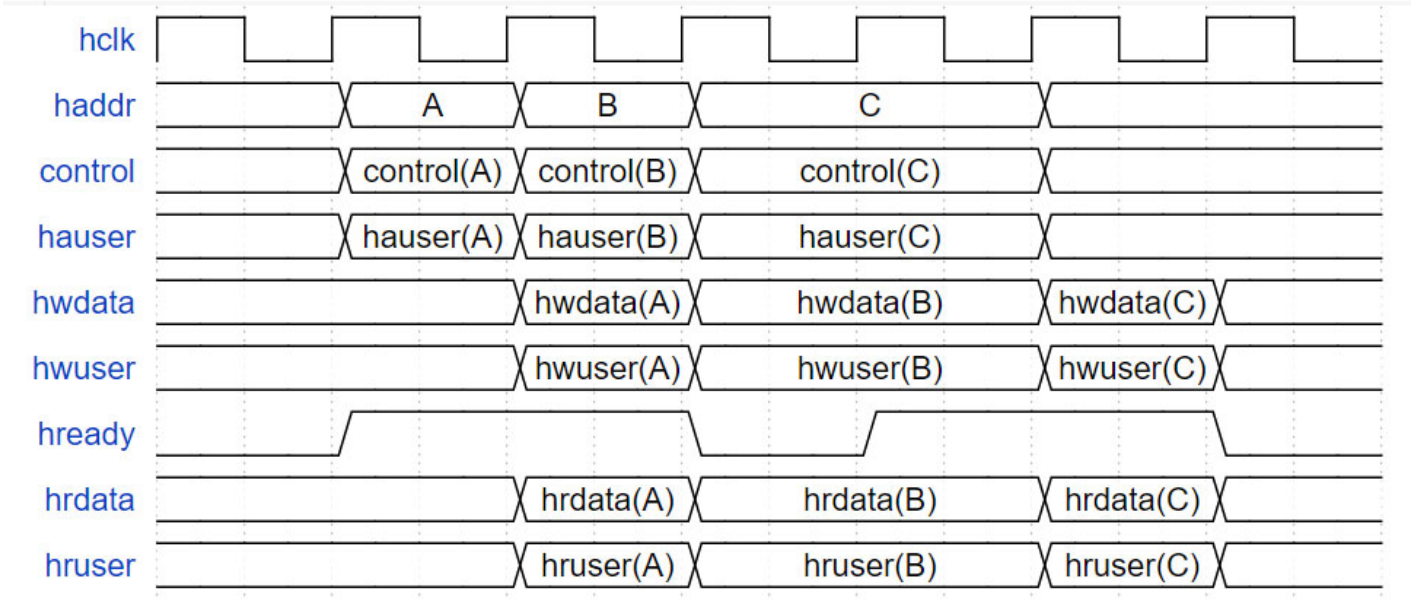


Figure 2-17 Multiple Transfers with User Signals



3 Parameter Descriptions

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

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

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

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
Top Level Parameters	
AMBA Lite	If set to True (1), the system is configured with only one master that never requests ownership of the bus, but is always granted the bus. No dummy master is required, since the slaves are not split capable. The master drives IDLE cycles when it does not want the bus. When this parameter is set to True: ■ Pause mode is not enabled ■ Default master number is changed to 1 ■ Number of masters is changed to 1 ■ HMASTERLOCK signal is not used for Locked Transfers. The HLOCK_M1 signal is used instead of the HMASTERLOCK signal. ■ Arbiter interface is removed ■ All slaves are made non split capable **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** AHB_LITE
Single-bit AHB Response in AHB-Lite Mode	When set to true, hresp is a single-bit signal, else it is of 2 bits. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_LITE==1 **Parameter Name:** AHB_SCALAR_HRESP
Select AHB Interface Type	This parameter selects the AHB interface type. **Values:** ■ AHB (0) ■ AHB5 (1) **Default Value:** AHB **Enabled:** AHB_LITE==1 and DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** AHB_INTERFACE_TYPE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Number of AHB Master Ports	The number of AHB Masters contained in the system. When the AMBA Lite parameter is configured, then this parameter is set to 1. You can have a system which has only one master but is not AMBA Lite. This allows you to program the default master to be the dummy master or the only existing master. The dummy master is the master that owns the bus when no other master wants it. There can be up to 15 masters connected to DW_ahb. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 **Default Value:** [<functionof> AHB_LITE] **Enabled:** AHB_LITE == 0 **Parameter Name:** NUM_AHB_MASTERS
AHB System Address Width	Selects the address width for the AHB address bus - 32-bit address or 64-bit address. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** HADDR_WIDTH
AHB Data Bus Width	Selects the width of the AHB data bus. The maximum 256-bit width is an arbitrary limitation enforced by coreConsultant. **Values:** 8, 16, 32, 64, 128, 256 **Default Value:** 32 **Enabled:** Always **Parameter Name:** AHB_DATA_WIDTH
External Endian Control	If True (1), the endian type of DW_ahb is controlled by the external pin, ahb_big_endian. ■ Big Endian: ahb_big_endian = 1 ■ Little Endian: ahb_big_endian = 0 If set to False (0), then the external signal ahb_big_endian is not included on the interface, and the endian type is set at configuration by the BIG_ENDIAN configuration parameter. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** AHB_XENDIAN

Table 3-1 Top Level Parameters (Continued)

Label	Description
System Endianness	By default, DW_ahb is configured as a Little-Endian system. The user can choose the endianness of the system if endian control is not external. If AHB_HAS_ARBIF is enabled, this parameter controls the endianness of accesses to the Arbiter Slave Interface. This parameter can be used to derive the endianness for the rest of the subsystem, so it is relevant (and changeable), even when AHB_HAS_ARBIF is disabled. **Values:** ■ Little-Endian (0) ■ Big-Endian (1) **Default Value:** Little-Endian **Enabled:** AHB_XENDIAN == 0 **Parameter Name:** BIG_ENDIAN
External Decoder	This parameter allows the decoder to be implemented within the parallel multi-layer matrix. If True (1), the decoder is external to DW_ahb. If False (0), decoder is internal to DW_ahb. For an internal decoder, the addresses need to be supplied by DW_ahb during configuration. Overlaps and inconsistencies are checked. An external decoder allows you to connect with any decoder with any number of remap options. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** AHB_HAS_XDCDR
Support AMBA Memory Remap	Allows the memory map to be swapped between normal and boot memories. When set to True, the system supports the DesignWare Memory Remap functionality. Remap functionality allows one set of addresses for boot, and another for normal operation. This setting must be done if two addressing modes are required. For more information, see "Remap Operation". **Values:** ■ false (0) ■ true (1) **Default Value:** 0 **Enabled:** AHB_HAS_XDCDR==0 **Parameter Name:** REMAP

Table 3-1 Top Level Parameters (Continued)

Label	Description
Support Arbiter Pause Mode	If set to True (1), the system supports the arbiter pause mode. This setting allows the granting of the bus to the dummy master when the system is entering the low-power mode. When AHB_LITE = 1, pause mode is disabled. **Values:** ■ false (0) ■ true (1) **Default Value:** 1 **Enabled:** AHB_LITE==0 **Parameter Name:** PAUSE
Support Delayed Pause Action	When the delayed pause action is supported, the pause signal, does not take effect until hready signal is high and htrans signal is IDLE. If delayed pause action is not set (False), DW_ahb enters the pause mode at the next hclk edge after the pause signal is set. By delaying the action on pause, any other transfers on the bus can be completed before the system is paused. **Values:** ■ false (0) ■ true (1) **Default Value:** [<functionof> AHB_LITE PAUSE] **Enabled:** (PAUSE==1) && (AHB_LITE==0) **Parameter Name:** AHB_DELAYED_PAUSE
Include Arbiter Interface	If you decide that there is no requirement for the programmable features within the AHB arbiter interface, this feature can be disabled (set to False). The peripheral slot (s0) is not available to other slaves. There is no arbiter interface when the system is an AHB Lite system. **Values:** ■ false (0) ■ true (1) **Default Value:** 1 **Enabled:** AHB_LITE==0 **Parameter Name:** AHB_HAS_ARBIF

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include Weighted Token Arbitration	Enables inclusion of a weighted token priority arbitration scheme. When the scheme is enabled, it is a third tier of arbitration. A master with clock tokens of a lower priority than a master with no clock tokens left to use will be granted the bus. When masters have used all clock tokens, the arbitration reverts to a two-tier arbitration. When a master has used all of its tokens, it will be granted the bus when masters with tokens are not requesting the bus. The internal arbiter slave must be included in order to use the weighted-token arbitration mode and to generate the debug outputs. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_HAS_ARBIF == 1 && AHB_LITE == 0 && NUM_AHB_MASTERS > 1 **Parameter Name:** AHB_WTEN
Include Weighted Token Outputs	Enables the inclusion of weighted token clock token counter outputs as top-level outputs that can help fine tune the number of tokens that you can assign to a master. These debug outputs show the number of tokens a master has left. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_HAS_ARBIF == 1 && AHB_LITE == 0 && AHB_WTEN == 1 **Parameter Name:** AHB_WTEN_DEBUG
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:** AHB_LITE == 1 && AHB_INTERFACE_TYPE == 0 **Parameter Name:** MID_WIDTH

Table 3-1 Top Level Parameters (Continued)

Label	Description
Support Full Incrementing Bursts	When a burst of unspecified length is issued from a master, updating the internal arbiter can be controlled. By supporting full incrementing bursts, the arbiter will not "early terminate" a burst transfer that is of an unspecified length. The entire burst is allowed to complete. By not supporting full incrementing bursts (the default mode of operation) makes the arbiter free to update the grants to the highest priority master when a master issues a burst transfer of an unspecified length. This, in effect, early terminates the currently granted transfer on the bus. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_LITE == 0 **Parameter Name:** AHB_FULL_INCR
Number of Slave Select Lines in the System	This number is the total number of slave select lines in the system. There may be slaves that are visible in one of the modes. There is still a slave select generated for the slave, so that in either of the addressing modes there can be 15 assigned slaves. If there is only one addressing mode, then this is the number of slaves in the system. **Values:** 1, ..., 31 **Default Value:** 4 **Enabled:** Always **Parameter Name:** NUM_IAHB_SLAVES
Number of AHB Slave Ports in Normal Mode	The number of slave select lines in the system in Normal mode, which is controlled by the slave's visibility. Slaves can be visible in both Normal and Boot modes. **Values:** 0, ..., 31 **Default Value:** NUM_IAHB_SLAVES **Enabled:** This parameter option is active only if you enable the Memory Remap Feature. **Parameter Name:** NUM_NAHB_SLAVES
Number of AHB Slave Ports in Boot Mode	The number of slave select lines contained in the system in Boot mode, which is controlled by the slave's visibility. Slaves can be visible in both Normal and Boot modes. **Values:** 0, ..., 31 **Default Value:** [<functionof> REMAP] **Enabled:** This parameter option is active only if you enable the Memory Remap Feature. **Parameter Name:** NUM_BAHB_SLAVES

Table 3-1 Top Level Parameters (Continued)

Label	Description
External Decoder Provides HSELS Back to DW_ahb for Routing to Slaves	If set to True (1), the decoder supplies hsel lines to DW_ahb for re-routing. If set to False (0), the decoder routs the hsel lines to the slaves. **Values:** ■ false (0) ■ true (1) **Default Value:** AHB_HAS_XDCDR **Enabled:** This parameter option is active only if you enable the External Decoder feature. **Parameter Name:** XDCDR_SUPPLIES_HSELS2AHB

3.2 AHB Source Code Configuration Parameters

Table 3-2 AHB Source Code Configuration Parameters

Label	Description
AHB Source Code Configuration	
Use DesignWare Foundation Synthesis Library	The component code utilizes DesignWare Foundation parts for optimal Synthesis QoR. Customers with only a DesignWare license MUST use Foundation parts. Customers with only a Source license, cannot use Foundation parts. Customers with both Source and DesignWare licenses can use Foundation parts. **Values:** ■ false (0) ■ true (1) **Default Value:** True if DesignWare License is available; False if no DesignWare License is available **Enabled:** Parameter is enabled if customer has both Source and DesignWare licenses. **Parameter Name:** USE_FOUNDATION

