SYNOPSYS®

DesignWare DW_ahb Databook

DW_ahb – Product Code

2.11a
June 2014

Copyright Notice and Proprietary Information Notice

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

http://www.synopsys.com/Company/Pages/Trademarks.aspx.

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

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

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

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	Oct 2012	Added the product code on the cover and in Table 1-1
2.10b	Nov 2011	Version change for 2011.11a release
2.10a	Oct 2011	Revised paragraph for Second Tier Arbitration explaining possibility of master being “starved” by bus
2.09a	Jun 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced “Related Documentation” section in Preface
2.09a	Sep 2010	Corrected names of include files and vcs command used for simulation
2.08a	Dec 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	Oct 2008	■ Updated description of DFT_MST register ■ Version change for 2008.10a release
2.07a	Jun 2008	Version change for 2008.06a release
2.06b	Jun 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, "Building and Verifying a Component or Subsystem" introduces you to using the DW_ahb within the coreAssembler and coreConsultant tools.
- Chapter 3, "Functional Description" describes the functional operation of the DW_ahb.
- Chapter 4, "Parameters" identifies the configurable parameters supported by the DW_ahb.
- Chapter 5, "Signals" provides a list and description of the DW_ahb signals.
- Chapter 6, "Registers" describes the programmable registers of the DW_ahb.
- Chapter 7, "Programming the DW_ahb" provides information needed to program the configured DW_ahb.
- Chapter 8, "Verification" provides information on verifying the configured DW_ahb.
- Chapter 9, "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, "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*.

Web Resources

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

Customer Support

To obtain support for your product:

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

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD)
    - Identify the hierarchy path to the DesignWare instance
    - Identify the timestamp of any signals or locations in the waveforms that are not understood
- Then, contact Support Center, with a description of your question and supplying the above information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained below, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

    Go to http://solvnet.synopsys.com/EnterACall and click on the link to enter a call.
    Provide the requested information, including:
    - **Product:** DesignWare Library IP
    - **Sub Product:** AMBA
    - **Tool Version:** <product version number>
    - **Problem Type:**
    - **Priority:**
    - **Title:** DW_ahb

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

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

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

Product Code

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

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

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

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

Component Name	Description
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 busses

1 Product Overview

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

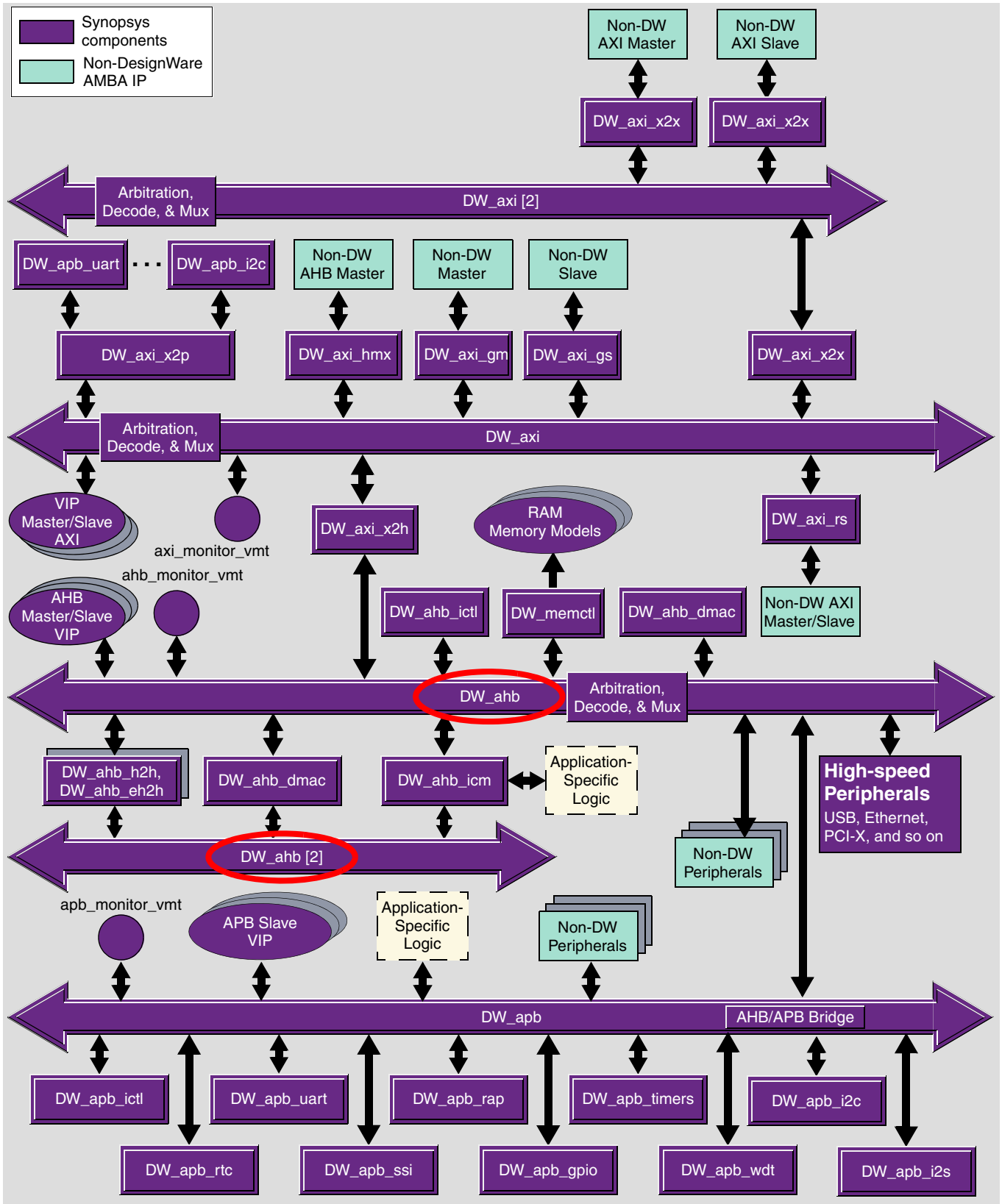
1.1 DesignWare System Overview

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

Attention Links resolve only if you are viewing this databook from your $DESIGNWARE_HOME tree, and to only those components that are installed in the tree.

Figure 1-1 Example of DW_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

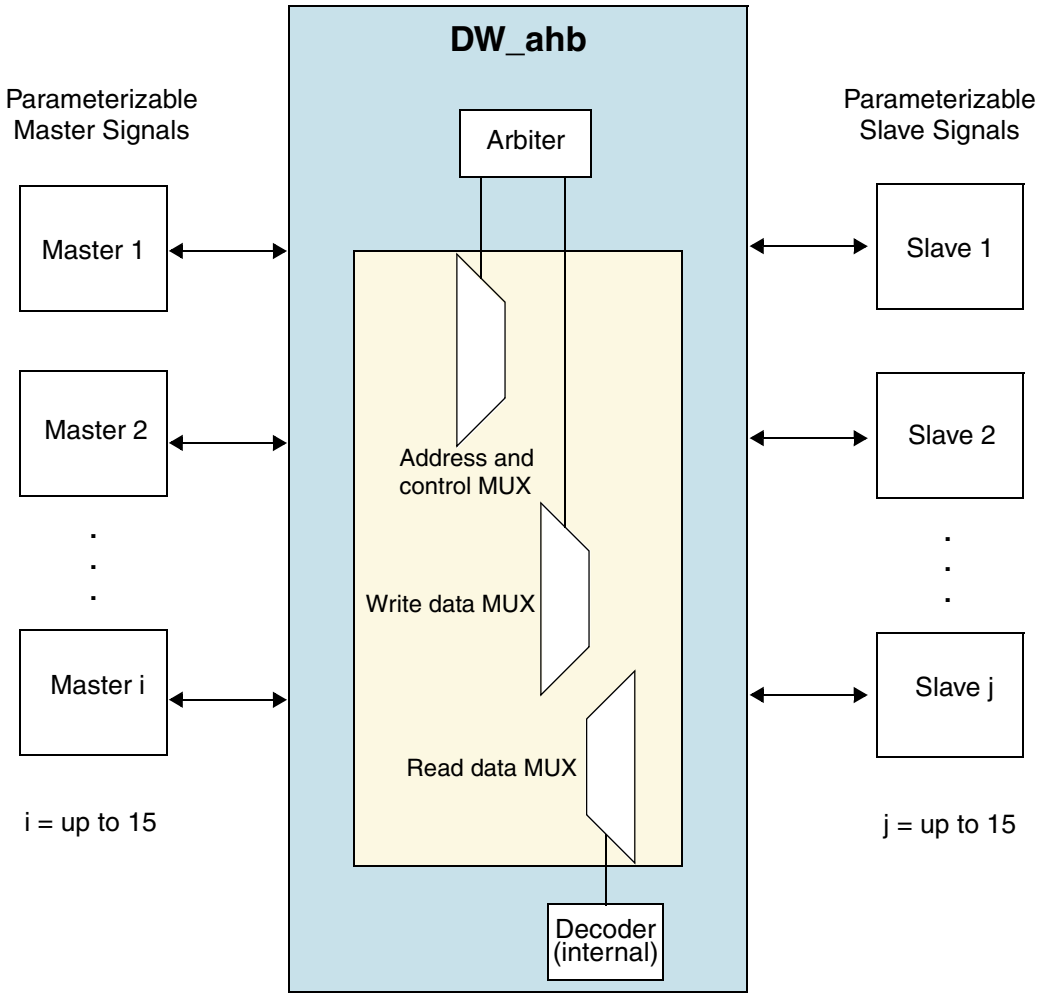
The Synopsys DW_ahb conforms to the *AMBA Specification, Revision 2.0* from ARM.

1.2.1 DW_ahb Block Diagram

As shown in Figure 1-2 on page 16, the DW_ahb consists of the following components:

- "Arbiter" on page 31 – 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 will occur:
  - Weighted-token arbitration will be excluded
  - Early burst termination will be disabled
  - Priorities will be fixed and hardcoded
- "Optional Internal Decoder" on page 40 – 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" on page 43 – 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" on page 43 – 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 15 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
- 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

Source code for this component is available on a per-project basis as a DesignWare Core. Please 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 97 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, refer to "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, refer to "Overview of the coreConsultant Configuration and Integration Process" on page 20.

For more information about implementing your DW_ahb component within a DesignWare subsystem using coreAssembler, refer to "Overview of the coreAssembler Configuration and Integration Process" on page 22.

2 Building and Verifying a Component or Subsystem

DesignWare Synthesizable IP (SIP) components for AMBA 2 and AMBA 3 AXI are packaged using Synopsys coreTools, which enable the user to configure, synthesize, and run simulations on a single SIP title, or to build a configured AMBA subsystem. You do this by generating a workspace view using one of the following coreTools applications:

- coreConsultant – Used for configuration, RTL generation, synthesis, and execution of packaged verification for a single SIP title. The *coreConsultant User Guide* provides complete information on using coreConsultant.
- coreAssembler – Used for building and configuration of a subsystem that connects multiple SIP titles, RTL generation, synthesis, and creation of a template subsystem testbench. The *coreAssembler User Guide* provides complete information on using coreAssembler.

A workspace is your working version of a DesignWare SIP component or subsystem. In fact, you can create several workspaces to experiment with different design alternatives.

Hint If you are unfamiliar with coreTools—which is comprised of the coreAssembler, coreConsultant, and coreBuilder tools—you can go to *Using DesignWare Library IP in coreAssembler* to "get started" learning how to work with DesignWare SIP components.

2.1 Setting up Your Environment

The DW_ahb is included in a release of DesignWare SIP components. It is assumed that you have already downloaded and installed the release. If you have not, you can download and install the latest versions of required tools using the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

You also need to set up your environment correctly using specific environment variables, such as DESIGNWARE_HOME, VERA_HOME, PATH, and SYNOPSYS. If you are not familiar with these requirements and the necessary licenses, refer to "Setting up Your Environment" in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

2.2 Overview of the coreConsultant Configuration and Integration Process

Once you have correctly downloaded and installed a release of DesignWare SIP components and then set up your environment, you can begin work on the DW_ahb using coreConsultant.

2.2.1 coreConsultant Usage

Figure 2-1 illustrates some general directories and files in a coreConsultant workspace.

Figure 2-1 coreConsultant Usage Flow

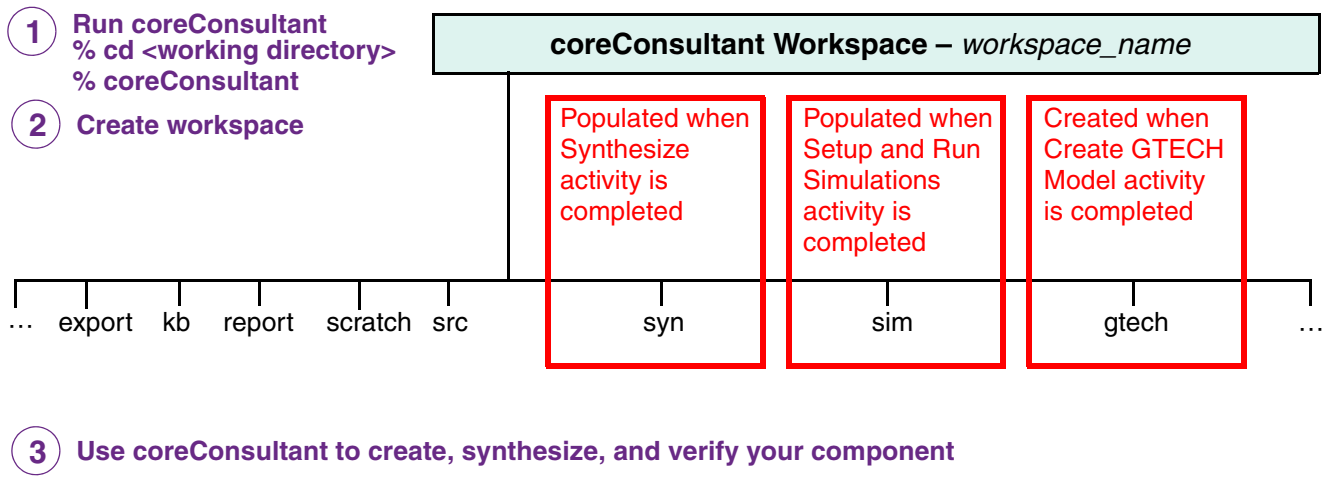


Table 2-1 provides a description of the implementation workspace directory and subdirectories.

Table 2-1 coreConsultant Implementation Workspace Directory Contents

Directory/Subdirectory	Description
auxiliary	Scripts and text files used by coreConsultant. Generated upon first creating workspace.
doc	Contains local copies of component-specific databooks. Generated upon first creating workspace.
export	Contains files used to integrate results from the completed source configuration and synthesis activities into your design (outside coreConsultant). Generated upon first creating workspace; populated during Specify Configuration activity.
gtech	Contains synthesis scripts and output netlists from gtech generation; also used for RTL simulation of encrypted source code. Generated during Generate GTECH Model activity.
kb	Contains knowledge base information used by coreConsultant. These are binary files containing the state of the design. Generated upon first creating workspace; populated and updated throughout activities.