3.3 AMBA 5 AHB Configuration Parameters

Table 3-3 AMBA 5 AHB Configuration Parameters

Label	Description
AMBA 5 AHB Configuration	
Include AHB5 Extended Memory Types Property	Select this parameter to include AHB5 Extended Memory Types Property in DW_ahb. When set to 1, DW_ahb adds additional hprot_m1[6:4] and hprot[6:4] signals to its interface. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_INTERFACE_TYPE==1 **Parameter Name:** AHB_HAS_EXTENDED_MEMTYPE
Include AHB5 Secure Transfers Property	Select this parameter to include AHB5 Secure Transfers Property in DW_ahb. When set to 1, DW_ahb adds additional hnonsec_m1 and hnonsec signals to its interface. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_INTERFACE_TYPE==1 **Parameter Name:** AHB_HAS_SECURE_XFER
Enable TrustZone Feature	When set to True, it enables the TrustZone support. An input port (tz_secure_s(j)) exists for each external AHB slave, indicating whether the slave is secure. When set to False, tz_secure_s(j) signal does not exist and TrustZone functionality is not supported - DW_ahb does not act on secure or non-secure transfers. For more information about this feature, refer to "TrustZone Support". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_HAS_SECURE_XFER==1 && AHB_HAS_XDCDR==0 **Parameter Name:** AHB_HAS_TZ_SUPPORT
Include Exclusive Transfers Property	Select this parameter to include AHB5 Exclusive Transfers property in DW_ahb. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_INTERFACE_TYPE==1 **Parameter Name:** AHB_HAS_EXC_XFER

Table 3-3 AMBA 5 AHB Configuration Parameters (Continued)

Label	Description
Width of the AHB5 Master Identifier Signal	This parameter determines the width of the AHB5 master identifier signals hmaster_m1. When set to 0, these ports are removed from DW_ahb. **Values:** 0, ..., 12 **Default Value:** 0 **Enabled:** AHB_INTERFACE_TYPE==1 **Parameter Name:** AHB5_HMASTER_WIDTH
Include Multiple Slave Select Feature	When set to True, a single slave interface supports multiple slave select, hsel_s(j), signals. For more information, please refer to the "Signal Descriptions" chapter. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_INTERFACE_TYPE==1 **Parameter Name:** AHB_MULTI_SLV_SEL

3.4 User Signal Configuration Parameters

Table 3-4 User Signal Configuration Parameters

Label	Description
User Signal Configuration	
Width of Address Channel User Bus	Specifies DW_ahb address channel user bus width. When set to 0, the address channel user signals are removed from the interface. **Values:** 0, ..., 256 **Default Value:** 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** AHB_A_UBW
Width of Write Data Channel User Bus	Specifies DW_ahb write data channel user bus width. When set to 0, the write data channel user signals are removed from the interface. **Values:** 0, ..., 256 **Default Value:** 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** AHB_W_UBW
Width of Read Data Channel User Bus	Specifies DW_ahb read data channel user bus width. When set to 0, the read data channel user signals are removed from the interface. **Values:** 0, ..., 256 **Default Value:** 0 **Enabled:** DWC-AMBA-AHB5-Fabric-Source Add on Source license exists. **Parameter Name:** AHB_R_UBW

3.5 Slave Memory Region Definition Parameters

Table 3-5 Slave Memory Region Definition Parameters

Label	Description
Slave 1	
Slave that returns data and response	The slave number that supplies the data and response. The value must be less than, or equal to the value of NUM_IAHB_SLAVES. The value of this parameter cannot equal 1, meaning you cannot alias this slave to itself. **Values:** 1, ..., 31 **Default Value:** 1 **Enabled:** This parameter option is available only if the HSEL_ONLY_S1 = 1 **Parameter Name:** ALIAS_S1
Alias this Slave to Another System Slave	Generates only hsel for this slave; requires an aliases for data and response from another slave. For more information, see "Slave Aliasing". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_INTERFACE_TYPE==0 **Parameter Name:** HSEL_ONLY_S1
Number of Memory Regions in Boot Mode	The number of memory regions in Boot Mode for slave 1. For more information, see "Multiple Address Regions". **Values:** ■ 1 Region (0) ■ 2 Regions (1) ■ 3 Regions (2) ■ 4 Regions (3) ■ 5 Regions (4) ■ 6 Regions (5) ■ 7 Regions (6) ■ 8 Regions (7) **Default Value:** 1 Region **Enabled:** This parameter option is available only for slave 1 and if the Slave Visibility Mode is set to "Boot" or "Normal and Boot". Additionally, this option applies only if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** MR_B1

Table 3-5 Slave Memory Region Definition Parameters (Continued)

Label	Description
Number of Memory Regions in Normal Mode	The number of memory regions in Normal Mode for slave 1. **Values:** ■ 1 Region (0) ■ 2 Regions (1) ■ 3 Regions (2) ■ 4 Regions (3) ■ 5 Regions (4) ■ 6 Regions (5) ■ 7 Regions (6) ■ 8 Regions (7) **Default Value:** 1 Region **Enabled:** This parameter option is available only for slave 1 and if the Slave Visibility Mode is set to "Normal" or "Normal and Boot". Additionally, this option applies only if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** MR_N1
Split Capable	If the slave has an hsplit bus, then set this parameter to True (1). When a slave is aliased, it takes its split capability from the slave it is aliased to. Therefore, this option is dimmed if the HSEL_ONLY_S1 parameter is set to True. Split Capable parameter is disabled if AHB lite mode is enabled (AHB_LITE == 1). **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** When a slave is aliased, it takes its split capability from the aliased slave number. Therefore, this option will be dimmed if the HSEL_ONLY_S1 parameter is set to 1. This parameter is disabled if AHB Lite mode is enabled; (AHB_LITE = 1). **Parameter Name:** SPLIT_CAPABLE_1
Slave Visibility Mode	A given slave is visible in either Boot Mode, Normal Mode, or both the modes, so the number of slaves visible in the system may vary depending on the operating mode. For more information, refer to "Slave Visibility". **Values:** ■ Normal (1) ■ Boot (2) ■ Normal & Boot (3) **Default Value:** Normal **Enabled:** This parameter option is active only if you enable the Memory Remap Feature and have configured to use an internal decoder (AHB_HAS_XDCDR = 0) in the top-level parameter options. **Parameter Name:** VISIBLE_1

Table 3-5 Slave Memory Region Definition Parameters (Continued)

Label	Description
Slave j	
Slave that returns data and response (for j = 2; j <= 31)	The value must be less than or equal to the value of NUM_IAHB_SLAVES. The value of this parameter cannot equal j, meaning you cannot alias this slave to itself. This parameter option is available only if the HSEL_ONLY_S(j) = 1. **Values:** 1, ..., 31 **Default Value:** 1 **Enabled:** HSEL_ONLY_S(j)==1 && NUM_IAHB_SLAVES>j **Parameter Name:** ALIAS_S(j)
Alias this slave to another system slave? (for j = 2; j <= 31)	When this is active, only the slave select port is generated on the interface. The corresponding data, and response ports are taken from another slave, which is specified by the "Number of slave which returns data and response". For more information, see "Multiple Address Regions". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** NUM_IAHB_SLAVES>j **Parameter Name:** HSEL_ONLY_S(j)
Support Multiple Memory Regions in Boot Mode? (for j = 2; j <= 31)	Number of regions in Boot Mode for slave(j). For more information, see "Multiple Address Regions". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** This parameter option is available only if the Slave Visibility Mode is set to "Boot" or "Normal & Boot". This option applies only if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** MR_B(j)
Support Multiple Memory Regions in Normal Mode? (for j = 2; j <= 31)	Number of regions in Normal Mode for slave(j). For more information, see "Multiple Address Regions". **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** This parameter option is available only if the Slave Visibility Mode is set to "Normal" or "Normal & Boot". Additionally, this option applies only if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** MR_N(j)

Table 3-5 Slave Memory Region Definition Parameters (Continued)

Label	Description
Split Capable ? (for j = 2; j <= 31)	If the slave has an hsplit bus, then set this parameter to True(1). **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** When a slave is aliased, it takes its split capability from the aliased slave number. Therefore, this option will be dimmed if the HSEL_ONLY_S(j) is 1. This parameter is disabled if AHB Lite mode is enabled; (AHB_LITE = 1). **Parameter Name:** SPLIT_CAPABLE_(j)
Slave Visibility Mode (for j = 2; j <= 31)	A given slave is visible in either Boot Mode, Normal Mode, or both modes, so the number of slaves visible in the system may vary depending on the operating mode. For more information, refer to "Slave Visibility". **Values:** ■ Normal (1) ■ Boot (2) ■ Normal & Boot (3) **Default Value:** Normal **Enabled:** This parameter option is active only if you enable the Memory Remap Feature and have **Parameter Name:** VISIBLE_(j)

3.6 Normal Mode Address Map Parameters

Table 3-6 Normal Mode Address Map Parameters

Label	Description
Slave 1	
Normal Mode Region x Start Address (for x = 1; x <= 8)	Region 1 through 8, normal addressing mode, start address for slave 1. Specified if the peripherals address region is spread over multiple regions. The regions can be separate blocks or contiguous blocks. For region 1 to 8 default value is: Region 1: 0x2000000 Region 2: 0x3000000 Region 3: 0x4000000 Region 4: 0x5000000 Region 5: 0x6000000 Region 6: 0x7000000 Region 7: 0x8000000 Region 8: 0x9000000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xfffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Normal Mode" is set to x and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R(x)_N_SA_1
Normal Mode Region x End Address (for x = 1; x <= 8)	Region 1 through 8, normal addressing mode, end address for slave 1. Specified if the peripherals address region is spread over multiple regions. For region 1 to 8 default value is: Region 1: 0x200ffff Region 2: 0x300ffff Region 3: 0x400ffff Region 4: 0x500ffff Region 5: 0x600ffff Region 6: 0x700ffff Region 7: 0x800ffff Region 8: 0x900ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** 0x200ffff **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Normal Mode" is set to x and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R(x)_N_EA_1

Table 3-6 Normal Mode Address Map Parameters (Continued)

Label	Description
Slave j	
Normal Mode Region 1 Start Address (for j = 2; j <= 31)	Region 1, normal addressing mode, start address for Slave 2 through 31. For region 2 to 31 default value is: Slave 2: 0xa000000 Slave 3: 0xc000000 Slave 4: 0xe000000 Slave 5: 0x10000000 Slave 6: 0x12000000 Slave 7: 0x14000000 Slave 8: 0x16000000 Slave 9: 0x18000000 Slave 10: 0x1a000000 Slave 11: 0x1c000000 Slave 12: 0x1e000000 Slave 13: 0x20000000 Slave 14: 0x22000000 Slave 15: 0x24000000 Slave 16: 0x4b000000 Slave 17: 0x4d000000 Slave 18: 0x4f000000 Slave 19: 0x51000000 Slave 20: 0x53000000 Slave 21: 0x55000000 Slave 22: 0x57000000 Slave 23: 0x59000000 Slave 24: 0x5b000000 Slave 25: 0x5d000000 Slave 26: 0x5f000000 Slave 27: 0x61000000 Slave 28: 0x63000000 Slave 29: 0x65000000 Slave 30: 0x67000000 Slave 31: 0x69000000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xfffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is available only if the Slave Visibility Mode is set to "Normal" or "Normal & Boot" and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_N_SA_(j)

Table 3-6 Normal Mode Address Map Parameters (Continued)

Label	Description
Normal Mode Region 1 End Address (for j = 2; j <= 31)	Region 1, normal addressing mode, end address for Slave 2 through 31. For region 2 to 31 default value is: Slave 2: 0xa00ffff Slave 3: 0xc00ffff Slave 4: 0xe00ffff Slave 5: 0x1000ffff Slave 6: 0x1200ffff Slave 7: 0x1400ffff Slave 8: 0x1600ffff Slave 9: 0x1800ffff Slave 10: 0x1a00ffff Slave 11: 0x1c00ffff Slave 12: 0x1e00ffff Slave 13: 0x2000ffff Slave 14: 0x2200ffff Slave 15: 0x2400ffff Slave 16: 0x4b00ffff Slave 17: 0x4d00ffff Slave 18: 0x4f00ffff Slave 19: 0x5100ffff Slave 20: 0x5300ffff Slave 21: 0x5500ffff Slave 22: 0x5700ffff Slave 23: 0x5900ffff Slave 24: 0x5b00ffff Slave 25: 0x5d00ffff Slave 26: 0x5f00ffff Slave 27: 0x6100ffff Slave 28: 0x6300ffff Slave 29: 0x6500ffff Slave 30: 0x6700ffff Slave 31: 0x6900ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** See Description **Enabled:** This parameter option is available only if the Slave Visibility Mode is set to "Normal" or "Normal & Boot" and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_N_EA_(j)

Table 3-6 Normal Mode Address Map Parameters (Continued)

Label	Description
Normal Mode Region 2 Start Address (for j = 2; j <= 31)	Region 2, normal addressing mode, start address for Slave 2 through 31. if the peripherals address region is spread over multiple regions. For region 2 to 31 default value is: Slave 2: 0xb000000 Slave 3: 0xd000000 Slave 4: 0xf000000 Slave 5: 0x11000000 Slave 6: 0x13000000 Slave 7: 0x15000000 Slave 8: 0x17000000 Slave 9: 0x19000000 Slave 10: 0x1b000000 Slave 11: 0x1d000000 Slave 12: 0x1f000000 Slave 13: 0x21000000 Slave 14: 0x23000000 Slave 15: 0x25000000 Slave 16: 0x4c000000 Slave 17: 0x4e000000 Slave 18: 0x50000000 Slave 19: 0x52000000 Slave 20: 0x54000000 Slave 21: 0x56000000 Slave 22: 0x58000000 Slave 23: 0x5a000000 Slave 24: 0x5c000000 Slave 25: 0x5e000000 Slave 26: 0x60000000 Slave 27: 0x62000000 Slave 28: 0x64000000 Slave 29: 0x66000000 Slave 30: 0x68000000 Slave 31: 0x6a000000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xfffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Normal Mode" is set to 1 and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R2_N_SA_(j)

Table 3-6 Normal Mode Address Map Parameters (Continued)

Label	Description
Normal Mode Region 2 End Address (for j = 2; j <= 31)	Region 2, normal addressing mode, end address for Slave 2 through 31. Specified if the peripheral address region is spread over multiple regions. For region 2 to 31 default value is: Slave 2: 0xb00ffff Slave 3: 0xd00ffff Slave 4: 0xf00ffff Slave 5: 0x1100ffff Slave 6: 0x1300ffff Slave 7: 0x1500ffff Slave 8: 0x1700ffff Slave 9: 0x1900ffff Slave 10: 0x1b00ffff Slave 11: 0x1d00ffff Slave 12: 0x1f00ffff Slave 13: 0x2100ffff Slave 14: 0x2300ffff Slave 15: 0x2500ffff Slave 16: 0x4c00ffff Slave 17: 0x4e00ffff Slave 18: 0x5000ffff Slave 19: 0x5200ffff Slave 20: 0x5400ffff Slave 21: 0x5600ffff Slave 22: 0x5800ffff Slave 23: 0x5a00ffff Slave 24: 0x5c00ffff Slave 25: 0x5e00ffff Slave 26: 0x6000ffff Slave 27: 0x6200ffff Slave 28: 0x6400ffff Slave 29: 0x6600ffff Slave 30: 0x6800ffff Slave 31: 0x6a00ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Normal Mode" is set to 1 and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Parameter Name:** R2_N_EA_(j)

3.7 Boot Mode Address Map Parameters

Table 3-7 Boot Mode Address Map Parameters

Label	Description
Slave 1	
Boot Mode Region x Start Address (for x = 1; x <= 8)	Region 1 through 8, boot addressing mode, start address for slave 1. Specified if the peripherals address region is spread over multiple regions. For region 1 to 8 default value is: Region 1: 0x2700000 Region 2: 0x2800000 Region 3: 0x2900000 Region 4: 0x2a00000 Region 5: 0x2b00000 Region 6: 0x2c00000 Region 7: 0x2d00000 Region 8: 0x2e00000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Boot Mode" is set to x, if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Parameter Name:** R(x)_B_SA_1
Boot Mode Region x End Address (for x = 1; x <= 8)	Region 1 through 8, boot addressing mode, end address for slave 1. Specified if the peripherals address region is spread over multiple regions. For region 1 to 8 default value is: Region 1: 0x270ffff Region 2: 0x280ffff Region 3: 0x290ffff Region 4: 0x2a0ffff Region 5: 0x2b0ffff Region 6: 0x2c0ffff Region 7: 0x2d0ffff Region 8: 0x2e0ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xfffffff : 0xffffffffffffffff **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Boot Mode" is set to x, if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Parameter Name:** R(x)_B_EA_1

Table 3-7 Boot Mode Address Map Parameters (Continued)

Label	Description
Slave j	
Boot Mode Region 1 Start Address (for j = 2; j <= 31)	Region 1, boot addressing mode, start address for Slave 2 through 31. For slave 2 to 31 default value is: Slave 2: 0x2f000000 Slave 3: 0x31000000 Slave 4: 0x33000000 Slave 5: 0x35000000 Slave 6: 0x37000000 Slave 7: 0x39000000 Slave 8: 0x3b000000 Slave 9: 0x3d000000 Slave 10: 0x3f000000 Slave 11: 0x41000000 Slave 12: 0x43000000 Slave 13: 0x45000000 Slave 14: 0x47000000 Slave 15: 0x49000000 Slave 16: 0x6b000000 Slave 17: 0x6d000000 Slave 18: 0x6f000000 Slave 19: 0x71000000 Slave 20: 0x73000000 Slave 21: 0x75000000 Slave 22: 0x77000000 Slave 23: 0x79000000 Slave 24: 0x7b000000 Slave 25: 0x7d000000 Slave 26: 0x7f000000 Slave 27: 0x81000000 Slave 28: 0x83000000 Slave 29: 0x85000000 Slave 30: 0x87000000 Slave 31: 0x89000000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xfffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is applicable if you have an internal decoder (AHB_HAS_XDCDR = 0) and if REMAP = 1. **Parameter Name:** R1_B_SA_(j)

Table 3-7 Boot Mode Address Map Parameters (Continued)

Label	Description
Boot Mode Region 1 End Address (for j = 2; j <= 31)	Region 1 boot, addressing mode, end address for slave 2 through 31. For slave 2 to 31 default value is: Slave 2: 0x2f00ffff Slave 3: 0x3100ffff Slave 4: 0x3300ffff Slave 5: 0x3500ffff Slave 6: 0x3700ffff Slave 7: 0x3900ffff Slave 8: 0x3b00ffff Slave 9: 0x3d00ffff Slave 10: 0x3f00ffff Slave 11: 0x4100ffff Slave 12: 0x4300ffff Slave 13: 0x4500ffff Slave 14: 0x4700ffff Slave 15: 0x4900ffff Slave 16: 0x6b00ffff Slave 17: 0x6d00ffff Slave 18: 0x6f00ffff Slave 19: 0x7100ffff Slave 20: 0x7300ffff Slave 21: 0x7500ffff Slave 22: 0x7700ffff Slave 23: 0x7900ffff Slave 24: 0x7b00ffff Slave 25: 0x7d00ffff Slave 26: 0x7f00ffff Slave 27: 0x8100ffff Slave 28: 0x8300ffff Slave 29: 0x8500ffff Slave 30: 0x8700ffff Slave 31: 0x8900ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** See Description **Enabled:** This parameter option is applicable only if you have an internal decoder (AHB_HAS_XDCDR = 0) and if REMAP = 1. **Parameter Name:** R1_B_EA_(j)

Table 3-7 Boot Mode Address Map Parameters (Continued)

Label	Description
Boot Mode Region 2 Start Address (for j = 2; j <= 31)	Region 2, boot addressing mode, start address for Slave 2 through 31. For slave 2 to 31 default value is: Slave 2: 0x30000000 Slave 3: 0x32000000 Slave 4: 0x34000000 Slave 5: 0x36000000 Slave 6: 0x38000000 Slave 7: 0x3a000000 Slave 8: 0x3c000000 Slave 9: 0x3e000000 Slave 10: 0x40000000 Slave 11: 0x42000000 Slave 12: 0x44000000 Slave 13: 0x46000000 Slave 14: 0x48000000 Slave 15: 0x4a000000 Slave 17: 0x6e000000 Slave 18: 0x70000000 Slave 19: 0x72000000 Slave 20: 0x74000000 Slave 21: 0x76000000 Slave 22: 0x78000000 Slave 23: 0x7a000000 Slave 24: 0x7c000000 Slave 25: 0x7e000000 Slave 26: 0x80000000 Slave 27: 0x82000000 Slave 28: 0x84000000 Slave 29: 0x86000000 Slave 30: 0x88000000 Slave 31: 0x8a000000 **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xfffffc00 : 0xfffffffffffffc00 **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Boot Mode" is set to 1, if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Parameter Name:** R2_B_SA_(j)

Table 3-7 Boot Mode Address Map Parameters (Continued)

Label	Description
Boot Mode Region 2 End Address (for j = 2; j <= 31)	Region 2, boot addressing mode, end address for Slave 2 through 31. For slave 2 to 31 default value is: Slave 2: 0x3000ffff Slave 3: 0x3200ffff Slave 4: 0x3400ffff Slave 5: 0x3600ffff Slave 6: 0x3800ffff Slave 7: 0x3a00ffff Slave 8: 0x3c00ffff Slave 9: 0x3e00ffff Slave 10: 0x4000ffff Slave 11: 0x4200ffff Slave 12: 0x4400ffff Slave 13: 0x4600ffff Slave 14: 0x4800ffff Slave 15: 0x4a00ffff Slave 17: 0x6e00ffff Slave 18: 0x7000ffff Slave 19: 0x7200ffff Slave 20: 0x7400ffff Slave 21: 0x7600ffff Slave 22: 0x7800ffff Slave 23: 0x7a00ffff Slave 24: 0x7c00ffff Slave 25: 0x7e00ffff Slave 26: 0x8000ffff Slave 27: 0x8200ffff Slave 28: 0x8400ffff Slave 29: 0x8600ffff Slave 30: 0x8800ffff Slave 31: 0x8a00ffff **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** See Description **Enabled:** This parameter option is available only if the "Multiple Memory Regions in Boot Mode" is set to 1, if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Parameter Name:** R2_B_EA_(j)

3.8 Arbiter Priority Assignments Parameters

Table 3-8 Arbiter Priority Assignments Parameters

Label	Description
Arbiter Priority Assignments	
Master i Priority (for i = 1; i <= 15)	Arbitration priority associated with master i. Priority 1 (0x1) is the lowest and priority 15 (0xf) is the highest. It is not possible to configure a priority of zero (0x0), which disables the master. However, it is possible to program it, provided the priority values are not hardcoded. **Values:** 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf **Default Value:** 0xi **Enabled:** NUM_AHB_MASTERS>1 && AHB_LITE == 0 **Parameter Name:** PRIORITY_(i)

3.9 Slave Arbiter Configuration Parameters

Table 3-9 Slave Arbiter Configuration Parameters

Label	Description
Slave Arbiter Configuration	
AHB Arbiter Start Address (Normal Mode)	Normal Mode start address for AHB arbiter. **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xffffffc00 : 0xfffffffffffffc00 **Default Value:** 0x1000000 **Enabled:** The arbiter slave interface must be included in the design (AHB_HAS_ARBIF = 1), and the decoder must be configured as internal (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_N_SA_0
AHB Arbiter End Address (Normal Mode)	Normal Mode end address for AHB arbiter **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** 0x10003ff **Enabled:** The arbiter slave interface must be included in the design (AHB_HAS_ARBIF = 1), and the decoder must be configured as internal (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_N_EA_0
Boot Mode start Address for AHB Arbiter	Boot Mode start address for AHB arbiter **Values:** (HADDR_WIDTH==32) ? 0x00000000 : 0x0000000000000000, ..., (HADDR_WIDTH==32) ? 0xffffffc00 : 0xfffffffffffffc00 **Default Value:** 0x26000000 **Enabled:** This parameter option is active only if you enable the Memory Remap Feature (REMAP = 1) in the top-level parameter options, include the arbiter slave interface in the design, (AHB_HAS_ARBIF = 1), and configure the decoder as internal (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_B_SA_0
AHB Arbiter End Address (Boot Mode)	Boot Mode end address for AHB arbiter **Values:** (HADDR_WIDTH==32) ? 0x000003ff : 0x00000000000003ff, ..., (HADDR_WIDTH==32) ? 0xffffffff : 0xffffffffffffffff **Default Value:** 0x260003ff **Enabled:** This parameter option is active only if you enable the Memory Remap Feature (REMAP = 1) in the top-level parameter options, include the arbiter slave interface in the design, (AHB_HAS_ARBIF = 1), and configure the decoder as internal (AHB_HAS_XDCDR = 0). **Parameter Name:** R1_B_EA_0

Table 3-9 Slave Arbiter Configuration Parameters (Continued)

Label	Description
Use Hard-coded Arbiter Priorities	If this parameter is set to Ture, the arbiter priorities will be read only. If it is set to No, the arbiter priorities can be programmed during runtime. **Values:** ■ false (0) ■ true (1) **Default Value:** 0 **Enabled:** If there is no arbiter slave interface, this parameter is dimmed and hardcoded to true. **Parameter Name:** HC_PRIORITIES
Default Master Number	A default master is required according to the *AMBA Specification (Rev. 2.0)*. You can set this parameter to 0 if you want the dummy master to act as the default master. For more information, refer to "Default Master versus Dummy Master" **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf **Default Value:** 0 **Enabled:** The value must be less than or equal to the value of NUM_AHB_MASTERS. If weighted-token arbitration is enabled, then this value is hardcoded to 0. **Parameter Name:** DFLT_MSTR_NUM
Use Hard-coded Default Master	If you set this parameter to True, Default Master ID will be read only. For more information, refer to "Hard-coded Default Master" **Values:** ■ false (0) ■ true (1) **Default Value:** 0 **Enabled:** AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** HC_DFLT_MSTR
Include Early Burst Termination Support	The Early Burst Termination logic is included when this parameter is set; otherwise the ahbarbint signal is removed. However, when this is set to False, it does not indicate that there will be no early burst termination performed by the arbiter. For more information, refer to "Early Burst Termination" **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** EBTEN