Table 2-1 coreConsultant Implementation Workspace Directory Contents (Continued)

Directory/Subdirectory	Description
leda	Contains Leda configuration files for the component. Generated upon first creating workspace; updated during Run Leda Coding Checker activity.
pkg	Contains RTL preprocessor scripts. Generated during Specify Configuration activity.
report	Contains all of the reports created by coreConsultant during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated upon first creating workspace; populated and updated throughout activities.
scratch	Contains temp files used during the coreConsultant processes. Generated upon first creating workspace; populated and updated throughout activities.
sim	Contains test stimulus and output files. Generated upon first creating workspace; updated during Setup and Run Simulations activity.
src	Includes the top-level RTL file, *design_name*.v. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated upon first creating workspace; populated during Specify Configuration activity.
syn	Contains synthesis files for the component. Generated upon first creating workspace; updated during Synthesis activity and Formal Verification activity.
tcl	Contains synthesis intent scripts. Generated upon first creating workspace.

For details on some key files created during coreConsultant activities, refer to "Database Files" on page 27.

For information on using coreConsultant, refer to the *coreConsultant User Guide*.

2.2.2 Configuring the DW_ahb within coreConsultant

The "Parameters" chapter on page 47 describes the DW_ahb hardware configuration parameters that you configure using the coreConsultant GUI.

The "Creating the RTL View of a Core" chapter in the *coreConsultant User Guide* discusses how to specify a configuration for an individual component like the DW_ahb.

2.2.3 Creating Gate-Level Netlists within coreConsultant

The "Creating the Gate-Level Netlist for a Core" chapter in the *coreConsultant User Guide* discusses how to create a translation of the RTL view into a technology-specific netlist for an individual component like the DW_ahb.

2.2.4 Verifying the DW_ahb within coreConsultant

The "Verification" chapter on page 97 provides an overview of the testbench available for DW_ahb verification using the coreConsultant GUI.

The "Verifying Your Implementation" chapter in the *coreConsultant User Guide* discusses how to simulate an individual component like the DW_ahb.

2.2.5 Running Leda on Generated Code with coreConsultant

When you select **Verify Component > Run Leda Coding Checker** from the Activity List, the corresponding Activity View appears. In this Activity View you select rules configuration file and define Leda command line switches.

2.3 Overview of the coreAssembler Configuration and Integration Process

Once you have correctly downloaded and installed a release of DesignWare SIP components and then set up your environment, you can begin work on your DesignWare subsystem with coreAssembler.

2.3.1 coreAssembler Usage

Figure 2-2 illustrates some general directories and files in a coreAssembler workspace.

Figure 2-2 coreAssembler Usage Flow

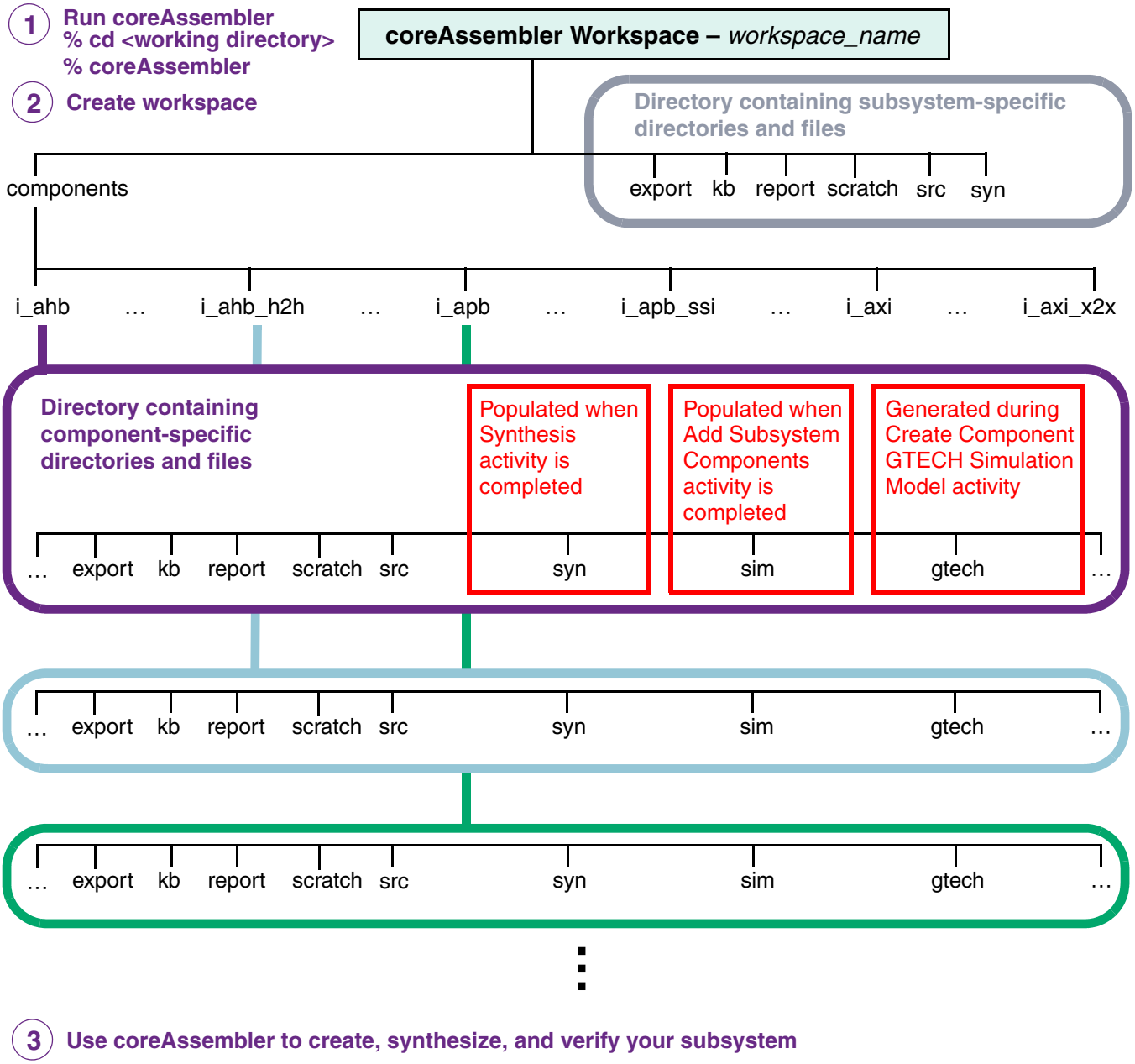


Table 2-2 provides a description of the implementation workspace directory and subdirectories.

Table 2-2 coreAssembler Implementation Workspace Directory Contents

Directory/Subdirectory	Description
components	Contains a directory for each IP component instance connected in the subsystem. Generated and populated with separate component directories upon first adding components; populated and updated throughout activities.
i_*component*/auxiliary	Scripts and text files used by coreAssembler. Generated during Add Subsystem Components activity.
i_*component*/doc	Contains local copies of component-specific databooks. Generated during Add Subsystem Components activity.
i_*component*/export	Contains files used to integrate results from the completed source configuration and synthesis activities into your design (outside coreAssembler). Generated during Add Subsystem Components activity; populated during Configure Components activity.
i_*component*/gtech	Contains synthesis scripts and output netlists from gtech generation; also used for RTL simulation of encrypted source code. Generated during Create Component GTECH Simulation Model activity.
i_*component*/kb	Contains knowledge base information used by coreAssembler. These are binary files containing the state of the design. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/leda	Contains Leda configuration files for the component. Generated during Add Subsystem Components activity; populated during Run Leda Coding Checker (for /i_*component*) activity.
i_*component*/pkg	Contains RTL preprocessor scripts. Generated during Configure Components activity.
i_*component*/report	Contains all of the reports created by coreAssembler during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/scratch	Contains temp files used during the coreAssembler processes. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/sim	Contains test stimulus and output files. Generated during Add Subsystem Components activity; updated during Setup and Run Simulations (for /i_*component*) activity.

Table 2-2 coreAssembler Implementation Workspace Directory Contents (Continued)

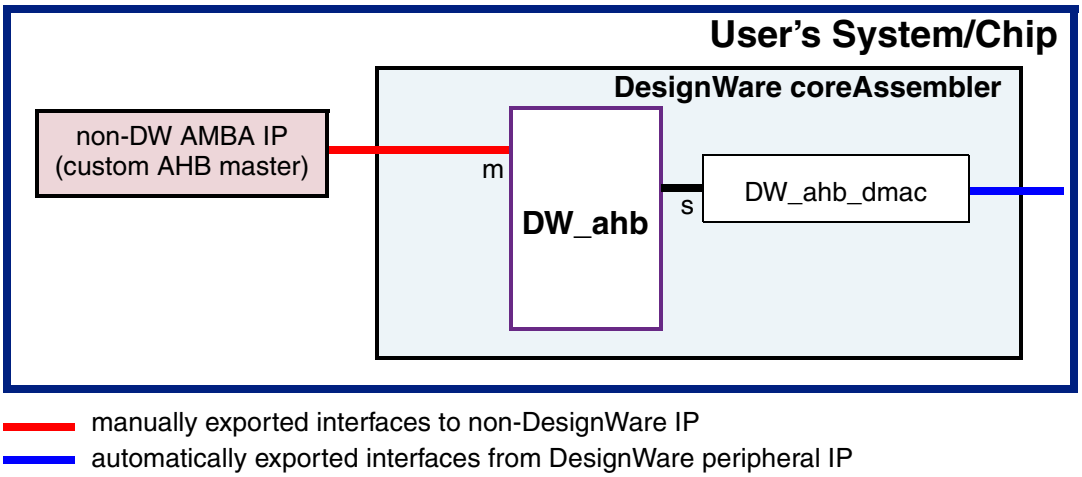
Directory/Subdirectory	Description
i_*component*/src	Includes the top-level RTL file, *design_name*.v. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated during Add Subsystem Components activity; populated during Specify Configuration activity.
i_*component*/syn	Contains synthesis files for the component. Generated during Add Subsystem Components activity; updated during Synthesis activity.
i_*component*/tcl	Contains synthesis intent scripts. Generated during Add Subsystem Components activity.
export	Contains subsystem files used to integrate the results from the completed source configuration and synthesis activities into your design (outside coreAssembler). Generated upon first creating workspace; populated starting with Memory Map Specification activity.
kb	Contains subsystem knowledge base information used by coreAssembler. These are binary files containing the state of the design. Generated upon first creating workspace; populated and updated throughout activities.
report	Contains subsystem reports created by coreAssembler during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated upon first creating workspace; populated and updated throughout activities.
scratch	Contains subsystem temp files used during the coreAssembler processes. Generated upon first creating workspace; populated and updated throughout activities.
src	Includes the RTL related to the subsystem. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated upon first creating workspace; populated starting with Generate Subsystem RTL activity.
syn	Contains synthesis files for the subsystem. Generated upon first creating workspace; updated during Synthesize activity and Formal Verification activity.

For details on some key files created during coreAssembler activities, refer to "Database Files" on page 27.

For information on using coreAssembler, refer to the *coreAssembler User Guide*. For information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI components within coreTools, refer to *Using DesignWare Library IP in coreAssembler*.

Figure 2-3 illustrates the DW_ahb in a simple subsystem.

Figure 2-3 DW_ahb in Simple Subsystem



The subsystem in Figure 2-3 contains the following components that you may want to use as you learn to use coreAssembler:

- DW_ahb
- DW_ahb_dmac
- AHB Master

The AHB Master is meant to be exported out of the design and then replaced by a real AHB Master—such as a CPU—later in the design process; at least one exported AHB master is required in a subsystem if you intend to do a basic simulation that tests connections.

2.3.2 Configuring the DW_ahb within a Subsystem

The "Parameters" chapter on page 47 describes the DW_ahb hardware configuration parameters that you configure using the coreAssembler GUI. Corresponding databooks for the other components in a subsystem contain "Parameters" chapters that describe their respective configuration parameters.

The "Creating the RTL View of a Subsystem" chapter in the *coreAssembler User Guide* discusses how to configure subsystem components and automatically connect them using the coreAssembler GUI.

2.3.3 Creating Gate-Level Netlists within coreAssembler

The "Creating the Gate-Level Netlist for a Subsystem" chapter in the *coreAssembler User Guide* discusses how to create a translation of the RTL view into a technology-specific netlist for a subsystem.

2.3.4 Verifying the DW_ahb within coreAssembler

The "Verification" chapter on page 97 provides an overview of the testbench available for DW_ahb verification using the coreAssembler GUI.

The "Verifying Subsystems and Components" chapter in the *coreAssembler User Guide* discusses how to simulate a subsystem.

2.3.5 Running Leda on Generated Code with coreAssembler

When you select **Verify Component > Run Leda Coding Checker for /i_*component*)** from the Activity List, the corresponding Activity View appears. In this Activity View you select rules configuration file and define Leda command line switches.

2.4 Database Files

The following subsections describe some key files created in coreConsultant and coreAssembler activities.

2.4.1 Design/HDL Files

The following sections describe the design and HDL files that are produced by coreConsultant and coreAssembler when configuring and verifying a DesignWare Synthesizable Component. The following files are created in different directories by coreConsultant and coreAssembler:

- coreConsultant – *workspace*/ directory
- coreAssembler – *workspace*/components/i_*component*/ directory

2.4.1.1 RTL-Level Files

The following table describes the RTL files that are generated by the Create RTL activity. They are encrypted except where otherwise noted. Any Synopsys synthesis tool or simulator can read encrypted RTL files.

Table 2-3 RTL-Level Files

Files	Encrypted?	Purpose
./src/*component*_cc_constants.v	No	Includes definitions and values of all configuration parameters that you have specified for the component.

./src/*component*.v	No	Top-level HDL file. Include the DesignWare libraries by using the following options in your simulator invocation:
		`+libext+.v+.V`
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`

./src/*component_submodule*.v	Yes	Sub-modules of component
./src/*component*_constants.v	No	Includes the constants used internally in the design.
./src/*component*_undef.v		Includes an undef for each of the definitions found in the *component*_cc_constants.v file; compiled in after the last file listed in ./src/*components*.lst when compiling multiple instances of the same IP.
./src/*component*.lst	No	Lists the order in which the RTL files should be read into tools, such as simulators or dc_shell. For example, use the following option to read the design into VCS: `vcs +v2k -f component.lst`

2.4.1.2 Simulation Model Files

The following table includes files generated for the component during the Generate GTECH Simulation activity. These files are needed when you are using a non-Synopsys simulator (when you can not use the encrypted RTL).

Table 2-4 Simulation Model Files

Files	Encrypted?	Purpose

./gtech/final/db/*component*.v	No	Simulation model of the component for use with non-Synopsys simulators. A technology-independent, gate-level netlist; VHDL and Verilog versions are generated. Include the DesignWare libraries by using the following options in your simulator invocation:
		`+libext+.v+.V`
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`

2.4.2 Synthesis Files

The following table includes files generated after the Create Gate-Level Netlist activity is performed on a component.