Table 3-9 Slave Arbiter Configuration Parameters (Continued)

Label	Description
Generate slave select on the interface	If set to True, the AHB interface signals for Slave 0 are provided as outputs from DW_ahb for reference. If set to False, AHB interface signals are not provided as outputs. These signals are hsel_s0, hready_resp_s0, hrdata_s0, and htrans_s0. In coreConsultant, this parameter is selected automatically depending on how AHB_HAS_ARBIF and AHB_LITE parameters are configured. ■ If AHB_LITE=1 or AHB_HAS_ARBIF=0, then this parameter is set to False. ■ If AHB_LITE=0 and AHB_HAS_ARBIF=1, then this parameter is set to True. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** 0 **Parameter Name:** GEN_HSEL0

3.10 Weighted Token Arbitration Parameters

Table 3-10 Weighted Token Arbitration Parameters

Label	Description
Weighted Token Arbitration	
Counting Mode	The token counters can count on clock cycles or on bus cycles to calculate the number of tokens a master is using. **Values:** ■ Clock-Cycle (0) ■ Bus-Cycle (1) **Default Value:** Clock-Cycle **Enabled:** AHB_WTEN == 1 && AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** AHB_TPS_MODE
Bits in Arbitration Counter	The width of the total counter is configurable and is used to reduce the number of registers required when the design is configured. The counter should be wide enough to count the sum of all the individual master clock tokens. **Values:** 4, ..., 32 **Default Value:** 32 **Enabled:** AHB_WTEN == 1 && AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** AHB_TCL_WIDTH
Bits in Master Token Counter	The width of the master counter is configurable and is used to reduce the number of registers required when the design is configured. Each master counter is the same width and needs to be wide enough to count the correct number of tokens for a master. **Values:** 4, ..., 32 **Default Value:** 32 **Enabled:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter **Parameter Name:** AHB_CCL_WIDTH
Use Hard-coded Tokens	The length of the arbitration period and the number of clock tokens for each master can be hardcoded to reduce the overall register count. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** AHB_WTEN == 1 && AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** AHB_HC_TOKENS

Table 3-10 Weighted Token Arbitration Parameters (Continued)

Label	Description
Total Cycle Limit	An arbitration period is defined over this number of cycles. When a new arbitration period starts, the master counters are reloaded. On the interface, the output ahb_wt_aps gives a one-cycle pulse when a new arbitration period begins. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xffff **Enabled:** AHB_WTEN == 1 && AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** AHB_TCL
Master i Clock Tokens (for i = 1; i <= NUM_AHB_MASTERS)	Each master is assigned a number of clock tokens that it can use and be guaranteed to get this number of cycles over an arbitration period. Masters with remaining tokens have priority over masters that have used all of their tokens. User-configured token values are summed to ensure that they do not exceed the total allocated number of tokens. A user can specify any number of tokens for a master. The larger the value, the more the number of tokens. To facilitate an infinite number of tokens, the value of 0 represents infinite tokens. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0xf **Enabled:** AHB_WTEN == 1 && AHB_HAS_ARBIF == 1 && AHB_LITE == 0 **Parameter Name:** AHB_CL_M(i)

4 Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- Clock and Resets on page 75
- Observability Signals on page 76
- Control Signals on page 78
- Master Signals on page 79
- External Decoder Signals on page 84
- Arbiter Slave Interface Signals on page 85
- Slave Signals on page 87
- Interrupt Signals on page 92

4.1 Clock and Resets Signals

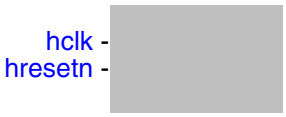
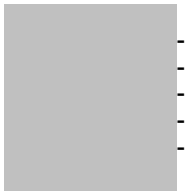


Table 4-1 Clock and Resets Signals

Port Name	I/O	Description
hclk	I	AHB Clock Signal. This clock times all bus transfers. All signal timings are related to the rising edge of hclk. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresetn	I	AHB Reset Signal. The bus reset signal is active low and is used to reset the system and the bus on the DesignWare Synthesizable Components interface. During reset, all DW_ahb masters must ensure that address and control signals are at valid levels, and that htrans_m(j) indicates the IDLE state. Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of hclk. DW_ahb 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

4.2 Observability Signals



- ahb_wt_mask
- ahb_wt_aps
- ahb_wt_count_m(i) (for i = 1; i <= NUM_AHB_MASTERS)
- hmaster
- hmaster_data

Table 4-2 Observability Signals

Port Name	I/O	Description
ahb_wt_mask[NUM_AHB_MASTERS:1]	O	Weighted token mask. Each bit of the bus represents the weighted token mask for that master, and it is active when a master has expired its assigned clock tokens. It is used for observation and verification of the DW_ahb. When a master has used its clock tokens, the priority changes where masters with no tokens are a lower priority than masters with tokens. **Exists:** AHB_WTEN==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ahb_wt_aps	O	It is used for observation and verification of the DW_ahb. The calculation of ownership of the bus is over an arbitration period, which starts once the weighted token mode is enabled and repeats after a fixed period. When an arbitration period starts, the weighted token masks are removed as master tokens are refreshed. **Exists:** AHB_WTEN==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ahb_wt_count_m(i)[(AHB_CCL_WIDTH-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	O	Weighted clock token outputs that help fine-tune the number of tokens that one can assign to a master. These debug outputs show the number of tokens a master has left. **Exists:** (AHB_WTEN==1) && (NUM_AHB_MASTERS>i) && (AHB_WTEN_DEBUG==1) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 Observability Signals (Continued)

Port Name	I/O	Description
hmaster[(HMASTER_WIDTH-1):0]	O	When AHB5_HMASTER_WIDTH=0, this signal indicates which master currently has ownership of the address and control bus. This is generated by the arbiter. When AHB5_HMASTER_WIDTH>0, this signal is mapped with the master identifier signal hmaster_m1, generated by master, which can be used to differentiate between multiple exclusive threads of master. **Exists:** Always **Synchronous To:** hclk **Registered:** AHB_LITE==1 ? No : (NUM_AHB_MASTERS>=8 ? Yes : (NUM_AHB_MASTERS>=4 ? "2:0=Yes;3:3=No" : (NUM_AHB_MASTERS>=2 ? "1:0=Yes;3:2=No" : "3:1=No;0:0=Yes"))) **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hmaster_data[(HMASTER_WIDTH-1):0]	O	Indicates which master currently has ownership of the data bus. **Exists:** Always **Synchronous To:** hclk **Registered:** AHB_LITE==1 ? No : (NUM_AHB_MASTERS>=8 ? Yes : (NUM_AHB_MASTERS>=4 ? "2:0=Yes;3:3=No" : (NUM_AHB_MASTERS>=2 ? "1:0=Yes;3:2=No" : "3:1=No;0:0=Yes"))) **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 Control Signals

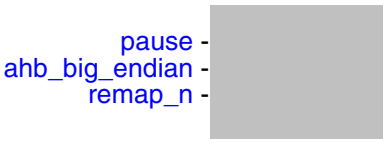


Table 4-3 Control Signals

Port Name	I/O	Description
pause	I	Asserted to put the arbiter into low-power mode by granting the dummy master ownership of the bus. **Exists:** PAUSE==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ahb_big_endian	I	AHB Endianness. **Exists:** AHB_XENDIAN==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
remap_n	I	Selection of memory map. When there are two memory maps, they are selected by remap_n. The Boot Mode (REMAP = 1) is selected when remap_n is Low; Normal Mode is selected when remap_n is High. When there is only one memory map, remap_n is not included in the top-level I/O. It is always "1" when there is only one memory map. **Exists:** REMAP==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.4 Master Signals

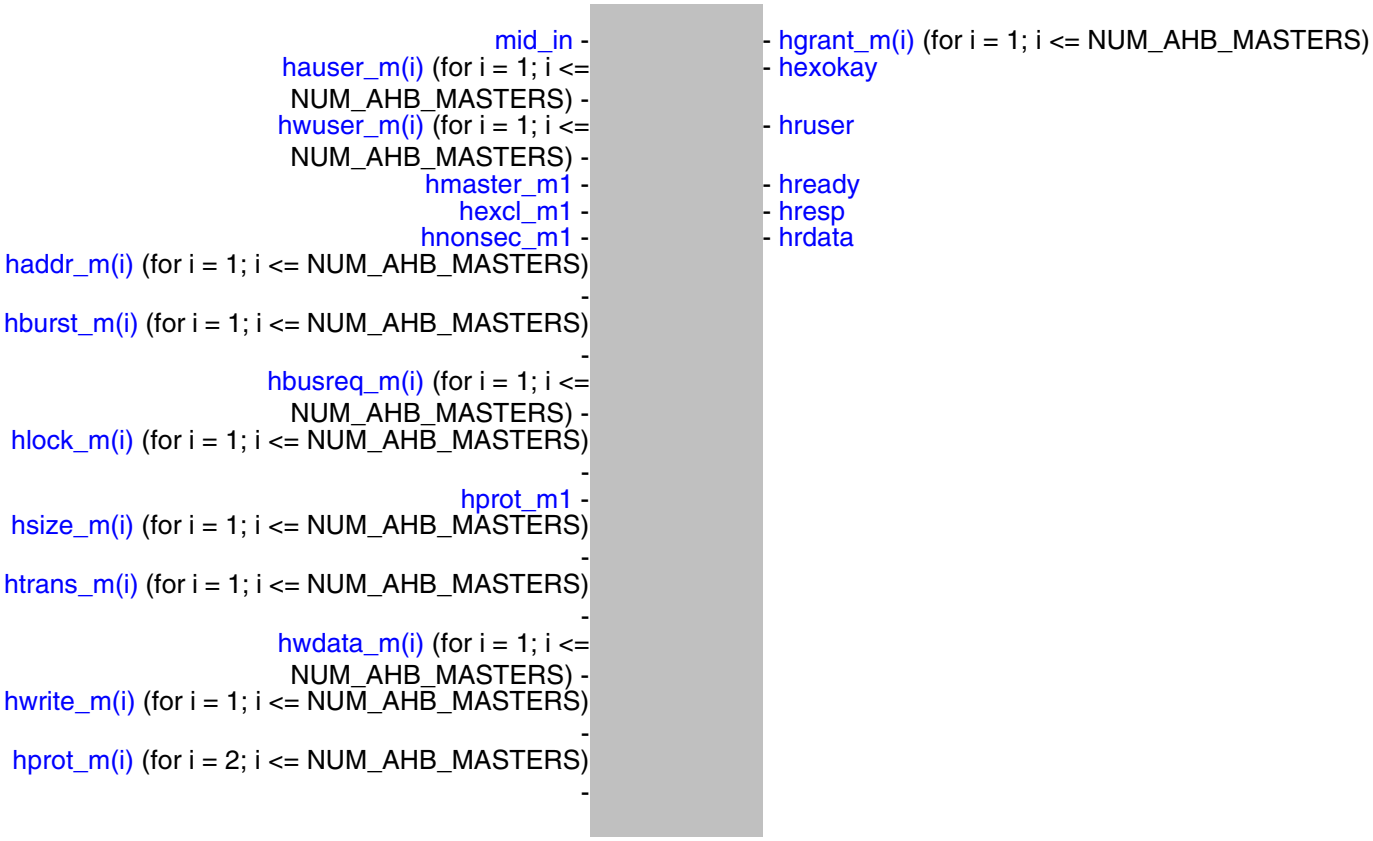


Table 4-4 Master Signals

Port Name	I/O	Description
mid_in[(MID_WIDTH-1):0]	I	Non standard Master ID sideband signal input (in AHB-Lite mode). **Exists:** MID_WIDTH!=0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hauser_m(i)[(AHB_A_UBW-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Master user bus for address channel. **Exists:** (AHB_A_UBW>0) && (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA

Table 4-4 Master Signals (Continued)

Port Name	I/O	Description
hwuser_m(i)[(AHB_W_UBW-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Master user bus for address channel. **Exists:** (AHB_W_UBW>0) && (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA
hmaster_m1[(AHB5_HMASTER_WIDTH-1):0]	I	Master identifier input from Master. This signal can be used to differentiate between multiple exclusive threads of Master. This signal can be used for purposes other than Exclusive Transfers. **Exists:** AHB5_HMASTER_WIDTH>0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA
hexcl_m1	I	Master Exclusive transfer type. When asserted, this signal indicates that the current transfer is part of an Exclusive access sequence. **Exists:** AHB_HAS_EXC_XFER==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hnonsec_m1	I	Master Non-secure or Secure transfer type. When asserted, this signal indicates that the current transfer is a Non-secure transfer. When de-asserted the transfer is a Secure transfer. **Exists:** AHB_HAS_SECURE_XFER==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
haddr_m(i)[(HADDR_WIDTH-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	I	AHB Address Bus. Address bus for each master in the system. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Signals (Continued)

Port Name	I/O	Description
hburst_m(i)[(HBURST_WIDTH-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Indicates if the transfer constitutes part of a burst. Each master in the system has its own hburst_m(i) bus. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hbusreq_m(i) (for i = 1; i <= NUM_AHB_MASTERS)	I	Bus request signal. Asserted by master to request access to the bus. There is one hbusreq_m(i) signal per master in the system. This signal is not included in an AHB Lite configuration **Exists:** (NUM_AHB_MASTERS>=i) && (AHB_LITE==0) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hlock_m(i) (for i = 1; i <= NUM_AHB_MASTERS)	I	Asserted by bus master to indicate that it wishes to carry out a locked transaction. There is one hlock_m(i) signal for each master in the system. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hprot_m1[(HPROT_WIDTH-1):0]	I	Master 1 Protection Control Signals. If the configuration parameter AHB_HAS_EXTENDED_MEMTYPE is set to 1, hprot_m1 is 7-bit signal else 4-bit signal. The 3-bit extension of the Master protection control signal, hprot_m1[6:4], indicates the extended memory types. These bits indicate the lookup, allocate and shareable attributes of the transfer. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hsize_m(i)[2:0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Indicates size of transfer. There is one hsize_m(i) bus for each master in the system. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Signals (Continued)

Port Name	I/O	Description
htrans_m(i)[1:0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Indicates the type of transfer being performed. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwdata_m(i)[(AHB_DATA_WIDTH-1):0] (for i = 1; i <= NUM_AHB_MASTERS)	I	Transfer write data. Each master has its own hwdata_m(i) signal. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwrite_m(i) (for i = 1; i <= NUM_AHB_MASTERS)	I	Transfer write signal. When HIGH, this signal indicates a write transfer. When LOW, this signal indicates a read transfer. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hgrant_m(i) (for i = 1; i <= NUM_AHB_MASTERS)	O	Asserted by arbiter to indicate that the requesting master has won ownership of the bus. There is a separate hgrant_m(i) signal for each master. This signal is not included in an AHB Lite configuration. **Exists:** (NUM_AHB_MASTERS>=i) && (AHB_LITE==0) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
hprot_m(i)[(HPROT_WIDTH-1):0] (for i = 2; i <= NUM_AHB_MASTERS)	I	Protection Control Signals. There is one hprot_m(i) bus for each master in the system. **Exists:** (NUM_AHB_MASTERS>=i) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Signals (Continued)

Port Name	I/O	Description
hexokay	O	Exclusive transfer response for Master. The exclusive transfer response received from the slave, hexokay_s(j), is passed to the master through this signal. When asserted, this signal indicates that the Exclusive transfer has been successful. When deasserted it indicates that the Exclusive transfer has failed. **Exists:** AHB_HAS_EXC_XFER==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hruser[(AHB_R_UBW-1):0]	O	Master user bus for read data channel. **Exists:** AHB_R_UBW > 0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA
hready	O	Ready response from selected slave. This signal is passed to all AHB masters and slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hresp[(HRESP_WIDTH-1):0]	O	Transfer response. This signal is passed to all AHB masters. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hrdata[(AHB_DATA_WIDTH-1):0]	O	Transfer read data. The read data bus is used to transfer data from bus slaves to the bus master during read operations. This signal is passed to all AHB masters. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.5 External Decoder Signals

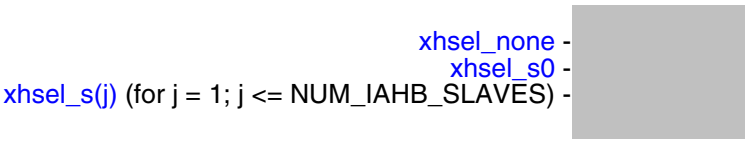


Table 4-5 External Decoder Signals

Port Name	I/O	Description
xhsel_none	I	Default slave select. **Exists:** AHB_HAS_XDCDR == 1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
xhsel_s0	I	Slave select from Arbiter Slave interface. **Exists:** (AHB_HAS_ARBIF == 1) && (AHB_HAS_XDCDR == 1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
xhsel_s(j)[(HSEL_S1_WIDTH-1):0] (for j = 1; j <= NUM_IAHB_SLAVES)	I	Slave select line. **Exists:** (NUM_IAHB_SLAVES>=j) && (AHB_HAS_XDCDR==1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.6 Arbiter Slave Interface Signals

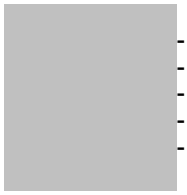


Table 4-6 Arbiter Slave Interface Signals

Port Name	I/O	Description
hsel_s0	O	When asserted, indicates that the arbiter slave has been selected. **Exists:** (GEN_HSEL0 == 1) && (AHB_HAS_ARBIF == 1) && ((AHB_HAS_XDCDR == 0) || ((AHB_HAS_XDCDR == 1) && (XDCDR_SUPPLIES_HSELS2AHB == 1))) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hready_resp_s0	O	Response from Arbiter Slave interface. When asserted, current transfer has completed. **Exists:** (AHB_HAS_ARBIF==1) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
hresp_s0[(HRESP_WIDTH-1):0]	O	Transfer response from Arbiter Slave interface. **Exists:** (AHB_HAS_ARBIF==1) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hrdata_s0[(AHB_DATA_WIDTH-1):0]	O	Readback data from Arbiter Slave interface. **Exists:** (AHB_HAS_ARBIF==1) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 Arbiter Slave Interface Signals (Continued)

Port Name	I/O	Description
hruser_s0[(AHB_R_UBW-1):0]	O	Readback data user from Arbiter Slave interface. **Exists:** (AHB_R_UBW>0) && (AHB_HAS_ARBIF==1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.7 Slave Signals

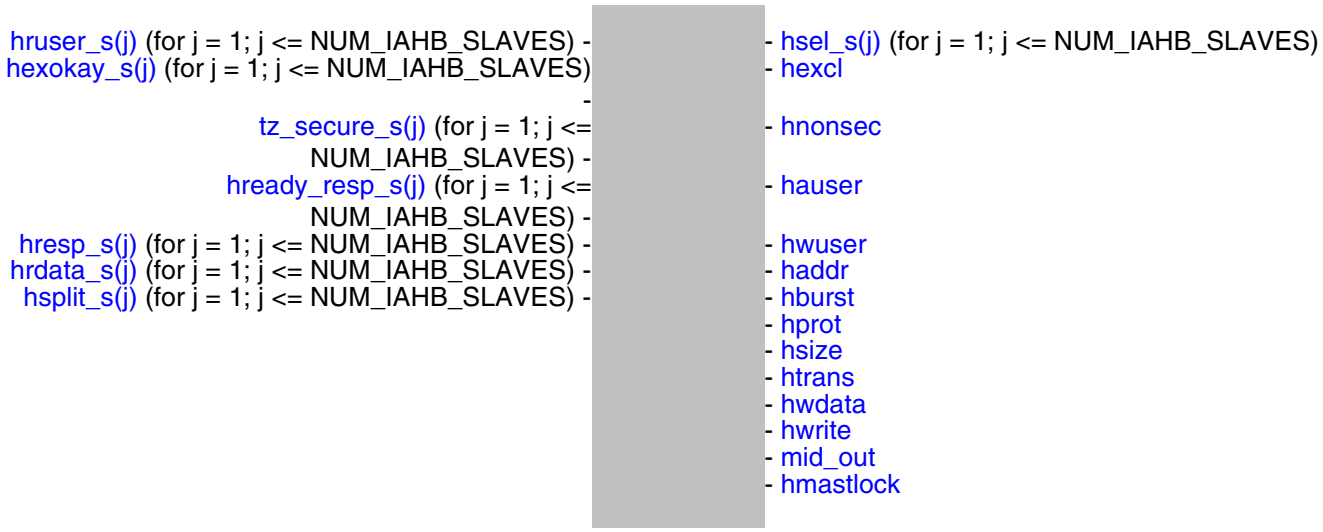


Table 4-7 Slave Signals

Port Name	I/O	Description
hruser_s(j)[(AHB_R_UBW-1):0] (for j = 1; j <= NUM_IAHB_SLAVES)	I	Slave user bus for read data channel. **Exists:** (NUM_IAHB_SLAVES >= j) && (AHB_R_UBW > 0) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA
hexokay_s(j) (for j = 1; j <= NUM_IAHB_SLAVES)	I	Exclusive transfer response from slave. When asserted, this signal indicates that Exclusive transfer has been successful. When deasserted it indicates that the Exclusive transfer has failed. **Exists:** (NUM_IAHB_SLAVES >= j) && (AHB_HAS_EXC_XFER == 1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-7 Slave Signals (Continued)

Port Name	I/O	Description
tz_secure_s(j) (for j = 1; j <= NUM_IAHB_SLAVES)	I	Slave Trustzone signal. This signal is asserted if attached slave is a secure slave; only secure access can be forwarded to secure slave. There is one tz_secure_s(j) signal for each slave in the system. **Exists:** (NUM_IAHB_SLAVES >= j) && (AHB_HAS_TZ_SUPPORT == 1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hsel_s(j)[(HSEL_S1_WIDTH-1):0] (for j = 1; j <= NUM_IAHB_SLAVES)	O	When asserted, the signal indicates that the slave has been selected. Each AHB slave has its own hsel_s(j) line. This is generated by the decoder block. **Exists:** (NUM_IAHB_SLAVES>=j) && ((AHB_HAS_XDCDR==0) || ((AHB_HAS_XDCDR==1) && (XDCDR_SUPPLIES_HSELS2AHB==1))) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hready_resp_s(j) (for j = 1; j <= NUM_IAHB_SLAVES)	I	Response from slave. When asserted, current transfer has completed. There is one hready_resp_s(j) for each slave in the system. **Exists:** (NUM_IAHB_SLAVES >= j) && (HSEL_ONLY_S(j) == 0) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hresp_s(j)[(HRESP_WIDTH-1):0] (for j = 1; j <= NUM_IAHB_SLAVES)	I	Transfer response from individual slave. There is one hresp_s(j) for each slave in the system. **Exists:** (NUM_IAHB_SLAVES >= j) && (HSEL_ONLY_S(j) == 0) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hrdata_s(j)[(AHB_DATA_WIDTH-1):0] (for j = 1; j <= NUM_IAHB_SLAVES)	I	Readback data from slaves. Each slave has its own hrdata bus. **Exists:** (NUM_IAHB_SLAVES>=j) && (HSEL_ONLY_S(j)==0) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-7 Slave Signals (Continued)

Port Name	I/O	Description
hsplit_s(j)[15:0] (for j = 1; j <= NUM_IAHB_SLAVES)	I	This bus is driven by split-capable slaves to indicate to the arbiter which master may proceed to complete a split transaction. Each split-capable slave drives its own hsplit_s(j) bus. **Exists:** (NUM_IAHB_SLAVES>=j) && (HSEL_ONLY_S(j)==0) && (SPLIT_CAPABLE_1==1) **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hexcl	O	Slave Exclusive transfer type. When asserted, this signal indicates to slave that current transfer is part of an Exclusive access sequence. **Exists:** AHB_HAS_EXC_XFER==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hnonsec	O	Slave Non-secure or Secure transfer type. When asserted, this signal indicates that the current transfer is a Non-secure transfer. When de-asserted the transfer is a Secure transfer. **Exists:** AHB_HAS_SECURE_XFER==1 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hauser[(AHB_A_UBW-1):0]	O	Slave user bus for address channel. **Exists:** AHB_A_UBW > 0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA
hwuser[(AHB_W_UBW-1):0]	O	Slave user bus for write data channel. **Exists:** AHB_W_UBW > 0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** NA

Table 4-7 Slave Signals (Continued)

Port Name	I/O	Description
haddr[(HADDR_WIDTH-1):0]	O	Address bus from selected master. This is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hburst[(HBURST_WIDTH-1):0]	O	Burst type from selected master. This is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hprot[(HPROT_WIDTH-1):0]	O	Transfer protection information from selected master. If the configuration parameter AHB_HAS_EXTENDED_MEMTYPE is set to 1, hprot is 7-bit signal else 4-bit signal. The 3-bit extension of the Master protection control signal, hprot[6:4], indicates the extended memory types. This signal indicates the lookup, allocate and shareable attributes of the transfer. Used only by slaves that can support this feature. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hsize[2:0]	O	Transfer size from selected master. Indicates the size of the transfer. This is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
htrans[1:0]	O	Transfer type from selected master. This is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-7 Slave Signals (Continued)