Table 2-5 Synthesis Files

Files	Encrypted?	Purpose
./syn/auxScripts	No	Auxiliary files for synthesis.
./syn/final/db/*component*.db	Binary format	Synopsys .db files (gate level) that can be read into dc_shell for further synthesis, if desired.
./syn/final/db/*component*.v	No	Gate-level netlist that is mapped to technology libraries that you specify.
./syn/constrain/script/*.*	No	Constraint files for the components.
./syn/final/report/*.*	No	Synthesis result files.

2.4.3 Verification Reference Files

Files described in the following table include information pertaining to the component's operation so that you can verify installation and configuration of the component has been successful. These files are not for re-use during system-level verification.

Table 2-6 Verification Reference Files

Files	Encrypted?	Purpose
./sim/runtest	No	Perl script that runs the Setup and Run Simulations activity from the command line.
./sim/runtest.log	No	The overall result of simulation, including pass/fail results.
./sim/test_*testname*/test.result	No	Pass/fail of individual test.

Table 2-6 Verification Reference Files

Files	Encrypted?	Purpose
./sim/test_*testname*/test.log	No	Log file for individual test.

3 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-Lite/AMBA 3 AHB-Lite The DesignWare AHB Lite configuration does not include the following:

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

3.1 DW_ahb Components

This section describes the functions of the following DW_ahb components:

- Arbiter
- "Optional Internal Decoder" on page 40
- "Optional External Decoder" on page 43
- "Multiplexer" on page 43

3.1.1 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.

3.1.1.1 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.

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, refer to "Master Priority Level Registers" on page 93.

- **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.

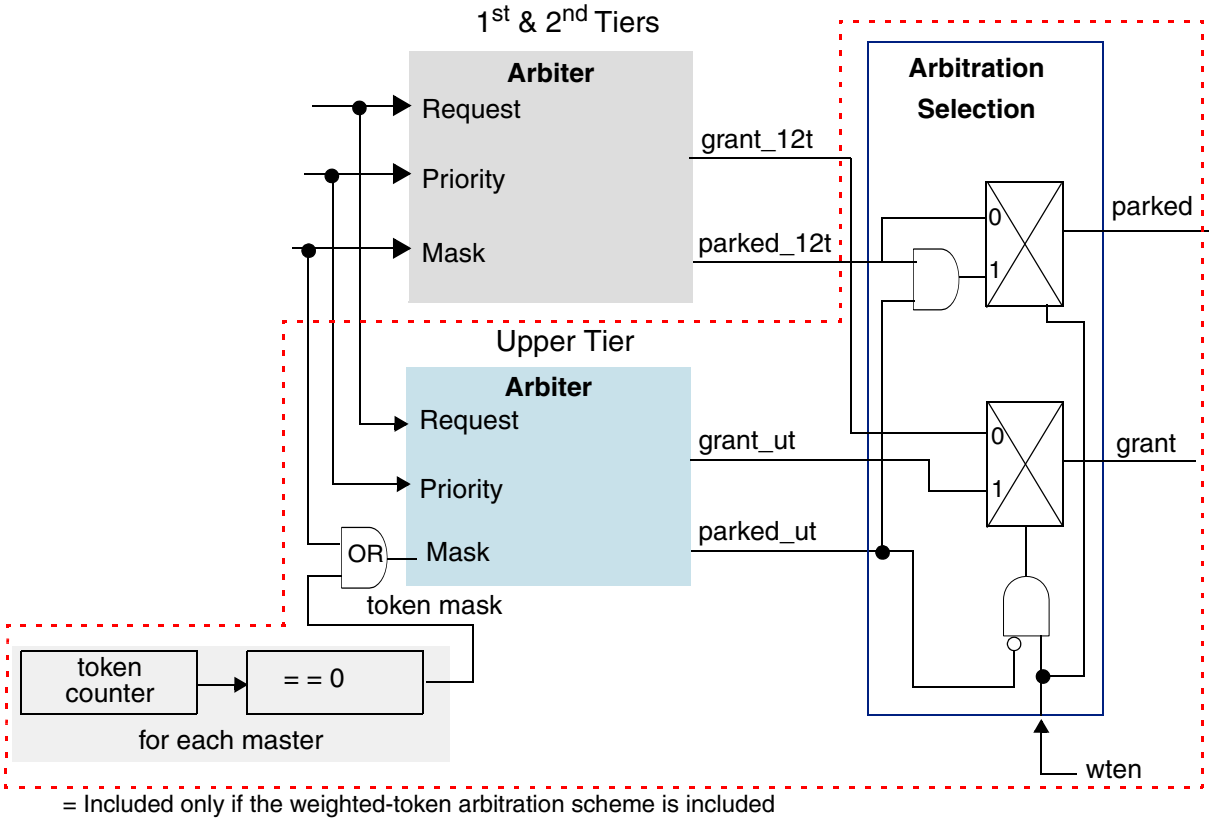
User-configured token values are summed—by adding 16 to the priority of a master when it has tokens left—to ensure that they do not exceed the total allocated number of tokens.

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. 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. For more information, refer to the configuration parameters for weighted-token arbitration in Table 4-11 on page 66.

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 3-1.

Figure 3-1 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 3-2 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 3-2 Bursts with Specified Length

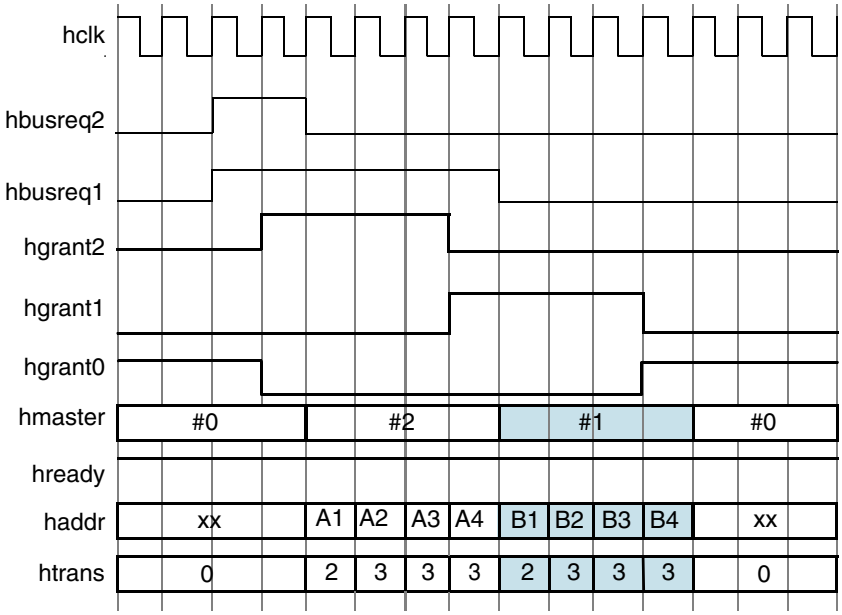
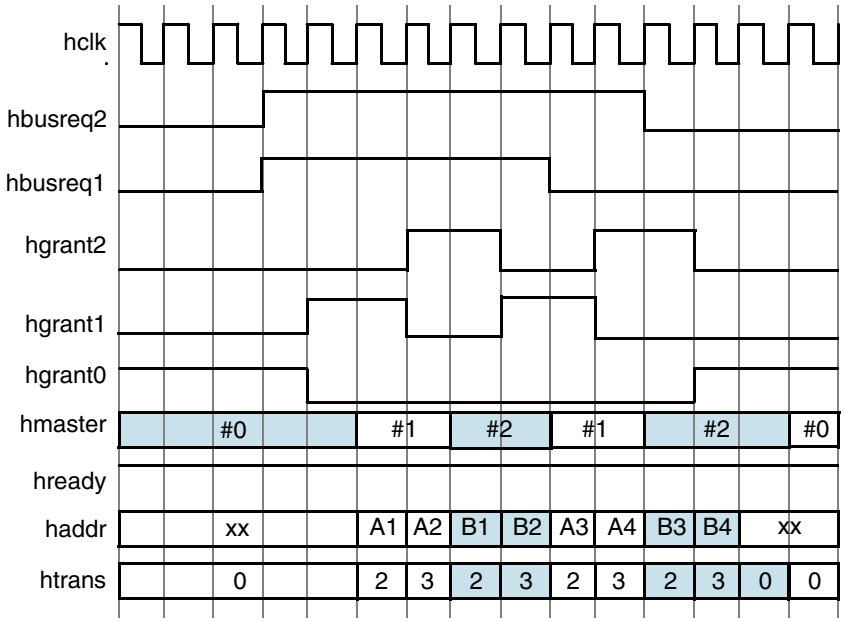


Figure 3-3 shows the timing that occurs when masters with undefined length bursts and the same priority level are early terminated by masters of equal or higher priority.

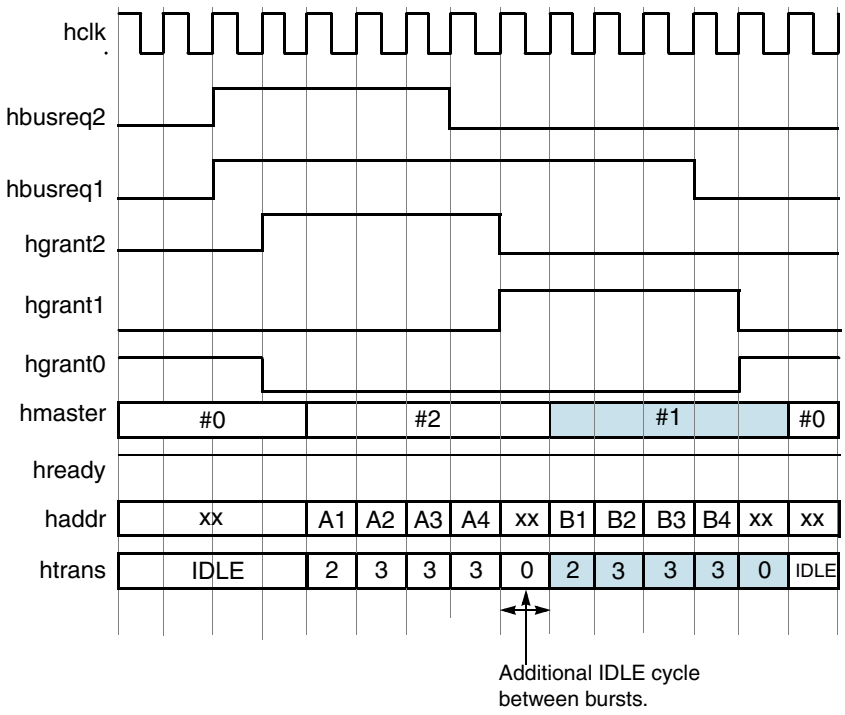
Figure 3-3 Bursts with Unspecified Length and Equal Priority Level



In Figure 3-3, 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 3-4 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 3-4 Bursts with Unspecified Length and Different Priority Level



3.1.1.2 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 will begin arbitration again to grant access to the bus, but masters with a lower priority than the current master will be 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.

3.1.1.3 Arbiter Slave Interface

Optional Feature. 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, refer to "Arbiter Slave Interface Registers" on page 93.

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.

3.1.1.4 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.

3.1.1.5 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 94. If there is no arbiter slave interface, the ID number of default master is hardcoded.

3.1.1.6 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 will include 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.

3.1.1.7 Delayed Pause Action

Optional Feature. When the delayed pause action is set during configuration, the pause signal, when set, will 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.

3.1.1.8 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.

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, refer to "Early Burst Termination Registers" on page 94.

3.1.1.9 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 will 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.

3.1.1.10 Weighted-Token Arbitration

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

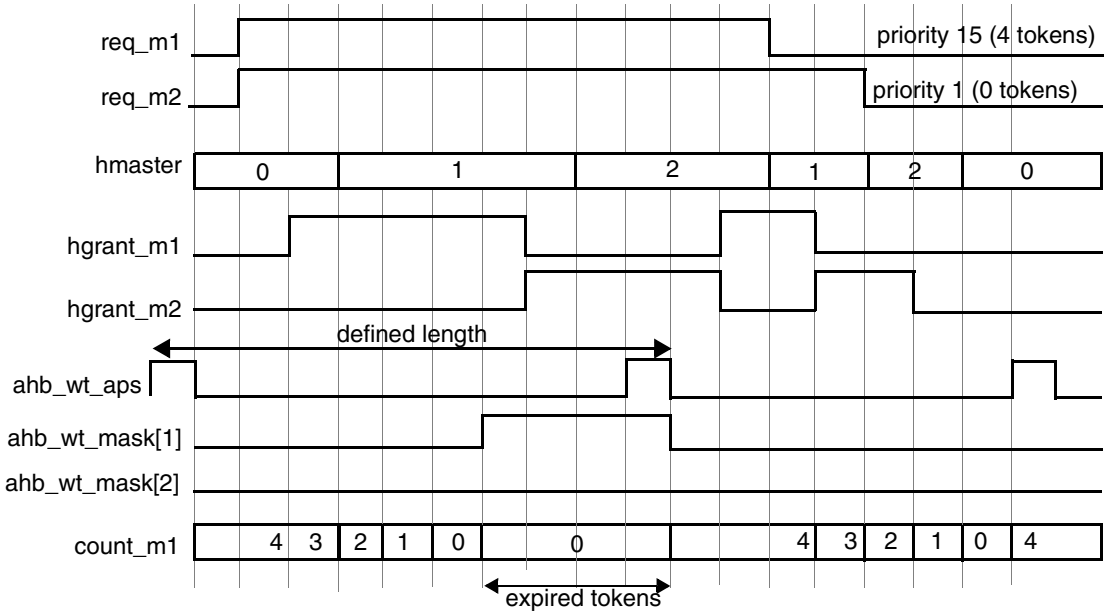
[(maximum number of cycles) – two cycles]

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 and the bus hands it over on the next cycle (100 cycles).

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 will operate 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 3-5 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, refer to "Weighted-Token Arbitration Registers" on page 95).

Figure 3-5 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 will always operate at the upper tier and will never run 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.