Port Name	I/O	Description
hwdata[(AHB_DATA_WIDTH-1):0]	O	Write data bus from selected master. This signal is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwrite	O	When High, this signal indicates a write transfer from selected master. When Low, this signal indicates a read transfer from selected master. This signal is passed to all AHB slaves. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mid_out[(MID_WIDTH-1):0]	O	Non standard Master ID sideband signal output (in AHB-Lite mode). **Exists:** MID_WIDTH!=0 **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hmastlock	O	Asserted to indicate that the transfer currently in progress is part of a locked transaction. This signal is driven by the arbiter. **Exists:** Always **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.8 Interrupt Signals

Table 4-8 Interrupt Signals

Port Name	I/O	Description
ahbarbint	O	Interrupt signal to Interrupt Controller. The arbiter will flag an interrupt when an Early Burst Termination occurs. **Exists:** (AHB_LITE==0) && (EBTEN==1) **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

5
Register Descriptions

This chapter details all possible registers in the IP. They are arranged hierarchically into maps and blocks (banks).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 controller 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 controller 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

These 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 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 once to this register 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_mem_map

Address Block	Description
DW_ahb_addr_block on page 96	DW_ahb address block **Exists:** Always

5.1 DW_ahb_mem_map/DW_ahb_addr_block Registers

DW_ahb 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_mem_map/DW_ahb_addr_block

Register	Offset	Description
AHB_PL(i) (for i = 1; i <= 15) on page 97	B+ 0x00 + 0x04*(i-1)	Arbitration Priority Master i Register
AHB_EBTCOUNT on page 98	B+0x3c	Early Burst Termination Count Register
AHB_EBT_EN on page 99	B+0x40	Early Burst Termination Enable
AHB_EBT on page 100	B+0x44	Early Burst Termination Register
AHB_DFLT_MASTER on page 101	B+0x48	Default Master ID Number Register
AHB_WTEN on page 103	B+0x4c	Weighted-Token Arbitration Scheme Enable Register
AHB_TCL on page 104	B+0x50	Master clock refresh period
AHB_CCLM(i) (for i = 1; i <= 15) on page 105	B+ 0x50 + 0x04*i	Master i Clock Token Register
AHB_VERSION_ID on page 106	B+0x90	Component Version ID Register

5.1.1 AHB_PL(i) (for i = 1; i <= 15)

- **Name:** Arbitration Priority Master i Register
- **Description:** This register provides the programmability to configure the priority levels, which is required for arbitration.
- **Size:** 32 bits
- **Offset:** B+ 0x00 + 0x04*(i-1)
- **Exists:** (AHB_LITE==0) && (NUM_AHB_MASTERS>=i)

Table 5-6 Fields for Register: AHB_PL(i) (for i = 1; i <= 15)

Bits	Name	Memory Access	Description
31:4	RSVD_AHB_PRIORITY_LEVELi	R	RSVD_AHB_PRIORITY_LEVELi Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
3:0	AHB_PRIORITY_LEVELi	R/W	Arbitration priority for master i register used for programming the AHB master priority. Priority level 1 is having the lowest priority, and 15 is having highest. For more information, refer the section "Master Priority Level Registers". **Value After Reset:** PRIORITY_[i] **Exists:** Always

5.1.2 AHB_EBTCOUNT

- **Name:** Early Burst Termination Count Register
- **Description:** Early Burst Termination Count Register. For more information refer the section "Early Burst Termination".
- **Size:** 32 bits
- **Offset:** B+0x3c
- **Exists:** (AHB_LITE ==0 && EBTEN==1)

Table 5-7 Fields for Register: AHB_EBTCOUNT

Bits	Name	Memory Access	Description
31:10	RSVD_AHB_EBTCOUNT	R	RSVD_AHB_EBTCOUNT Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always
9:0	AHB_EBTCOUNT	R/W	Early burst termination count register. Maximum number of cycles a transfer can take before being subject to an early burst termination. **Value After Reset:** 0x0 **Exists:** Always

5.1.3 AHB_EBT_EN

- **Name:** Early Burst Termination Enable
- **Description:** Early Burst Termination Enable. For more information refer the section "Early Burst Termination"
- **Size:** 32 bits
- **Offset:** B+0x40
- **Exists:** (AHB_LITE ==0 && EBTEN==1)

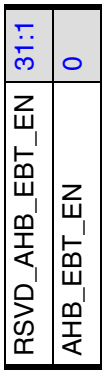


Table 5-8 Fields for Register: AHB_EBT_EN

Bits	Name	Memory Access	Description
31:1	RSVD_AHB_EBT_EN	R	RSVD_AHB_EBT_EN Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	AHB_EBT_EN	R/W	Early burst termination enable is to enable or disable EBT through the software programming. **Values:** ■ 0x0 (DISABLED): Disables the Early Burst Termination ■ 0x1 (ENABLED): Enables the Early Burst Termination **Value After Reset:** 0x0 **Exists:** Always

5.1.4 AHB_EBT

- **Name:** Early Burst Termination Register
- **Description:** Early Burst Termination Register. For more information refer the section "Early Burst Termination"
- **Size:** 32 bits
- **Offset:** B+0x44
- **Exists:** (AHB_LITE ==0 && EBTEN==1)

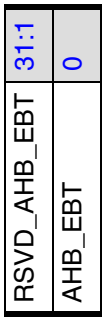


Table 5-9 Fields for Register: AHB_EBT

Bits	Name	Memory Access	Description
31:1	RSVD_AHB_EBT	R	RSVD_AHB_EBT Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	AHB_EBT	R	Early burst termination register. Set when an Early Burst Termination takes place. The register is cleared when read by the processor. **Values:** ■ 0x1 (EBT): Early Burst Termination Occured. ■ 0x0 (NO_EBT): No Early Burst Termination Occured. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.5 AHB_DFLT_MASTER

- **Name:** Default Master ID Number Register
- **Description:** Default Master ID Number Register. For more information refer section "Default Master versus Dummy Master".
- **Size:** 32 bits
- **Offset:** B+0x48
- **Exists:** !AHB_LITE

Table 5-10 Fields for Register: AHB_DFLT_MASTER

Bits	Name	Memory Access	Description
31:4	RSVD_AHB_DFLT_MASTER	R	RSVD_AHB_DFLT_MASTER Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always

Table 5-10 Fields for Register: AHB_DFLT_MASTER (Continued)

Bits	Name	Memory Access	Description
3:0	AHB_DFLT_MASTER	* Varies	Default master ID number register. The default master is the master that is granted by the bus when no master has requested ownership. **Dependencies**: ■ DFT_MST is Read Only if HC_DFLT_MSTR = 1 or AHB_HAS_ARBIF = 1'b0. ■ If HC_DFLT_MSTR = 0 and AHB_HAS_ARBIF = 1'b1, this register accepts only writes for valid master numbers. Any write of a value greater than `NUM_AHB_MASTERS results in the DFT_MST register being set to 0x0. **Value After Reset:** DFLT_MSTR_NUM **Exists:** Always **Memory Access:** {(HC_DFLT_MSTR == 1 || AHB_HAS_ARBIF==0) ? "read-only" : "read-write"}

5.1.6 AHB_WTEN

- **Name:** Weighted-Token Arbitration Scheme Enable Register
- **Description:** Weighted-Token Arbitration Scheme Enable Register. For more information refer section "Weighted-Token Arbitration".
- **Size:** 32 bits
- **Offset:** B+0x4c
- **Exists:** (AHB_LITE==0 && AHB_WTEN ==1)

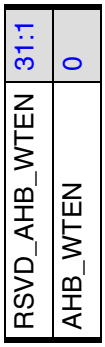


Table 5-11 Fields for Register: AHB_WTEN

Bits	Name	Memory Access	Description
31:1	RSVD_AHB_WTEN	R	RSVD_AHB_WTEN Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always
0	AHB_WTEN	R/W	Weighted-token arbitration scheme enable register. Accessible only when the weighted token scheme is configured through coreConsultant. **Values:** ■ 0x0 (DISABLED): Disables Weighted-Token Arbitration Scheme ■ 0x1 (ENABLED): Enables Weighted-Token Arbitration Scheme **Value After Reset:** 0x0 **Exists:** Always

5.1.7 AHB_TCL

- **Name:** Master clock refresh period
- **Description:** Master clock refresh period. For more information refer the section "Weighted-Token Arbitration"
- **Size:** 32 bits
- **Offset:** B+0x50
- **Exists:** (AHB_LITE==0 && AHB_WTEN ==1)

Table 5-12 Fields for Register: AHB_TCL

Bits	Name	Memory Access	Description
31:y	RSVD_AHB_TCL	R	RSVD_AHB_TCL Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** AHB_TCL_WIDTH
x:0	AHB_TCL	R/W	Master clock refresh period. Accessible only when the weighted token scheme is configured through coreConsultant. Value can be hardcoded to reduce the register count. Can count clock cycles or bus cycles. **Value After Reset:** AHB_TCL **Exists:** Always **Range Variable[x]:** AHB_TCL_WIDTH - 1

5.1.8 AHB_CCLM(i) (for i = 1; i <= 15)

- **Name:** Master i Clock Token Register
- **Description:** Master Clock Token Register
- **Size:** 32 bits
- **Offset:** B+ 0x50 + 0x04*i
- **Exists:** (AHB_LITE==0) && (NUM_AHB_MASTERS>=i) && (AHB_WTEN==1)

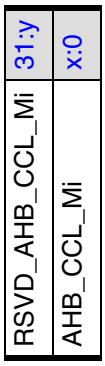


Table 5-13 Fields for Register: AHB_CCLM(i) (for i = 1; i <= 15)

Bits	Name	Memory Access	Description
31:y	RSVD_AHB_CCL_Mi	R	RSVD_AHB_CCL_Mi Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** AHB_CCL_WIDTH
x:0	AHB_CCL_Mi	* Varies	Number of tokens a master can use on the bus before it has to arbitrate on the first-tier rather than the upper-tier. Accessible only when the weighted token scheme is configured through coreConsultant and the relevant number of masters is specified. If a value of zero is configured, then the bus is deemed to have infinite tokens and will always operate in the upper-tier of arbitration. Clock tokens or bus cycle tokens for a master can be hardcoded or programmable. **Value After Reset:** AHB_CL_M[i] **Exists:** Always **Range Variable[x]:** AHB_CCL_WIDTH - 1 **Memory Access:** {(AHB_HC_TOKENS == 1 || AHB_HAS_ARBIF==0) ? "read-only" : "read-write"}

5.1.9 AHB_VERSION_ID

- **Name:** Component Version ID Register
- **Description:** Component Version ID Register. This register provides the component version ID.
- **Size:** 32 bits
- **Offset:** B+0x90
- **Exists:** [<functionof> "(AHB_HAS_ARBIF==1)"]

Table 5-14 Fields for Register: AHB_VERSION_ID

Bits	Name	Memory Access	Description
31:0	AHB_VERSION_ID	R	ASCII value for each number in the version, followed by *. For example 32_30_31_2A represents the version 2.01*. **Value After Reset:** AHB_VERSION_ID **Exists:** Always

6 Programming the DW_ahb

This chapter describes the programmable features of the DW_ahb.

6.1 Programming Considerations

The following sections outline programming considerations for the DW_ahb.

6.1.1 Operation Modes

If you do not enable the Remap feature in coreConsultant, only one memory map is generated for Normal Mode, the default operation. Because the DW_ahb supports the DesignWare Remap Feature, there is the possibility that two memory maps are generated—one for Normal Mode and one for Boot Mode—if you have configured the DW_ahb to use the internal decoder (AHB_HAS_XDCDR = 0).

For an example memory map for both Boot and Normal modes, refer to Figure 2-6.

6.1.2 Arbiter Slave Interface Registers

"Register Descriptions" on page 93 describes the memory map for the programmable registers that make up the optional arbiter slave interface, which exists when AHB_HAS_ARBIF=1. The memory map includes the various programmable registers for arbitration priority levels, early burst termination, the default master ID number, weighted token, and coreKit version ID. These registers are discussed later in this section.

All arbiter registers appear on four-byte boundaries in the memory map. The arbiter is a 32-bit, little-endian AHB slave and is hard-coded as slave 0. The base address, or location of the arbiter slave, within your memory map is programmable through coreConsultant.

6.1.3 Master Priority Level Registers

Each master in the DW_ahb system has its own 4-bit priority configuration register, which shows the priority of each master. The priority level ranges from 0 to 15, with a priority 0 signifying that the master has been disabled. Highest priority is given to masters with priority 15, while masters with priority 1 have the lowest level of priority.

Each priority register is named PL*i*, where *i* denotes the number of the master in question, and "PL" stands for "priority level." The registers are aligned to longword (four-byte) boundaries.

For the DW_ahb, the reset priority level assigned to each master is configured through the Synopsys coreConsultant tool. The priority assigned to each master is the reset priority level. By default, it is equal to that master's number. For example, master 1 has priority 1, master 2 has priority 2, and so on. This means that by default, each master has a unique priority associated with it.

Note A master's priority level cannot be assigned to zero (disabled) in coreConsultant.

When you specify the parameters for DW_ahb, it is possible to hard code the priorities of each master. If you choose the hard-coded priority levels, the registers for the priorities within the memory map are not included within the design. It is possible to read the priorities, but it is not possible to change them. When the registers do exist, their reset condition is the value as specified by the user. When a master writes to any of the priority registers, hmaster is queried so that if a master attempts to write zero into its own priority register, thus masking it from the arbitration scheme, the write is ignored and the contents of the register remains unchanged.

Note Only the priority registers for up to NUM_AHB_MASTERS are generated. If there are fewer masters than the maximum of 15, then the corresponding arbitration priority register address locations are not valid addresses. If there is a read or write to any of these locations, a two-cycle error response is generated.

If you specify the gap between the start and end address for the arbiter slave interface as greater than 1 KB, the memory map detailed in "Register Descriptions" on page 93 is aliased at 1 KB intervals throughout the arbiter memory space because the arbiter interface uses only haddr[9:0] to decode the register space. For example, if you were to access address location Base+0x400, you would be accessing the arbitration priority for master 1, provided that the gap between the start and end addresses is greater than 1 KB. Any attempt to access locations within the 1 KB block, other than those specified, results in an error.

6.1.4 Early Burst Termination Registers

The Early Burst Termination feature is enabled when the configuration parameter EBTEN is set to True. The DW_ahb supports this feature by providing a programmable 10-bit counter in the arbiter, EBTCOUNT. This counter can be enabled or disabled by writing to an additional register, EBT_EN. If EBT_EN = 1 (is enabled) and a master assumes ownership of the bus, the counter assumes the value contained in the EBTCOUNT register. The counter decrements on each clock tick during the transfer. If it reaches zero before the transfer completes, then the arbiter terminates the master's ownership and commence arbitration between other competing masters. The EBT register is read-only as far as masters are concerned. When there is an early burst termination, the EBT register is asserted. It is cleared whenever the register is read.

As only one master has ownership of the bus at any given time, only one counter needs to be instantiated to implement the Early Burst Termination capability. The counter is disabled by default, and is cleared when reset. When the Early Burst Termination count expires, indicating the early termination of a burst, an interrupt is sent to the Interrupt Controller by ahbarbint going active. This signal stays active until it is cleared by a master/CPU.

6.1.5 Default Master Register

If no master is requesting bus access, the DW_ahb arbiter grants the bus to the default master. You can program the ID number of the designated default master in the DFT_MST register. Unless you explicitly program this register, the dummy master also acts as the default master. It is possible for you to hard code the index of the default master in coreConsultant. In this case, the register for the default master within the memory map is not included within the design. It is possible to read what the default master is, but it is not possible to change it. When the register does exist, its reset condition is the value as specified by the user in coreConsultant.

6.1.6 Weighted-Token Arbitration Registers

The weighted token scheme can be enabled through coreConsultant (AHB_WTEN). When it is not enabled, then the arbitration scheme is the normal scheme where masters are not restricted to the amount of time they spend on the bus. Once enabled, then each master's tokens are loaded at the start of an arbitration period into counters that decrement each time the corresponding master is granted the bus.

The arbitration period (AHB_TCL) is the number of clock tokens that the arbitration is carried over. Each master's token value plus two is added to give the minimum total arbitration period. The value needs to be at least the minimum; otherwise master may not get sufficient time on the bus. When a new arbitration period starts, masking is cleared for a master that has used its clock tokens. An arbitration period starts when the following occurs:

- Weighted token enable is written to
- Whenever a master comes out of pause mode
- Whenever the arbitration period counter reaches zero and restarts

The arbitration period can be seen with the pulse ahb_wt_aps, which pulses high for one cycle when the count begins. The arbitration period can be hardcoded or programmable. The width of the counter is configurable and needs to be wide enough to sum all the individual master tokens.

Each master is allocated a number of tokens that it needs to have to operate at the upper-tier arbitration scheme (AHB_CL_M(i)). The number of tokens can be hardcoded or left programmable. The width of the counters that count the number of used tokens can be controlled through coreConsultant. A master uses a token each time it is on the bus, even if it is issuing idle cycles. When a master expires its token, it uses the first-tier arbitration scheme. An expired master can be observed by looking at ahb_wt_mask. There is one bit for each master in the system.

During configuration, the priorities of each master must be unique for use within the weighted token scheme. It is possible to reprogram the priority registers to the same values (such as equal priorities) after configuration and still have the weighted-token arbitration.

7 Verification

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

Note The DW_ahb verification testbench is built with DesignWare Verification IP (VIP). Make sure that you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

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

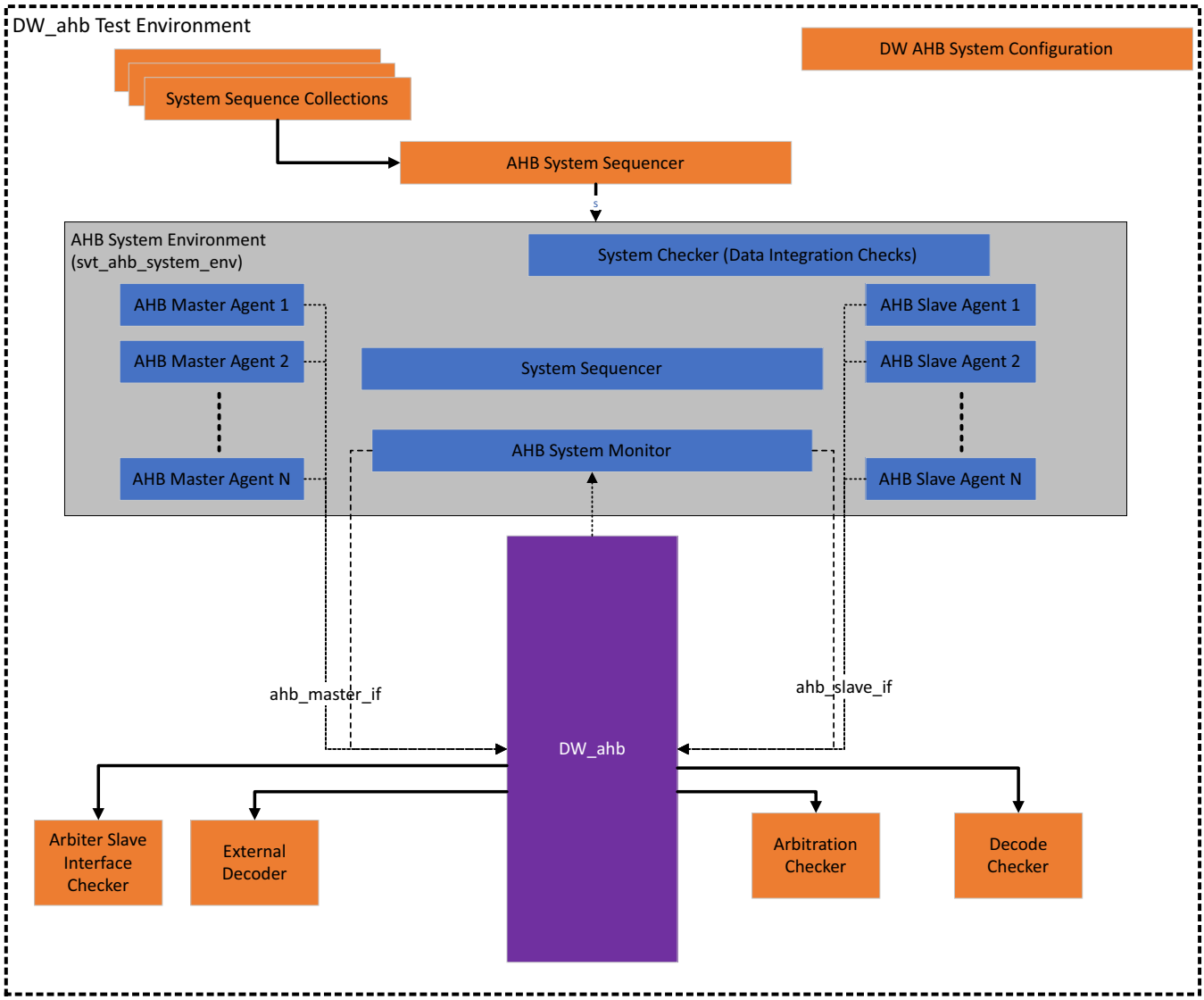
This chapter discusses the following sections:

7.1 Verification Environment

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

Figure 7-1 shows the verification environment of the DW_ahb testbench:

Figure 7-1 DW_ahb UVM Verification Environment



The testbench consists of the following elements:

- Testbench uses the standard SVT VIP for the protocol interfaces:
  - AHB VIP Interface for connecting master and slave ports of DUT.
  - AHB SVT VIP system environment:

    AHB master and slave agents, there are N master and M slave agents which drives request and response transactions respectively.

    N - Number of masters (configurable in coreConsultant using the NUM_AHB_MASTER parameter)

M - Number of slaves (configurable in coreConsultant using the NUM_IAHB_SLAVES parameter)

- Testbench uses the custom components:
  - External Decoder

    This component is responsible for generating slave select line by decoding the address when the AHB_HAS_XDCDR parameter is enabled.
  - Decoder Checker

    As the design has Internal Decoder, this component is responsible for internal decoder checking. There is a configuration class handle available in this component to pass required decoder configuration values. This component is responsible for providing checks for the following features during every cycle:
    - Remap
    - Multiple Address Regions
    - Slave Aliasing
    - Slave Visibility
    - Default Slave
  - Arbitration Checker

    As the design supports master port arbitration based on the selected arbitration algorithms, there should be a component checking the correctness of the port arbitration. This component is responsible for the arbitration checking. This component checks for the following features:
    - Arbitration Schemes
    - SPLIT/RETRY transfers
    - Default Master, Dummy master
    - Pause
    - Early Burst Termination
    - Full Incrementing Burst
  - Arbiter slave interface checker

    This checker is only active when the AHB_HAS_ARBIF parameter and the GEN_HSEL0 parameter are enabled. This is responsible for checking optional arbiter slave interface.
  - Sequence library

    The system sequence library consists of a set of sequence from VIP as well as a set of sequence developed to create traffic from design side.

7.2 Testbench Directories and Files

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

Table 7-1 shows the various directories and associated files:

Table 7-1 DW_ahb Testbench Directory Structure

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

7.3 Packaged Testcases

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

The associated test cases shipped, and their description is explained in Table 7-2:

Table 7-2 DW_ahb Test Description

Test Name	Test Description
test_100_rand_master_slave	This test verifies a basic bus functionality that a single master can access any random slave on the bus. There is no competition among the masters. Each master present in the system in turn requests the bus, gets granted, performs a number of transfers of different burst types and relinquish the bus. This test is used to prove that each master can gain access to the bus, performing a write/read operation successfully to any random slave.
test_101_arb_basic	This test is for verifying basic arbitration process of DW_ahb. This test is used to prove that the arbiter correctly grants competing requesting masters according to priority rules and AMBA protocol rules. As a part of this test, all the masters present in the system requests the bus simultaneously for the ownership of the bus, by asserting its respective request signals. Based on the priority algorithm, respective master is given grant to perform different burst types of random read/write transfers to any random slave. This test checks whether all the masters are given access, and whether the highest priority master is given the first grant.

Table 7-2 DW_ahb Test Description (Continued)

Test Name	Test Description
test_102_arb_split	This test verifies arbitration process when a particular slave response results as SPLIT. Whether the current master, which owns the bus, performing transfers is de-granted or not and some other master is taken the ownership of the bus or not when SPLIT response is occurred. Response types are chosen SPLIT and OKAY randomly from random slaves.
test_103_arb_retry	This test verifies arbitration process when a particular slave response results as RETRY. Whether the current master, which owns the bus, performing transfers is de-granted and continue to participate in arbitration and can win arbitration if it is highest priority one when RETRY response is occurred. Response types are chosen RETRY and OKAY randomly from random slaves.
test_104_arbif	The AHB slave interface holds the values of the master priorities, the index of the default master, settings for early burst termination, token counters settings for the weighted token arbitration scheme, and the coreKit version ID. Depending on the user configuration, some of these register locations does not exist. This test is included when the AHB_HAS_ARBIF parameter is enabled. This test verifies the arbif interface of DUT using a burst transfer for multiple registers we expect the data and verify with the data read.
test_105_ahb_lite	This test verifies DW_ahb when it is configured in the LITE mode. When AHB_LITE is set, then there is only one master connected to the bus (master1). This master sends random transactions to all slave. Checks that there are no SPLIT/RETRY responses from the slaves; all slaves are made non-split capable.