3.1.2 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 43). 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 42
- "Slave Aliasing" on page 42
- "Slave Visibility" on page 42
- "Default Slave" on page 43

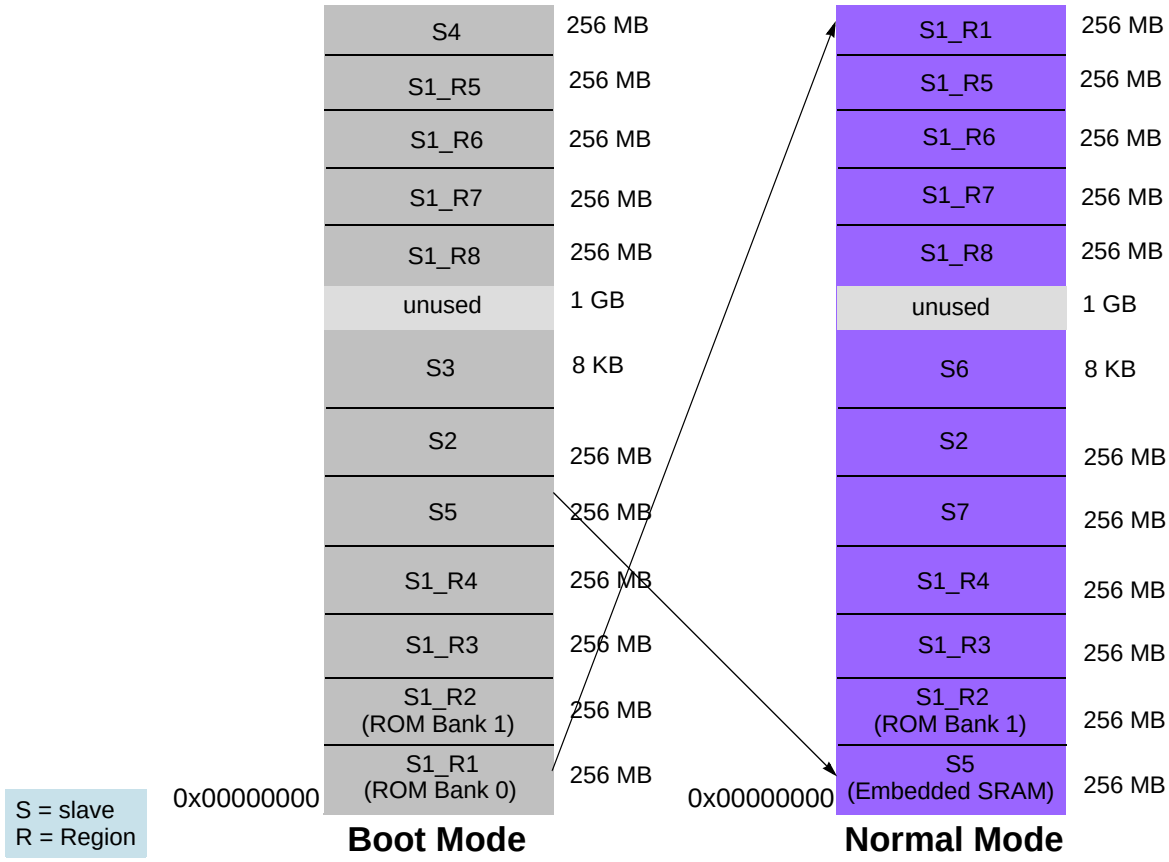
3.1.2.1 Remap Operation

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 3-6 illustrates memory maps for both Boot and Normal modes.

Figure 3-6 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 will be 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).

3.1.2.2 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.

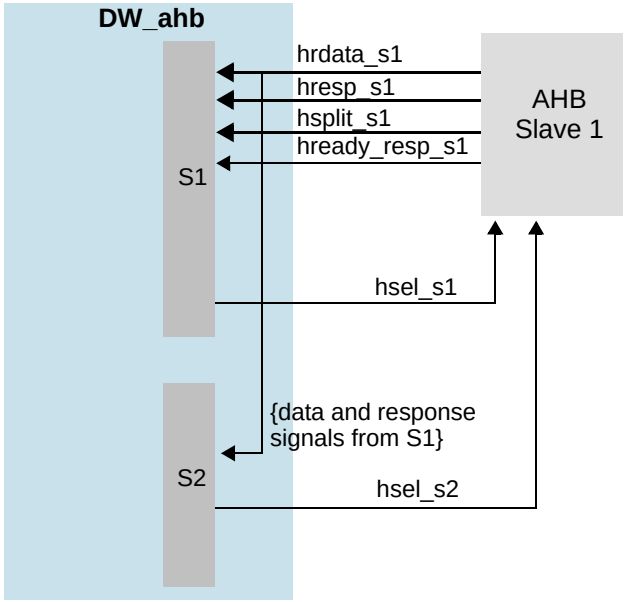
A slave does not have to occupy a continuous area of the memory map. DW_ahb is designed so that slave 1 can have up to eight regions (refer to Figure 3-6) in both Boot and Normal modes. It does not need the same number of regions in both modes. This allows a memory controller, for instance, to have banks at non-contiguous memory locations. For instance, in Figure 3-6 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.

3.1.2.3 Slave Aliasing

The DW_ahb also includes an alias feature, shown in Figure 3-7 on page 42, 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 3-7. 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 3-7 DW_ahb with Aliased Slave



3.1.2.4 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 3-6 on page 41, 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.

3.1.2.5 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.

3.1.3 Optional External Decoder

During configuration of the DW_ahb, users can choose to have an external decoder. By having the decoder external to the DW_ahb, users 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.

3.1.4 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 was addressed in the previous cycle.

3.2 Timing Diagrams

For timing, refer to the following diagrams:

- Bursts with specified length (Figure 3-2 on page 35)
- Bursts with unspecified length and equal priority levels (Figure 3-3 on page 35)
- Bursts with unspecified length and different priority levels (Figure 3-4 on page 36)

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

- Simple transfer (Figure 3-8 on page 44)
- Transfer with wait states (Figure 3-9 on page 44)
- Multiple transfers (Figure 3-10 on page 45)
- Granting access with no wait states (Figure 3-11 on page 45)
- Granting access with wait states (Figure 3-12 on page 45)
- Data bus ownership (Figure 3-13 on page 46)
- Hand-over after burst (Figure 3-14 on page 46)