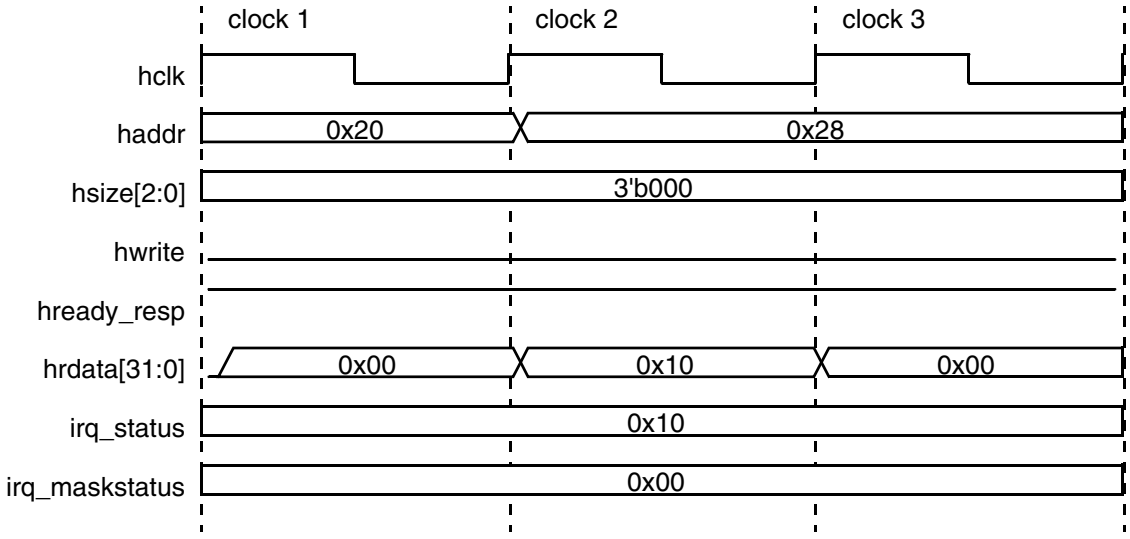
8 Integration Considerations

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

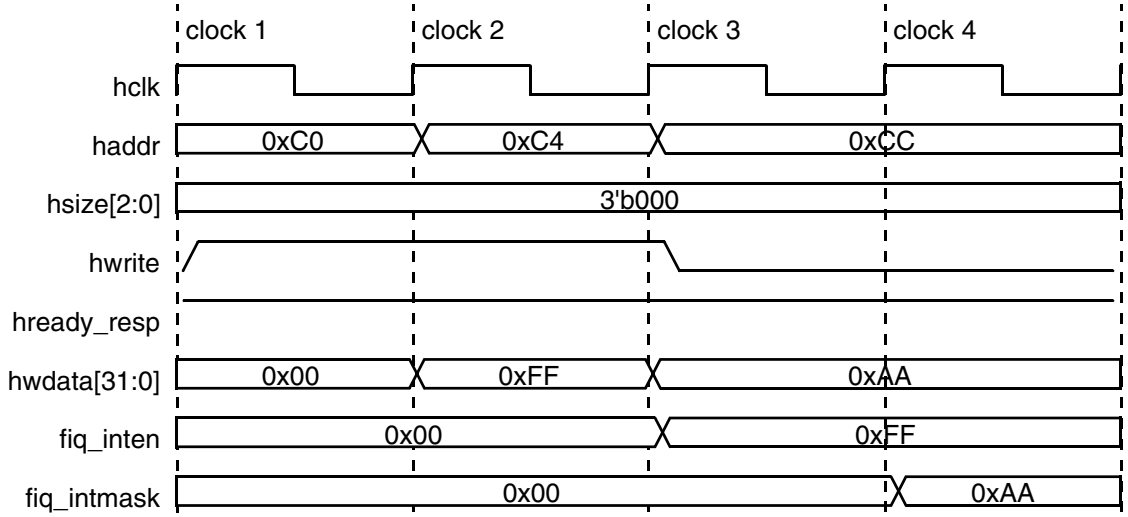


8.2 Write Accesses

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

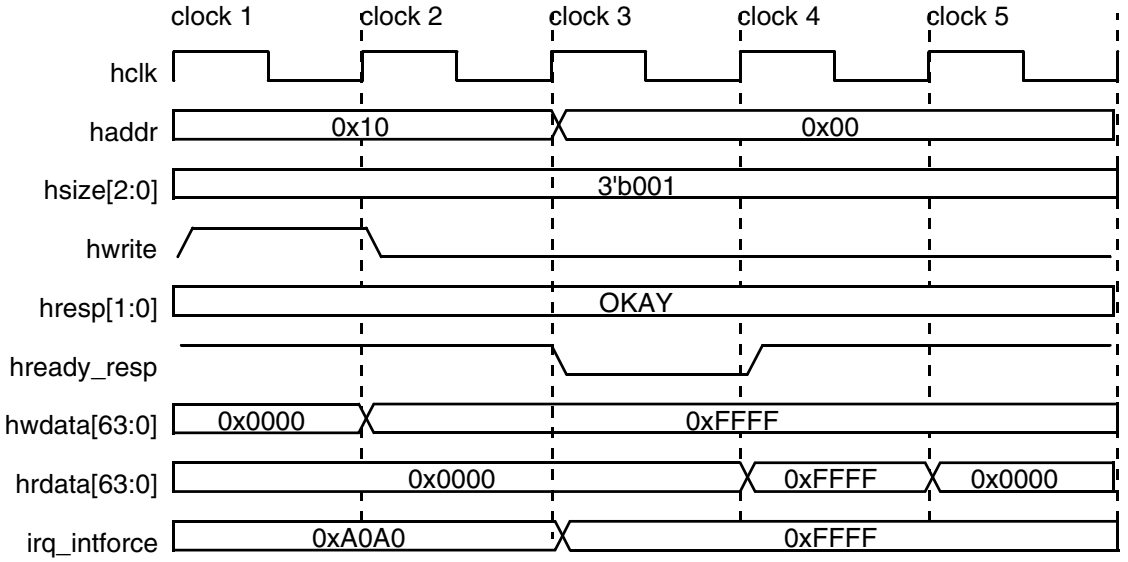


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

8.3 Consecutive Write-Read

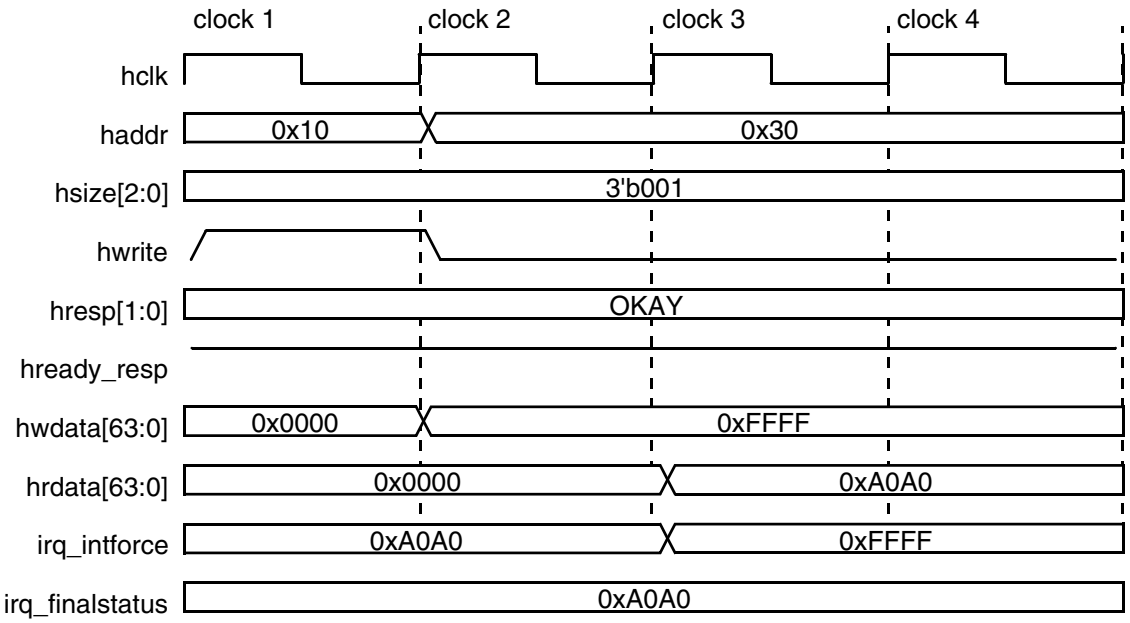
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_ictl inserts a “wait state” when it detects a read immediately 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.

8.5.1 Power Consumption, Frequency and Area, and DFT Coverage

Table 8-1 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage in DW_ahb using the industry standard 7nm technology library.

Table 8-1 Synthesis Results for DW_ahb

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	hclk = 300MHz	1835	5 nW	0.027 mW	100	99.98	99.5
Typical Configuration 1 AHB_LITE = 1 NUM_IAHB_SLAVES = 4 R1_N_EA_1 = 0x010007ff R1_N_SA_1 = 0x00000000 AHB_INTERFACE_TYPE = 1 AHB_HAS_EXTENDED_MEMTYPE = 1 AHB_HAS_SECURE_XFER = 1 AHB_HAS_TZ_SUPPORT = 1 AHB_HAS_EXC_XFER = 1 AHB5_HMASTER_WIDTH = 6 AHB_A_UBW = 256 AHB_W_UBW = 8 AHB_R_UBW = 32 AHB_DATA_WIDTH = 64	hclk = 300MHz	533	2 nW	0.002 mW	100	100	99.5

Table 8-1 Synthesis Results for DW_ahb (Continued)

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Typical Configuration 2 NUM_AHB_MASTERS = 15 NUM_IAHB_SLAVES = 15 AHB_DATA_WIDTH = 256 AHB_XENDIAN = 1 AHB_HAS_XDCDR = 1 AHB_HAS_ARBIF = 1 AHB_WTEN = 1 AHB_FULL_INCR = 1 EBTEN = 1 SPLIT_CAPABLE_1 = 1 SPLIT_CAPABLE_2 = 1 SPLIT_CAPABLE_4 = 1	hclk = 300MHz	22301	60 nW	0.252 mW	99.82	98.58	100

A

DesignWare Synthesizable Component Constants

Table A-1 provides the DesignWare Synthesizable Components constant definitions. These definitions can also be found in DW_amba_constants.v file in the src directory of your DW_ahb coreKit.

Table A-1 DesignWare Synthesizable Components Constant Definitions

DesignWare Synthesizable Components Constant	Value
HBURST_WIDTH	3
HMASTER_WIDTH	4
HPROT_WIDTH	4
HRESP_WIDTH	2
HSIZE_WIDTH	3
HSPLIT_WIDTH	16
HTRANS_WIDTH	2
HBURST Values	
SINGLE	3‘b000
INCR	3‘b001
WRAP4	3‘b010
INCR4	3‘b011
WRAP8	3‘b100
INCR8	3‘b101
WRAP16	3‘b110
INCR16	3‘b111

DesignWare Synthesizable Components Constant	Value
HRESP Values	
OKAY	2‘b00
ERROR	2‘b01
RETRY	2‘b10
SPLIT	2‘b11
HSIZE Values	
BYTE	3‘b000
HWORD	3‘b001
WORD	3‘b010
LWORD	3‘b011
DWORD	3‘b100
WORD4	3‘b101
WORD8	3‘b110
WORD16	3‘b111
HTRANS Values	
IDLE	2‘b00
BUSY	2‘b01
NONSEQ	2‘b10
SEQ	2‘b11
HWRITE/PWRITE Values	
READ	1‘b0
WRITE	1‘b1
Generic Definitions	
TRUE	1‘b1
FALSE	1‘b0
zero8	8‘b0
zero16	16‘b0
zero32	32‘b0
KBYTE	1024

B
Basic Core Module (BCM) Library

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

This chapter provides information about the BCMs used in DW_ahb.

B.1 BCM Library Components

Table B-1 describes the list of BCM library components used in DW_ahb.

Table B-1 BCM Library Components

BCM Module Name	BCM Description	DWBB Equivalent
DW_ahb_bcm01	Minimum/Maximum Value	DW_minmax
DW_ahb_bcm02	Universal Multiplexer	DW01_mux_any
DW_ahb_bcm53	Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme	DW_arb_2t

C
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 C-1 Internal Parameters

Parameter Name	Equals To
AHB_VERSION_ID	32'h3231352a
HBURST_WIDTH	3
HMASTER_WIDTH	(AHB5_HMASTER_WIDTH>0 ? AHB5_HMASTER_WIDTH : 4)
HPROT_WIDTH	(AHB_HAS_EXTENDED_MEMTYPE ==1 ? 7:4)
HRESP_WIDTH	(AHB_SCALAR_HRESP ==1 ? 1 : 2)
HSEL_S1_WIDTH	(AHB_MULTI_SLV_SEL ==1) ? ((MR_N1 > MR_B1) ? (1 + MR_N1) : (1 + MR_B1)) : 1
IDLE	2'b00

D
Glossary

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

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

peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.