Figure 3-8 Simple Transfer

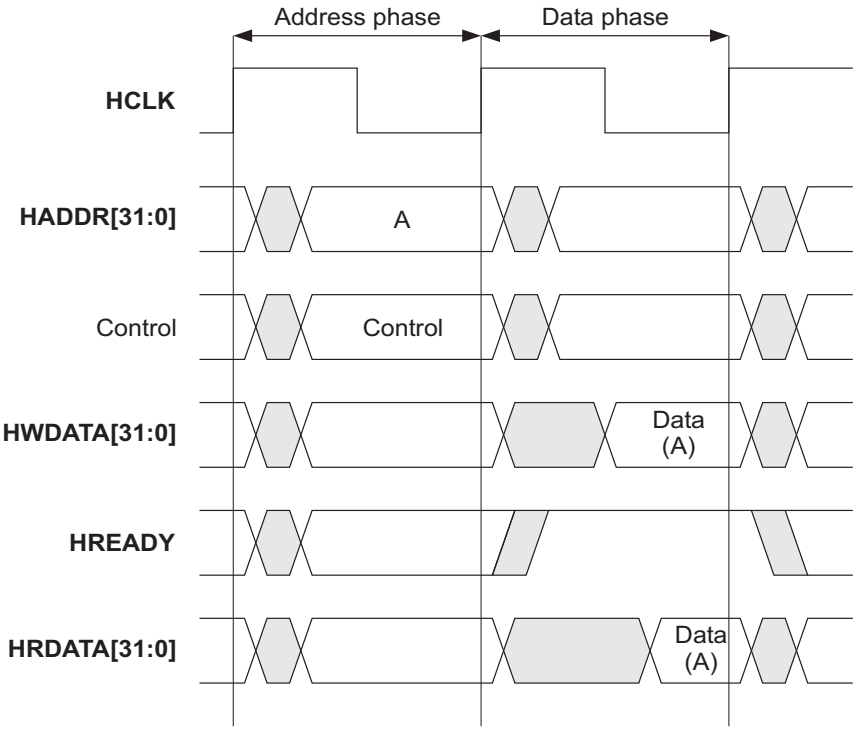


Figure 3-9 Transfer with Wait States

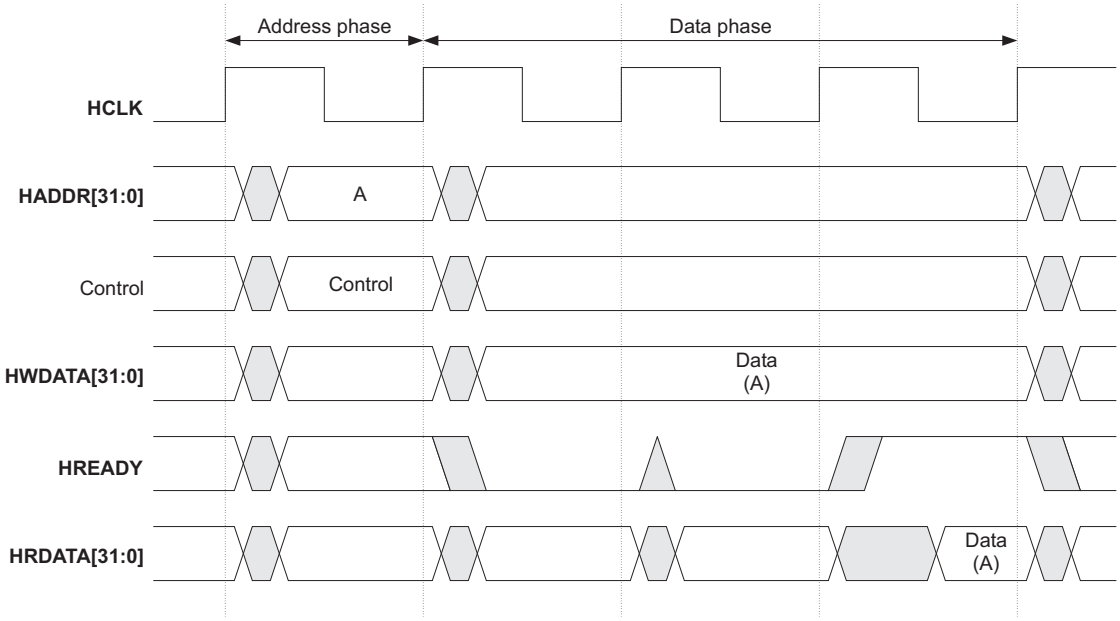


Figure 3-10 Multiple Transfers

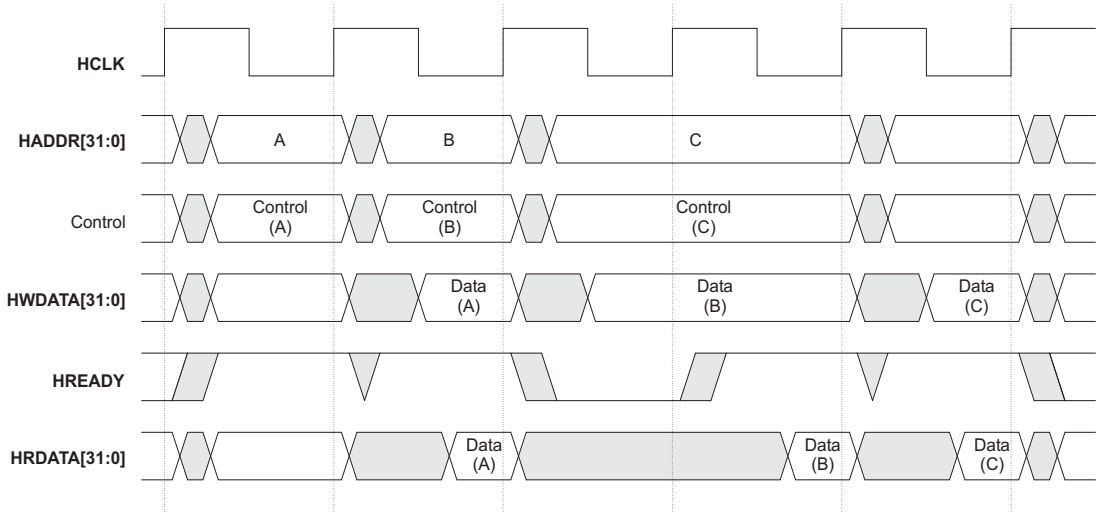


Figure 3-11 Granting Access with No Wait States

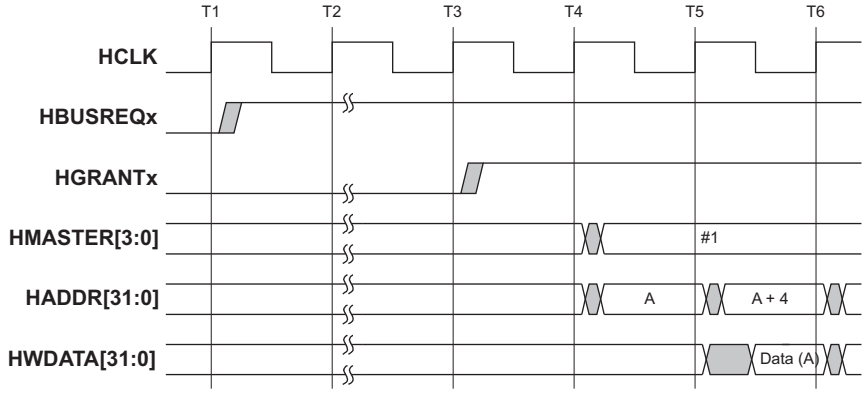


Figure 3-12 Granting Access with Wait States

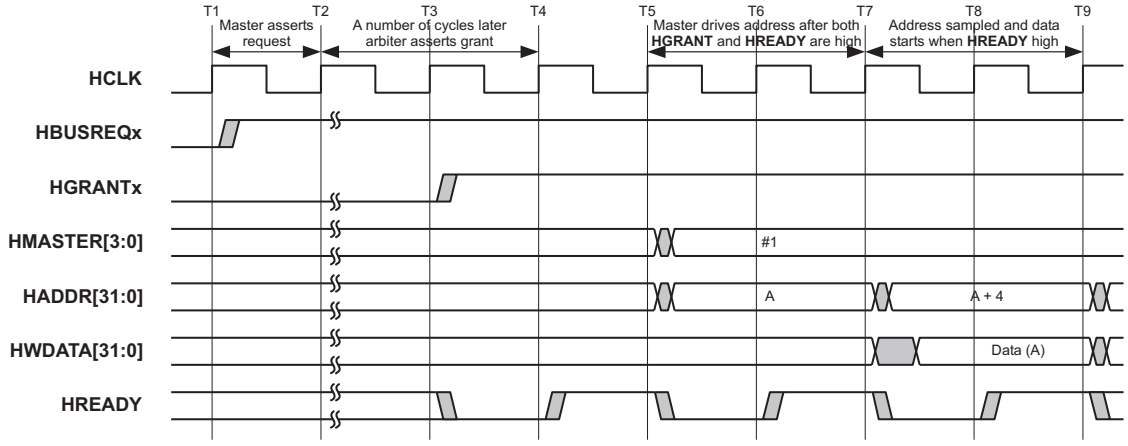


Figure 3-13 Data Bus Ownership

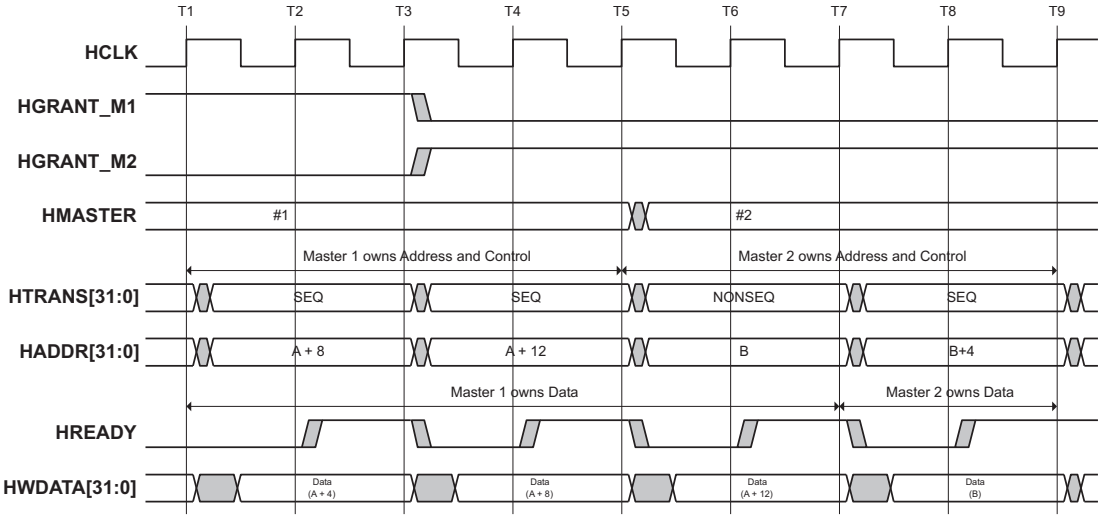
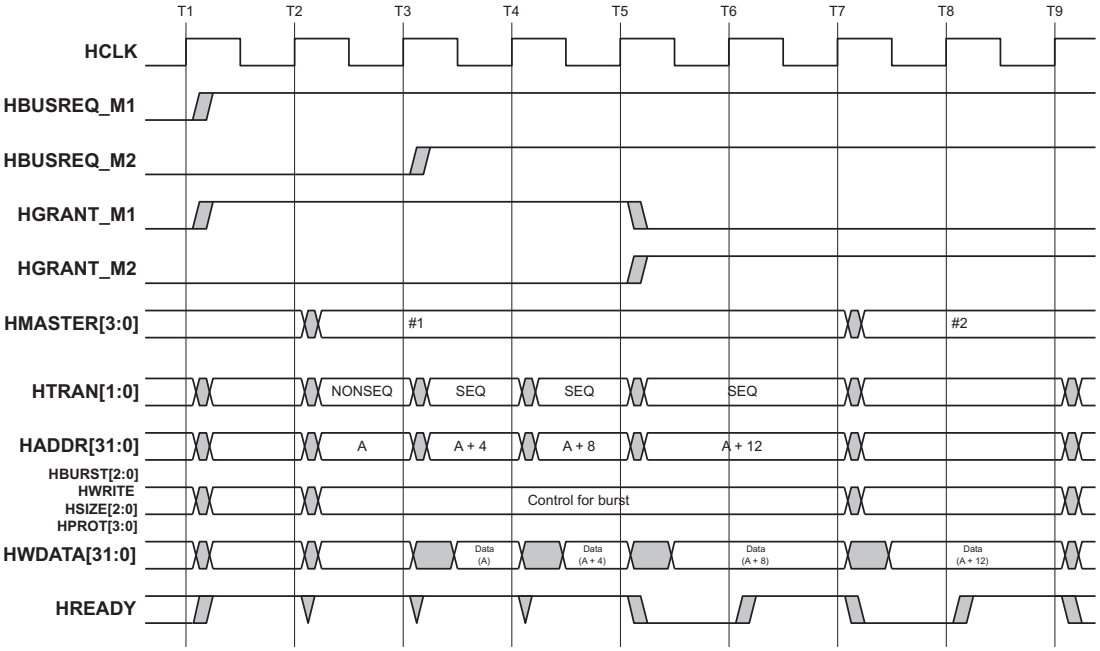


Figure 3-14 Handover after Burst



4 Parameters

This chapter describes the configuration parameters used by the DW_ahb. The settings of the configuration parameters determine the I/O signal list of the DW_ahb peripheral.

4.1 Parameter Descriptions

You use coreConsultant or coreAssembler to configure the following parameters and generate the configured code.

Attention	When using coreConsultant or coreAssembler, you can right-click on a parameter label to access a “What’s This” popup dialog that will tell you the details for that particular parameter. The information in each What’s This dialog essentially matches the information in the parameter descriptions below.

4.1.1 DW_ahb Top-Level Parameters

Table 4-1 lists the DW_ahb top-level parameter descriptions.

Table 4-1 DW_ahb Top-Level Parameters

Label	Parameter Definition
AMBA Lite?	**Parameter Name:** AHB_LITE **Legal Values:** True/False **Default Value:** False **Dependencies:** None. **Description:** If set to True (1), the system is configured with only one master that never has to request ownership of the bus because it is always granted the bus. No dummy master is required, since the slaves are not split capable. A master will drive IDLE cycles when it does not want the bus. This is also true for the following: ■ Pause mode is not enabled. ■ Default master number is changed to 1. ■ Number of masters is changed to 1. ■ Arbiter interface is removed. ■ All slaves are made non split capable.
Number of AHB Master Ports	**Parameter Name:** NUM_AHB_MASTERS **Legal Values:** 1 to 15 **Default Value:** 2 **Dependencies:** When AHB Lite is configured, then this is set to 1. **Description**: The number of AHB masters contained in the system.
AHB System Address Width	**Parameter Name:** HADDR_WIDTH **Legal Values:** 32 to 64 bits **Default Value:** 32 **Dependencies:** None. **Description**: Chooses the address width for the AHB address bus.
AHB Data Bus Width	**Parameter Name:** AHB_DATA_WIDTH **Legal Values:** 8, 16, 32, 64, 128, or 256 bits **Default Value:** 32 **Dependencies:** None. **Description**: Selects the width of the AHB data bus. The maximum 256-bit width is an arbitrary limitation enforced by coreConsultant.

Table 4-1 DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
External endian control?	**Parameter Name:** AHB_XENDIAN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: 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_bin_endian is not included on the interface, and the endian type is set at configuration by the BIG_ENDIAN configuration parameter.
System Endianness	**Parameter Name:** BIG_ENDIAN **Legal Values:** Little-Endian (0) or Big-Endian (1) **Default Value:** Little-Endian (0) **Description**: By default, the DW_ahb is configured as a little-endian system. You can choose the endianness of the system. 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.
External Decoder?	**Parameter Name:** AHB_HAS_XDCDR **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: This parameter allows the decoder to be implemented within the parallel multi-layer matrix. If True (1), the decoder is external to the DW_ahb. If False (0), decoder is internal to the DW_ahb. For an internal decoder, the addresses need to be supplied by the DW_ahb at configuration. Overlaps and inconsistencies are checked. An external decoder allows users to connect any decoder with any number of remap options.
Support AMBA Memory Remap Feature?	**Parameter Name:** REMAP **Legal Values:** True (1) or False (0) **Default Value:** False (0 - Normal Mode) **Dependencies:** This option is relevant only when there is an internal decoder (AHB_HAS_XDCDR = 0). **Description**: Allows the memory map to be swapped. When set, the system supports the DesignWare Memory Remap functionality. Remap allows one set of addresses for boot, and another for normal operation.This setting must be set if two addressing modes are required. For more information, see “Remap Operation” on page 40.

Table 4-1 DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
Support Arbiter Pause Mode?	**Parameter Name:** PAUSE **Legal Values:** True (1) or False (0) **Default Value:** True (1) Dependencies: Not available in an AHB Lite system (AHB_LITE = 1). **Description**: If set to True (1), the system supports the arbiter pause mode. This setting allows granting of the bus to the Dummy master when the system enters low power mode. When in AHB_LITE, pause mode is disabled.
Support Delayed Pause Action?	**Parameter Name:** AHB_DELAYED_PAUSE **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** Not available when PAUSE = False (0). **Description**: When the delayed pause action is supported, the pause signal, when set, will not take effect until hready is high and htrans is IDLE. If this action is not set (False), 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.
Include Arbiter Interface?	**Parameter Name:** AHB_HAS_ARBIF **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** Available if the configured AHB is not an AHB Lite AHB. **Description:** 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, meaning it is unused. There is no arbiter interface when the system is an AHB Lite system.
Include Weighted Token Arbitration?	**Parameter Name:** AHB_WTEN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** The configured AHB is not an AHB Lite AHB, and you are including the internal arbiter interface. **Description:** 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.

Table 4-1 DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
Include Weighted Token Outputs?	**Parameter Name:** AHB_WTEN_DEBUG **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** Not available when AHB_WTEN = False (0). **Description:** Enables the inclusion of weighted token clock token counter outputs as top-level outputs that can 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.
Support full incrementing bursts?	**Parameter Name:** AHB_FULL_INCR **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None **Description:** 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.
Total Number of Slave Select Lines in the system	**Parameter Name:** NUM_IAHB_SLAVES **Legal Values:** 1 to 15 **Default Value:** 4 **Dependencies:** None **Description**: 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 slaves assigned. If there is only one addressing mode, then this is the number of slaves in the system.
Number of AHB Slave Ports in Normal Mode	**Parameter Name:** NUM_NAHB_SLAVES **Legal Values:** 1 to 15 **Default Value:** 4 **Dependencies:** This parameter option is active only if you enable the Memory Remap Feature. **Description**: 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.

Table 4-1 DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
Number of AHB Slave Ports in Boot Mode	**Parameter Name:** NUM_BAHB_SLAVES **Legal Values:** 1 to 15 **Default Value:** 0 **Dependencies:** This parameter option is active only if you enable the Memory Remap Feature. **Description**: 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.
External Decoder Provides HSELS back to DW_ahb for routing to Slaves?	**Parameter Name:** XDCDR_SUPPLIES_HSELS2AHB **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** This parameter option is active only if you enable the External Decoder feature. **Description**: If set to True (1), the decoder supplies hsel lines to the DW_ahb for re-routing. If set to False (0), the decoder routs the hsel lines to the slaves.

4.1.2 AHB Source Code Configuration

Table 4-2 lists the configuration parameter for DW_ahb source code.

Table 4-2 AHB Source Code Configuration

Label	Parameter Definition
Use DesignWare Foundation Synthesis Library	**Parameter Name:** USE_FOUNDATION **Legal Values:** True (1) or False (0) **Default Value:** True if DesignWare License is available; False if *no* DesignWare License is available **Dependencies:** Parameter is enabled if customer has both Source and DesignWare licenses Description: The component code utilizes DesignWare Foundation parts for optimal Synthesis QoR. Customers with only a DesignWare license *must* use Foundation parts. Customers with only a Source license *cannot* use Foundation parts. Customers with both Source and DesignWare licenses have the option of using DesignWare Foundation parts.

4.1.3 Slave Memory Region Definition

Table 4-3 describes the memory region definition parameters for the first DW_ahb slave (slave 1), which can have up to eight address regions in Boot, Normal, or both modes, whereas slaves 2 through 15 can have only up to two address regions.

Table 4-3 Slave 1 Memory Region Definition

Label	Parameter Definition
Number of slave which returns data and response	**Parameter Name:** ALIAS_S1 **Legal Values:** 1 to NUM_IAHB_SLAVES **Default Value:** 1 **Dependencies:** This parameter option is available only if the HSEL_ONLY_S1 is set to Yes. **Description:** 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 *i*, meaning you cannot alias this slave to itself.
Alias this system slave to another system slave?	**Parameter Name:** HSEL_ONLY_S1 **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description:** Generates only hsel for this slave; requires an aliases for data and response from another slave. For more information, see “Slave Aliasing” on page 42.
Number of Memory Regions in Boot Mode?	**Parameter Name:** MR_B1 **Legal 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 (0) **Dependencies:** This parameter option is available only for slave 1 and if the Slave Visibility Mode is set to “Boot” or “Normal & Boot”. Additionally, this option applies only if you have an internal decoder (AHB_HAS_XDCDR = 0). **Description:** The number of memory regions in Boot Mode for slave 1. For more information, see “Multiple Address Regions” on page 42.

Table 4-3 Slave 1 Memory Region Definition (Continued)

Label	Parameter Definition
Number of Memory Regions in Normal Mode?	**Parameter Name:** MR_N1 Legal 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 (0) **Dependencies:** This parameter option is available only for slave 1 and 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). **Description:** The number of memory regions in Normal Mode for slave 1. For more information, see “Multiple Address Regions” on page 42.
Split Capable?	**Parameter Name:** SPLIT_CAPABLE_1 **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** When a slave is aliased, it takes its split capability from the slave it is aliased to. Therefore, this option will be dimmed if the HSEL_ONLY_S1 is set to Yes. This parameter is disabled if AHB Lite mode is enabled; (AHB_LITE = 1). **Description:** If the slave has an hsplit bus, then set this parameter to True (1).
Slave Visibility Mode	**Parameter Name:** VISIBLE_1 **Legal Values:** Normal, Boot, Normal & Boot **Default Value:** Normal **Dependencies:** This parameter option is active only if you enable the Memory Remap Feature (REMAP = 1) and have configured to use an internal decoder (AHB_HAS_XDCDR = 0) in the top-level parameter options. **Description:** 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” on page 42.

Table 4-4 provides the memory region definition parameters for slave 2 through 15. Slave 1 was designed with many regions for those designs that may want to allocated this address space to a memory controller.

Table 4-4 Slaves 2–15 Memory Region Definition

Label	Parameter Definition
Number of slave which returns data and response	**Parameter Name:** ALIAS_S*(i)*, where i = 2 through 15 **Legal Values:** 1 to NUM_IAHB_SLAVES **Default Value:** 1 **Dependencies:** This parameter option is available only if the HSEL_ONLY_S*i* = 1. **Description:** The value must be less than or equal to the value of NUM_IAHB_SLAVES. The value of this parameter cannot equal *i*, meaning you cannot alias this slave to itself.
Alias this slave to another system slave?	**Parameter Name:** HSEL_ONLY_S(*i*), where i = 2 through 15 **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description:** Generates only hsel for this slave and aliases it to another slave. For more information, see “Multiple Address Regions” on page 42.
Support Multiple Memory Regions in Boot Mode?	**Parameter Name:** MR_B(*i*), where i = 2 through 15 **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** 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). **Description:** Number of regions in Boot Mode for slave(*i*). For more information, see “Multiple Address Regions” on page 42.
Support Multiple Memory Regions in Normal Mode?	**Parameter Name:** MR_N(*i*), where i = 2 through 15 **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** 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). **Description:** Number of regions in Normal Mode for slave(*i*). For more information, see “Multiple Address Regions” on page 42.
Split Capable?	**Parameter Name:** SPLIT_CAPABLE_*(i)*, where i = 2 to 15 **Legal Values:** True (1) or False (0) **Default Value:** No **Dependencies:** 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*i* = 1. This parameter is disabled if AHB Lite mode is enabled; (AHB_LITE = 1). **Description:** If the slave has an hsplit bus, then set this parameter to True (1).

Table 4-4 Slaves 2–15 Memory Region Definition (Continued)

Label	Parameter Definition
Slave Visibility Mode	**Parameter Name:** VISIBLE_(*i*), where i = 2 through 15 **Legal Values:** Normal (1), Boot (2), Normal & Boot (3) **Default Value:** Normal (1) **Dependencies:** 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. **Description:** 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" on page 42.

4.1.4 Normal Mode Address Map Parameters

Table 4-5 describes the normal mode address map parameters for the first DW_ahb slave (slave 1), which can have up to eight address regions in normal, whereas slaves 2 through 15 can have only up to two address regions.

Table 4-5 Normal Mode Address Map (Slave 1) Configuration

Label	Parameter Definition
Normal Mode Region *x* Start Address (where *x* is 1 to 8)	**Parameter Name:** R*x*_N_SA_1, where *x* is 1 to 8 **Legal Values:** 0x0000000 to 0xfffffc00 **Default Value:** for regions 1 to 8, the default value is: Region 1: 0x*2000000* Region 2: 0x*3000000* Region 3: 0x*4000000* Region 4: 0x*5000000* Region 5: 0x6000000 Region 6: 0x7000000 Region 7: 0x8000000 Region 8: 0x9000000 **Dependencies:** 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). **Description:** Regions 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.

Table 4-5 Normal Mode Address Map (Slave 1) Configuration (Continued)

Label	Parameter Definition
Normal Mode Region *x* End Address (where *x* is 1 to 8)	**Parameter Name:** R*x*_N_EA_1 (where *x* is 1 to 8) **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** for regions 1 to 8, the default value is: Region 1: 0x200ffff Region 2: 0x300ffff Region 3: 0x*400ffff* Region 4: 0x*500ffff* Region 5: 0x600ffff Region 6: 0x700ffff Region 7: 0x800ffff Region 8: 0x900ffff **Dependencies:** This parameter option is available only if the MR_N(i) is set to *x* and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Description:** Regions 1 through 8, normal addressing mode, end address for slave 1. Specified if the peripherals address region is spread over multiple regions.

Table 4-6 provides the normal mode address map configuration parameters for slaves 2 through 15.

Table 4-6 Normal Mode Address Map (Slaves 2 – 15) Configuration

Label	Parameter Definition
Normal Mode Region 1 Start Address	**Parameter Name:** R1_N_SA_(i), where i = 2 through 15 **Legal Values:** 0x00000000 to 0xfffffc00 **Default Value:** for slaves 2 to 15, the 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 **Dependencies:** 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). **Description:** Region 1, normal addressing mode, start address for slaves 2 through 15.

Table 4-6 Normal Mode Address Map (Slaves 2 – 15) Configuration (Continued)

Label	Parameter Definition
Normal Mode Region 1 End Address	**Parameter Name:** R1_N_EA_(i)*, where i = 2 through 15* **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** for slaves 2 to 15, the default value is: Slave 2: 0xa00ffff Slave 3: 0xb00ffff 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 **Dependencies:** 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). **Description:** Region 1, normal addressing mode, end address for slaves 2 through 15.
Normal Mode Region 2 Start Address	**Parameter Name:** R2_N_SA_(i)*, where i = 2 through 15* **Legal Values:** 0x00000000 to 0xfffffc00 **Default Value:** for slaves 2 to 15, the 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 **Dependencies:** This parameter option is available only if the “Multiple Memory Regions in Normal Mode” is set to True (1) and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Description:** Region 2, normal addressing mode, start address for slaves 2 through 15. Specified if the peripherals address region is spread over multiple regions.

Table 4-6 Normal Mode Address Map (Slaves 2 – 15) Configuration (Continued)

Label	Parameter Definition
Normal Mode Region 2 End Address	**Parameter Name:** R2_N_EA_(i) **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** for slaves 2 to 15, the 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 **Dependencies:** This parameter option is available only if the “Multiple Memory Regions in Normal Mode” is set to True (1) and if you have an internal decoder (AHB_HAS_XDCDR = 0). **Description:** Region 2, normal addressing mode, end address for slaves 2 through 15. Specified if the peripherals address region is spread over multiple regions.

4.1.5 Boot Mode Address Map Parameters

Table 4-7 describes the boot mode address map parameters for the first DW_ahb slave (slave 1), which can have up to eight address regions in boot mode, whereas slaves 2 through 15 can have only up to two address regions.

Table 4-7 Boot Mode Address Map (Slave 1) Configuration

Label	Parameter Definition
Boot Mode Region *x* Start Address (where *x* is 1 to 8)	**Parameter Name:** R*x*_B_SA_1 (where *x* is 1 to 8) **Legal Values:** 0x00000000 to 0xfffffc00 **Default Value:** for regions 1 to 8, the default value is: Region 1: 0x*27*000000 Region 2: *0x28*000000 Region 3: 0x*29000000* Region 4: 0x*2a000000* Region 5: 0x2b000000 Region 6: 0x2c000000 Region 7: 0x2d000000 Region 8: 0x2e000000 **Dependencies:** 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. **Description:** Regions 1 through 8, boot addressing mode, start address for slave 1. Specified if the peripherals address region is spread over multiple regions.
Boot Mode Region *x* End Address (where *x* is 1 to 8)	**Parameter Name:** R*x*_B_EA_1 (where *x* is 1 to 8) **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** for regions 1 to 8, the default value is: Region 1: 0x2700ffff Region 2: 0x2800ffff Region 3: 0x*2900ffff* Region 4: 0x*2a00ffff* Region 5: 0x2b00ffff Region 6: 0x2c00ffff Region 7: 0x2d00ffff Region 8: 0x2e00ffff **Dependencies:** 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. **Description:** Regions 1 through 8, boot addressing mode, end address for slave 1. Specified if the peripherals address region is spread over multiple regions.

Table 4-8 provides the boot mode address map configuration parameters for slaves 2 through 15.

Table 4-8 Boot Mode Address Map (Slaves 2 – 15) Configuration

Label	Parameter Definition
Boot Mode Region 1 Start Address	**Parameter Name:** R1_B_SA_(i), where i = 2 through 15 **Legal Values:** 0x00000000 to 0xfffffc00 **Default Value:** for slaves 2 to 15, the 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 **Dependencies:** This option is applicable only if you have an internal decoder (AHB_HAS_XDCDR = 0) and if REMAP = 1. **Description:** Region 1, boot addressing mode, start address for slaves 2 through 15.
Boot Mode Region 1 End Address	**Parameter Name:** R1_B_EA_(i), where i = 2 through 15 **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** for slaves 2 to 15, the 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 **Dependencies:** This option is applicable only if you have an internal decoder (AHB_HAS_XDCDR = 0) and if REMAP = 1. **Description:** Region 1 boot, addressing mode, end address for slaves 2 through 15.

Table 4-8 Boot Mode Address Map (Slaves 2 – 15) Configuration (Continued)

Label	Parameter Definition
Boot Mode Region 2 Start Address	**Parameter Name:** R2_B_SA_(i), where i = 2 through 15 **Legal Values:** 0x00000000 to 0xfffffc00 **Default Value:** 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 **Dependencies:** This parameter option is available only if “Support Multiple Memory Regions in Boot Mode” is set to True (1), if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Description:** Region 2, boot addressing mode, start address for slave N.
Boot Mode Region 2 End Address	**Parameter Name:** R2_B_EA_(i), where i = 2 through 15 **Legal Values:** 0x000003ff to 0xffffffff **Default Value:** 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 **Dependencies:** This parameter option is available only if “Support Multiple Memory Regions in Boot Mode” is set to True (1), if you have an internal decoder (AHB_HAS_XDCDR = 0), and if REMAP = 1. **Description:** Region 2, boot addressing mode, end address for slaves 2 through 15.

4.1.6 Arbiter Priority Parameters

Specify the priority level for each DW_ahb master by setting the configuration parameter described in Table 4-9. The table includes the coreConsultant field name and the parameter definition. You will need the parameter name if you want to run coreConsultant in batch mode.

Table 4-9 Arbiter Priority Assignment Parameters

Label	Parameter Definition
Master *i* (i = 1 through 15)	**Parameter Name:** PRIORITY_*i* **Legal Values:** 0x1 to 0xf **Default Value:** The default value is equal to the master number. For instance, master 1 has a default priority of 1 (0x1), master 2 has a default priority of 2 (0x2), and so on. **Dependencies:** None. **Description:** 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.

4.1.7 Arbiter Slave Interface Parameters

Table 4-10 provides the parameters for the DW_ahb arbiter slave interface, which is activated when AHB_HAS_ARBIF=1. The information in the tables shows the coreConsultant field name and the parameter definition.

Table 4-10 Slave Arbiter Configuration Parameters

Label	Parameter Definition
AHB Arbiter Start Address (Normal Mode)	**Parameter Name:** R1_N_SA_0 **Legal Values:** 0x0 to 0xfffffc00 **Default Value:** 0x1000000 **Dependencies:** 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). **Description:** Normal Mode start address for AHB arbiter.
AHB Arbiter End Address (Normal Mode)	**Parameter Name:** R1_N_EA_0 **Legal Values:** 0x0 to 0xffffffff **Default Value:** 0x10003ff **Dependencies:** 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). **Description:** Normal Mode end address for AHB arbiter

Table 4-10 Slave Arbiter Configuration Parameters (Continued)

Label	Parameter Definition
AHB Arbiter Start Address (Boot Mode)	**Parameter Name:** R1_B_SA_0 **Legal Values:** 0x0 to 0xfffffc00 **Default Value:** 0x26000000 **Dependencies:** 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). **Description:** Boot Mode start address for AHB arbiter.
AHB Arbiter End Address (Boot Mode)	**Parameter Name:** R1_B_EA_0 **Legal Values:** 0x0 to 0xffffffff **Default Value:** 0x260003ff **Dependencies:** 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). **Description:** Boot Mode end address for AHB arbiter.
Use Hard-coded Arbiter Priorities?	**Parameter Name:** HC_PRIORITIES **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** If there is no arbiter slave interface, this parameter is dimmed and hardcoded to Yes. **Description:** If this parameter is set to Yes, only the priorities will be read. If it is set to No, the priorities can be programmed during runtime.
Default Master Number	**Parameter Name:** DFLT_MST_NUM **Legal Values:** 0 to NUM_AHB_MASTERS **Default Value:** 0 **Dependencies:** 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. **Description:** A default master is required according to the *AMBA Specification (Rev. 2.0)*. You can set this to 0 if you want the dummy master to act as the default master. For more information, refer to “Default Master versus Dummy Master” on page 37.
Use Hard-coded Default Master	**Parameter Name:** HC_DFLT_MSTR **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** If there is no arbiter slave interface, this parameter is dimmed and hardcoded to Yes. **Description:** If you set this parameter to Yes, only the ID of the default master will be read. For more information, refer to “Hard-coded Default Master” on page 37.

Table 4-10 Slave Arbiter Configuration Parameters (Continued)

Label	Parameter Definition
Include Early Burst Termination Support	**Parameter Name:** EBTEN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** If there is no arbiter slave interface, this parameter is dimmed and set to False (0). **Description:** 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" on page 38.
Generate slave select on the interface	**Parameter Name:** GEN_HSEL0 Legal Values: True (1) or False (0) **Default Value:** False (0) **Dependencies:** The arbiter slave interface must be included in the design (AHB_HAS_ARBIF=1) and the system must not be AMBA Lite (AHB_LITE=0) for this parameter to be set to True. **Description:** If set to True, the AHB interface signals for Slave 0 are provided as outputs from DW_ahb for reference. These signals are hsel_s0, hready_resp_s0, hrdata_s0, and htrans_s0. If set to False, these signals are not provided as outputs. In coreConsultant, this parameter is chosen automatically, depending on how AHB_HAS_ARBIF and AHB_LITE are configured. If AHB_LITE=1 or if 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.

4.1.8 Parameters for Weighted-Token Control Signals

Table 4-11 describes the parameters of weighted-token arbitration. These signals are included on the interface if the AHB is not an AHB Lite configuration and if weighted-token arbitration has been enabled.

Table 4-11 Configuration Parameters for Weighted-Token Arbitration

Label	Parameter Definition
Counting Mode	**Parameter Name:** AHB_TPS_MODE **Legal Values:** Clock-Cycle (0) or Bus-Cycle (1) **Default Value:** Clock-Cycle (0) **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. **Description:** The token counters can count on clock cycles or on bus cycles to calculate the number of tokens a master is using.
Bits in arbitration counter	**Parameter Name:** AHB_TCL_WIDTH **Legal Values:** 4 to 32 **Default Value:** 32 **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. **Description:** 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.
Bits in master token counter	**Parameter Name:** AHB_CCL_WIDTH **Legal Values:** 4 to 32 **Default Value:** 32 **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. The number of bits in the arbitration counter must not be less than the number of bits in the master token counter. **Description:** 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.
Use Hard-coded tokens?	**Parameter Name:** AHB_HC_TOKENS **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. **Description:** The length of the arbitration period and the number of clock tokens for each master can be hardcoded to reduce the overall register count.

Table 4-11 Configuration Parameters for Weighted-Token Arbitration

Label	Parameter Definition
Total Cycle Limit	**Parameter Name:** AHB_TCL **Legal Values:** 0x0 to 0xffffffff **Default Value:** 0x0000ffff **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. The maximum value is controlled by the number of bits in the arbitration counter. **Description:** 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.
Master Clock Tokens	**Parameter Name:** AHB_CL_M(*i*), for *i* in 1 to NUM_AHB_MASTERS **Legal Values:** 0x0 to 0xffffffff **Default Value:** 0x0000ffff **Dependencies:** Configured AHB is not an AHB Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. The maximum value is controlled by the number of bits in a master token counter. **Description:** 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.

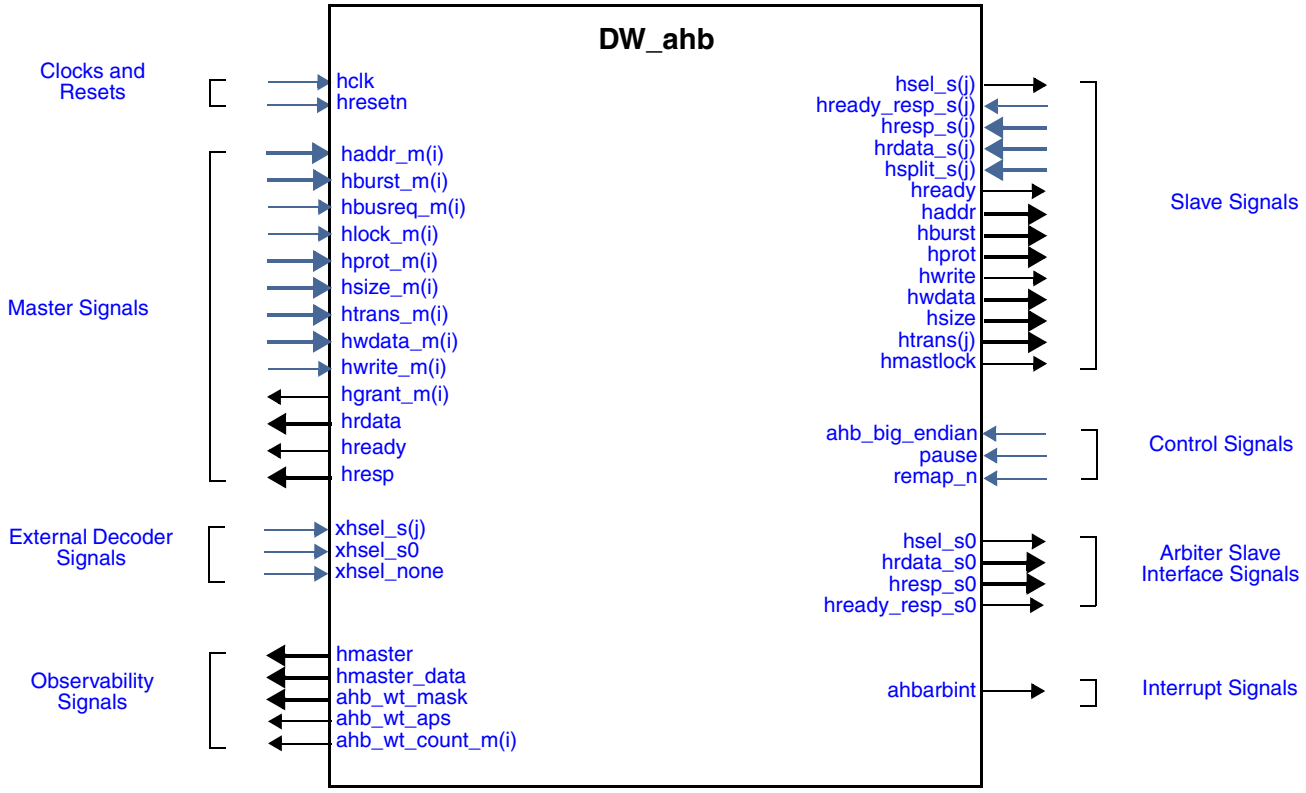
5 Signals

This chapter describes the DW_ahb I/O signals.

5.1 DW_ahb Interface Diagram

Figure 5-1 shows the I/O signals for DW_ahb.

Figure 5-1 DW_ahb Interface Diagram



5.2 DW_ahb Signal Descriptions

Table 5-1 identifies the signals that are associated with the DW_ahb.

Note

The Description column in Table 5-1 provides detailed information about each signal.

- In the **Registered** field, a “Yes” indicates whether an I/O signal is directly connected to an internal register and nothing else. An I/O signal is also considered to be registered if the signal is connected to one or more inverters or buffers between the I/O port and internal register, but not connected to any logic that involves another signal.
- The **Input/Output Delay** field provides the percentage of the clock cycle assumed to be used by logic outside this design. The given value is used to automatically define the default synthesis constraints for input/output delay.

Table 5-1 DW_ahb Signal Description

Name	Width	I/O	Description
Clocks and Resets			
hclk	1 bit	Input	AHB Clock Signal. This clock times all bus transfers. All signal timings are related to the rising edge of hclk. **Active State:** N/A **Registered:** N/A **Synchronous to:** N/A **Default Input Delay:** N/A
hresetn	1 bit	Input	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(i) indicates the IDLE state. **Active State:** Low **Registered:** N/A **Synchronous to:** 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. **Default Input Delay:** N/A
Master Signals Generated for i = 1; i <= Number of AHB Masters (1 to 15)			
haddr_m(i)	32 bits	Input	AHB Address Bus. Address bus for each master in the system. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
hburst_m(i)	3 bits	Input	Indicates if the transfer constitutes part of a burst. Each master in the system has its own hburst_m(i) bus. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hbusreq_m(i)	1 bit	Input	*Optional.* 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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hlock_m(i)	1 bit	Input	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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hprot_m(i)	4 bits	Input	Protection Control Signals. There is one hprot_m(i) bus for each master in the system. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hsize_m(i)	3 bits	Input	Indicates size of transfer. There is one hsize_m(i) bus for each master in the system. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
htrans_m(i)	2 bits	Input	Indicates the type of transfer being performed. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
hwdata_m(i)	[w–1:0]	Input	Transfer write data. Each master has its own hwdata_m(i) signal. Width: Width of the AHB data bus. Maximum width is 256 bits. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hwrite_m(i)	1 bit	Input	Transfer write signal. When HIGH, this signal indicates a write transfer. When LOW, this signal indicates a read transfer. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hgrant_m(i)	1 bit	Output	*Optional.* 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. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** This signal is not included in an AHB Lite configuration.
hrdata	[w–1:0]	Output	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. Width: Width of the AHB data bus. Maximum width is 256 bits. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hready	1 bit	Output	Ready response from selected slave. This signal is passed to all AHB masters and slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hresp	2 bits	Output	Transfer response. This signal is passed to all AHB masters. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
Slave Signals Generated for j = 1; j <= Number of AHB Slaves (1 to 15)			
hsel_s(j)	1 bit	Output	*Optional*. 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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** Included when internal decoder is used (parameter AHB_HAS_XDCDR = 0).
hready_resp_s(j)	1 bit	Input	*Optional*. Response from slave. When asserted, current transfer has completed. There is one hready_resp_s(j) signal for each slave in the system. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hresp_s(j)	2 bits	Input	Transfer response from individual slave. There is one hresp_s(j) bus for each slave in the system. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hrdata_s(j)	[w–1:0]	Input	*Optional*. Readback data from slaves. Each slave has its own hrdata bus. Width: Width of the AHB data bus. Maximum width is 256 bits. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
hsplit_s(j)	16 bits	Input	*Optional*. 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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** Exists only if slave is split capable (SPLIT_CAPABLE_*(i)* = 1).

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
hready	1 bit	Output	Ready response from selected slave. This signal is passed to all AHB masters and slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
haddr	32 bits	Output	Address bus from selected master. This is passed to all AHB slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hburst	3 bits	Output	Burst type from selected master. This is passed to all AHB slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hprot	4 bits	Output	Transfer protection information from selected master. Used only by slaves that can support this feature. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hwrite	1 bit	Output	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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hwdata	[w–1:0]	Output	Write data bus from selected master. This signal is passed to all AHB slaves. Width: Width of the AHB data bus. Maximum width is 256 bits. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
hsize	3 bits	Output	Transfer size from selected master. Indicates the size of the transfer. This is passed to all AHB slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
htrans(j)	2 bits	Output	*Optional*. Transfer type from selected master. This is passed to all AHB slaves. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30%
hmastlock	1 bit	Output	Asserted to indicate that the transfer currently in progress is part of a locked transaction. This signal is driven by the arbiter. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 30%
Control Signals			
ahb_big_endian	1 bit	Input	*Optional*. AHB Endianness. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** This signal controls the endianness of the DW_ahb instead of being set at configuration time. If the configuration parameter AHB_XENDIAN is set to True, this signal is included on the interface and controls the endianness. If AHB_XENDIAN is set to False, the signal is not included on the interface.
pause	1 bit	Input	*Optional*. Asserted to put the arbiter into low-power mode by granting the dummy master ownership of the bus. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** This signal is included if the PAUSE parameter is enabled.

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
remap_n	1 bit	Input	*Optional*. 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. **Active State:** Low **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20%
External Decoder Signals			
xhsel_s(j)	1 bit	Input	Optional. Slave select line. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** This signal is included if the DW_ahb is configured to use an external decoder (AHB_HAS_XDCDR = 1).
xhsel_s0	1 bit	Input	Optional. Slave select from Arbiter Slave interface. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** This signal is included if the DW_ahb is configured to use an external decoder (AHB_HAS_XDCDR = 1) and the arbiter interface is included.
xhsel_none	1 bit	Input	*Optional*. Default slave select. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Input Delay:** 20% **Dependencies:** This signal is included if the DW_ahb is configured to use an external decoder (AHB_HAS_XDCDR = 1).
Observability Signals			
hmaster	4 bits	Output	Indicates which master currently has ownership of the address and control bus. This is generated by the arbiter. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 20%

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
hmaster_data	4 bits	Output	Indicates which master currently has ownership of the data bus. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 30%
ahb_wt_mask	[*y*:1]	Output	*Optional*. 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. Width: y = NUM_AHB_MASTERS **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** This signal is included when the weighted token mask is configured.
ahb_wt_aps	1 bit	Output	*Optional*. 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. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** This signal is included when the weighted token mask is configured and indicates when the weighted-token arbitration period starts.
ahb_wt_count_m(i)	1 bit	Output	*Optional*. 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. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** These signals are included on the interface when the configuration parameter AHB_WTEN = True (1) and AHB_WTEN_DEBUG = True (1).

Table 5-1 DW_ahb Signal Description (Continued)

Name	Width	I/O	Description
Arbiter Slave Interface Signals			
hsel_s0	1 bit	Output	*Optional*. When asserted, indicates that the arbiter slave has been selected. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 30% **Dependencies:** This signal is included if GEN_HSEL0 is set to True.
hrdata_s0	[w-1:0]	Output	*Optional*. Readback data from Arbiter Slave interface. Width: Width of the AHB data bus. Maximum width is 256 bits. **Active State:** High **Registered:** No **Synchronous to:** hclk **Default Output Delay:** 70% **Dependencies:** This signal is included if GEN_HSEL0 is set to True.
hresp_s0	2 bits	Output	*Optional*. Transfer response from Arbiter Slave interface. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 70% **Dependencies:** This signal is included if GEN_HSEL0 is set to True.
hready_resp_s0	1 bit	Output	*Optional*. Response from Arbiter Slave interface. When asserted, current transfer has completed. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 70% **Dependencies:** This signal is included if GEN_HSEL0 is set to True.
Interrupt Signals			
ahbarbint	1 bit	Output	*Optional*. Interrupt signal to Interrupt Controller. The arbiter will flag an interrupt when an Early Burst Termination occurs. **Active State:** High **Registered:** Yes **Synchronous to:** hclk **Default Output Delay:** 20% **Dependencies:** This signal is not included if the EBTEN parameter is set to No.

6 Registers

This chapter describes the programmable registers of the DW_ahb.

6.1 Register Memory Map

Table 6-1 shows the memory map for the DW_ahb peripheral.

Table 6-1 Memory Map – Arbiter Slave Interface Registers

Name	Address Offset	R/W	Width	Description
Master Priority Level Registers				
PL1	0x00	R/W	4 bits	Arbitration priority for master 1 **Reset Value:** ‘PRIORITY_1
PL2	0x04	R/W	4 bits	Arbitration priority for master 2 **Reset Value:** ‘PRIORITY_2
PL3	0x08	R/W	4 bits	Arbitration priority for master 3 **Reset Value:** ‘PRIORITY_3
PL4	0x0c	R/W	4 bits	Arbitration priority for master 4 **Reset Value:** ‘PRIORITY_4
PL5	0x10	R/W	4 bits	Arbitration priority for master 5 **Reset Value:** ‘PRIORITY_5
PL6	0x14	R/W	4 bits	Arbitration priority for master 6 **Reset Value:** ‘PRIORITY_6
PL7	0x18	R/W	4 bits	Arbitration priority for master 7 **Reset Value:** ‘PRIORITY_7
PL8	0x1c	R/W	4 bits	Arbitration priority for master 8 **Reset Value:** ‘PRIORITY_8
PL9	0x20	R/W	4 bits	Arbitration priority for master 9 **Reset Value:** ‘PRIORITY_9

Table 6-1 Memory Map – Arbiter Slave Interface Registers (Continued)

Name	Address Offset	R/W	Width	Description
PL10	0x24	R/W	4 bits	Arbitration priority for master 10 **Reset Value:** ‘PRIORITY_10
PL11	0x28	R/W	4 bits	Arbitration priority for master 11 **Reset Value:** ‘PRIORITY_11
PL12	0x2c	R/W	4 bits	Arbitration priority for master 12 **Reset Value:** ‘PRIORITY_12
PL13	0x30	R/W	4 bits	Arbitration priority for master 13 **Reset Value:** ‘PRIORITY_13
PL14	0x34	R/W	4 bits	Arbitration priority for master 14 **Reset Value:** ‘PRIORITY_14
PL15	0x38	R/W	4 bits	Arbitration priority for master 15 **Reset Value:** ‘PRIORITY_15
Early Burst Termination Registers				
EBTCOUNT	0x3c	R/W	10 bits	Early burst termination count register **Reset Value:** 0x0
EBT_EN	0x40	R/W	1 bit	Early burst termination enable **Reset Value:** 0x0
EBT	0x44	Read to Clear	1 bit	Early burst termination register **Reset Value:** EBT cleared
Default Master Register				
DFT_MST	0x48	R/W	4 bits	Default master ID number **Reset Value:** 0x0
Weighted-Token Arbitration Registers				
WTEN	0x4c	R/W	1 bit	Weighted- Token Arbitration Scheme Enable. **Reset Value:** 0x0
AHB_TCL	0x50	R/W	See Description	Master clock refresh period. **Width:** TCL = AHB_TCL_WIDTH
Address locations for master tokens				
AHB_CL_M(i)	for i in 1-15 0x50 + (0x04*i)	R/W	See Description	Master clock tokens. **Width:** CCL = AHB_CCL_WIDTH

Table 6-1 Memory Map – Arbiter Slave Interface Registers (Continued)

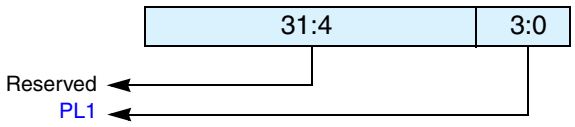
Name	Address Offset	R/W	Width	Description
AHB_COMP_VERSION	0x90	R	32 bits	Component version ID **Reset Value:** See the DW_ahb releases table in the AMBA 2 Release Notes

6.2 Register and Field Descriptions

The following sections contain the memory diagrams and field descriptions for the individual registers.

6.2.1 PL1

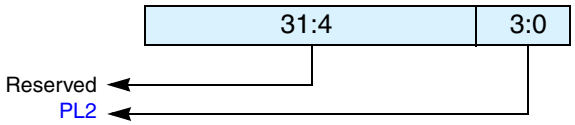
- **Name:** Arbitration Priority Master 1 Register
- **Size:** 4 bits
- **Address Offset:** 0x00
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL1	R/W	Arbitration priority for master 1 **Reset Value:** ‘PRIORITY_1

6.2.2 PL2

- **Name:** Arbitration Priority Master 2 Register
- **Size:** 4 bits
- **Address Offset:** 0x04
- **Read/Write Access:** Read/Write

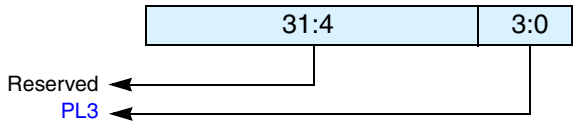


Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL2	R/W	Arbitration priority for master 2 **Reset Value:** ‘PRIORITY_2

6.2.3 PL3

- **Name:** Arbitration Priority Master 3 Register
- **Size:** 4 bits

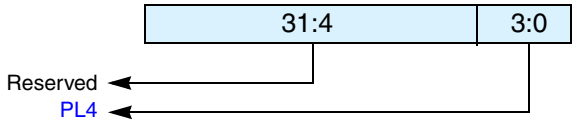
- **Address Offset:** 0x08
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL3	R/W	Arbitration priority for master 3 **Reset Value:** ‘PRIORITY_3

6.2.4 PL4

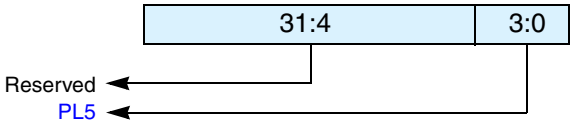
- **Name:** Arbitration Priority Master 4 Register
- **Size:** 4 bits
- **Address Offset:** 0x0c
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL4	R/W	Arbitration priority for master 4 **Reset Value:** ‘PRIORITY_4

6.2.5 PL5

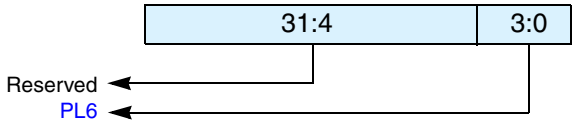
- **Name:** Arbitration Priority Master 5 Register
- **Size:** 4 bits
- **Address Offset:** 0x10
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL5	R/W	Arbitration priority for master 5 **Reset Value:** ‘PRIORITY_5

6.2.6 PL6

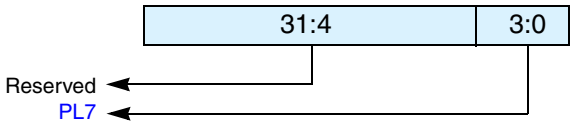
- **Name:** Arbitration Priority Master 6 Register
- **Size:** 4 bits
- **Address Offset:** 0x14
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL6	R/W	Arbitration priority for master 6 **Reset Value:** ‘PRIORITY_6

6.2.7 PL7

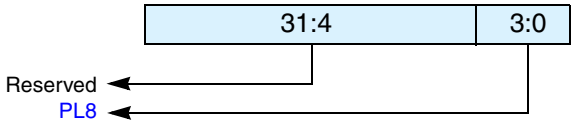
- **Name:** Arbitration Priority Master 7 Register
- **Size:** 4 bits
- **Address Offset:** 0x18
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL7	R/W	Arbitration priority for master 7 **Reset Value:** ‘PRIORITY_7

6.2.8 PL8

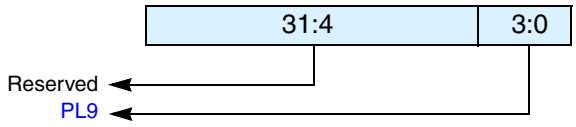
- **Name:** Arbitration Priority Master 8 Register
- **Size:** 4 bits
- **Address Offset:** 0x1c
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL8	R/W	Arbitration priority for master 8 **Reset Value:** ‘PRIORITY_8

6.2.9 PL9

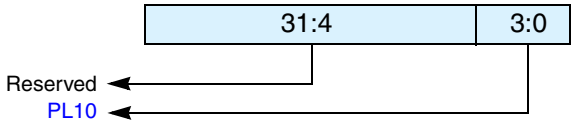
- **Name:** Arbitration Priority Master 9 Register
- **Size:** 4 bits
- **Address Offset:** 0x20
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL9	R/W	Arbitration priority for master 9 **Reset Value:** ‘PRIORITY_9

6.2.10 PL10

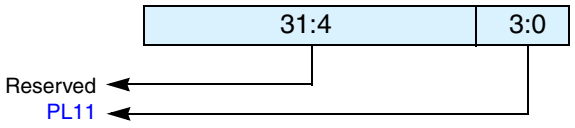
- **Name:** Arbitration Priority Master 10 Register
- **Size:** 4 bits
- **Address Offset:** 0x24
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL10	R/W	Arbitration priority for master 10 **Reset Value:** ‘PRIORITY_10

6.2.11 PL11

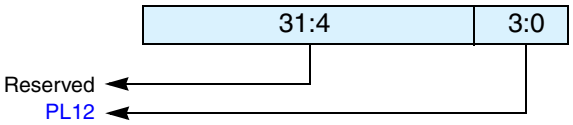
- **Name:** Arbitration Priority Master 11 Register
- **Size:** 4 bits
- **Address Offset:** 0x28
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL11	R/W	Arbitration priority for master 11 **Reset Value:** ‘PRIORITY_11

6.2.12 PL12

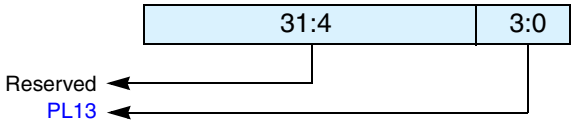
- **Name:** Arbitration Priority Master 12 Register
- **Size:** 4 bits
- **Address Offset:** 0x2c
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL12	R/W	Arbitration priority for master 12 **Reset Value:** ‘PRIORITY_12

6.2.13 PL13

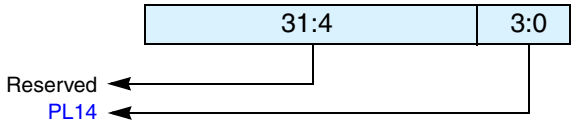
- **Name:** Arbitration Priority Master 13 Register
- **Size:** 4 bits
- **Address Offset:** 0x30
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL13	R/W	Arbitration priority for master 13 **Reset Value:** ‘PRIORITY_13

6.2.14 PL14

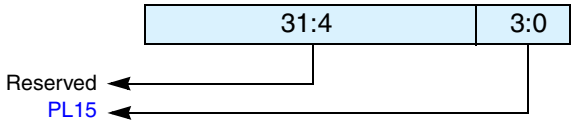
- **Name:** Arbitration Priority Master 14 Register
- **Size:** 4 bits
- **Address Offset:** 0x34
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL14	R/W	Arbitration priority for master 14 **Reset Value:** ‘PRIORITY_14

6.2.15 PL15

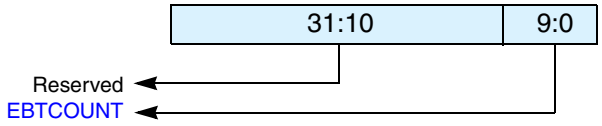
- **Name:** Arbitration Priority Master 15 Register
- **Size:** 4 bits
- **Address Offset:** 0x38
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:4	Reserved		
3:0	PL15	R/W	Arbitration priority for master 15 **Reset Value:** ‘PRIORITY_15

6.2.16 EBTCOUNT

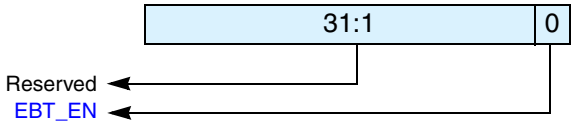
- **Name:** Early Burst Termination Count Register
- **Size:** 10 bits
- **Address Offset:** 0x3c
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:10	Reserved		
9:0	EBTCOUNT	R/W	Early burst termination count register. Maximum number of cycles a transfer can take before being subject to an early burst termination. **Reset Value:** 0x0

6.2.17 EBT_EN

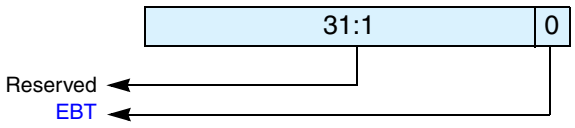
- **Name:** Early Burst Termination Enable
- **Size:** 1 bit
- **Address Offset:** 0x40
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:1	Reserved		
0	EBT_EN	R/W	Early burst termination count register **Reset Value:** 0x0 (disabled)

6.2.18 EBT

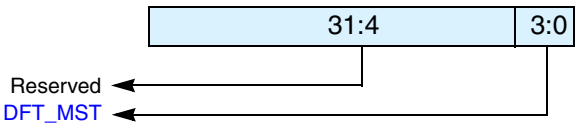
- **Name:** Early Burst Termination Register
- **Size:** 1 bit
- **Address Offset:** 0x44
- **Read/Write Access:** Read to clear



Bits	Name	R/W	Description
31:1	Reserved		
0	EBT	Read to clear	Early burst termination register. Set when an Early Burst Termination takes place. The register is cleared when read by the processor. **Reset Value:** EBT cleared

6.2.19 DFT_MST

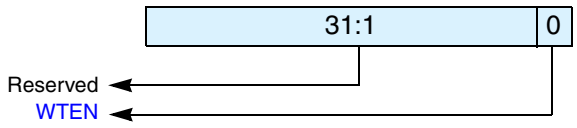
- **Name:** Default Master ID Number Register
- **Size:** 4 bits
- **Address Offset:** 0x48
- **Read/Write Access:** Read/Write*



Bits	Name	R/W	Description
31:4	Reserved		
3:0	DFT_MST	R/W*	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. **Reset Value:** 0x0

6.2.20 WTEN

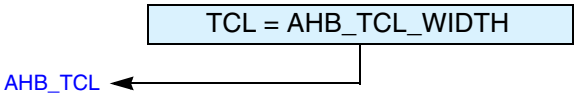
- **Name:** Weighted-Token Arbitration Scheme Enable Register
- **Size:** 1 bit
- **Address Offset:** 0x4c
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
31:1	Reserved		
0	WTEN	R/W	Weighted-token arbitration scheme enable register. Accessible only when the weighted token scheme is configured through coreConsultant. **Reset Value:** 0x0

6.2.21 AHB_TCL

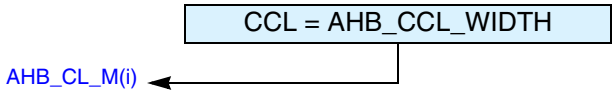
- **Name:** Default Master Register
- **Size:** TCL = AHB_TCL_WIDTH
- **Address Offset:** 0x50
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
See Description	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. **Width:** TCL = AHB_TCL_WIDTH

6.2.22 AHB_CL_M(*i*)

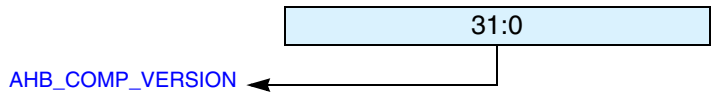
- **Name:** Master Clock Token Register
- **Size:** CCL = AHB_CCL_WIDTH
- **Address Offset:** For *i* in 1-15, 0x50 + (0x04**i*)
- **Read/Write Access:** Read/Write



Bits	Name	R/W	Description
See Description	AHB_CL_M(*i*)	R/W	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. **Width:** CCL = AHB_CCL_WIDTH

6.2.23 AHB_COMP_VERSION

- **Name:** Component Version ID Register
- **Size:** 32 bits
- **Address Offset:** 0x90
- **Read/Write Access:** Read



Bits	Name	R/W	Description
31:0	AHB_COMP_VERSION	R	ASCII value for each number in the version, followed by *. For example 32_30_31_2A represents the version 2.01*. **Reset Value:** See the DW_ahb releases table in the "AMBA 2 Release Notes"

7 Programming the DW_ahb

This chapter describes the programmable features of the DW_ahb.

7.1 Programming Considerations

The following sections outline programming considerations for the DW_ahb.

7.1.1 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 will be 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 3-6 on page 41.

7.1.2 Arbiter Slave Interface Registers

Table 6-1 on page 79 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.

7.1.3 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 will be 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.

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 will be ignored and the contents of the register will remain unchanged.

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 Table 6-1 on page 79 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 was greater than 1 KB. Any attempt to access locations within the 1 KB block, other than those specified, will result in an error.

7.1.4 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 will terminate 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 will be disabled by default, and will be 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.

7.1.5 Default Master Register

If no master is requesting bus access, the DW_ahb arbiter will grant 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 will also act 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.

7.1.6 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.

8 Verification

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

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

8.1 Overview of Vera Tests

The DW_ahb verification testbench performs the following set of tests, which are listed in the Tests tab of the coreConsultant Verification activity. By default, all of the tests are enabled to run. The tests have been written to exhaustively verify the functionality and have also achieved maximum RTL code coverage.

8.1.1 Internal Decoder

This test verifies the address decoder functionality of DW_ahb. Master 1 is always granted the bus and scans the memory map progressively. A contiguous set of address locations before and after each memory map region boundary (start address or end address) is exercised.

If both addressing modes are available, the testbench runs Boot Mode initially and then repeats the test for Normal Mode. If DW_ahb is only configured for Normal Mode, then the Boot Mode tests are not run.

The test checks that a mismatch between observed hsel_s(j) and expected values calculated internally by the test according to the configuration. During the test all the slaves in the system are accessed in order to exercise all hsel_s(j)lines. All the slaves are configured for 0 wait states and OKAY response.

8.1.2 Default Slave

The default slave functionality of DW_ahb captures all unspecified address locations and provides the data and transfer response if a master addresses any of the unspecified memory map locations.

These tests build on the decoder test and use random addresses to generate addresses that are not assigned to any slave. The following checks are performed:

- Default slave responds with a two-cycle ERROR response when an unassigned address location in the system is addressed and htrans is not IDLE.
- Default slave responds with a zero wait state OKAY response when an unassigned address location in the system is addressed and htrans is IDLE.

8.1.3 External Decoder

Exercises all xhsel_s(i) and xhsel_none lines.

8.1.4 Dummy Master/Default Master

This test verifies the functionality of the Dummy Master and the Default Master implemented in DW_ahb. The following checks are performed:

- When the dummy master owns the bus, it drives only IDLE transfers on the bus.
- The dummy master never requests the bus, but is granted the bus if it is specified as the default master.
- The default master owns the bus after reset and when nobody is requesting access to the bus.
- The default master observed on the bus corresponds to the programmed/configured value.

8.1.5 Arbiter Slave Interface

When DW_ahb includes an AHB slave interface (as specified by the AHB_HAS_ARBIF parameter), this test verifies the read/write and read only register functionality of the interface. When DW_ahb does not include an AHB slave interface, the test is not executed.

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.

The following checks are performed:

- Default values after reset on all registers.
- Read/write register access.
- Response to unused registers (non existent locations within the memory map).
- Response to unused locations (those beyond the memory map limit of the slave interface).
- When enabled (according to EBTEN parameter) Early burst termination (EBT) functionality. Reset on read of the EBT status register.

8.1.6 Multiplexing

This test verifies the correct functionality of the DW_ahb's multiplexers. The current bus owner index, hmaster, controls the multiplexing of all the address and control signals. Write data control is accomplished by an hready delayed version of hmaster. The read back data, the transfer response, and the slave response are controlled by an hready delayed version of hsel.

The test verifies the following three distinct multiplexing operations:

- Address multiplexers (haddr, hburst, hprot, hsize, htrans, hwrite)
- Write data multiplexers (hwdata)
- Read back multiplexers (hrdata, hready_resp, hresp)

8.1.7 Granting

This is the basic arbitration function of DW_ahb. These tests are used to prove that the arbiter correctly grants competing requesting masters according to priority rules and AMBA protocol rules.

In the first part of the test there is no competition. Each of the master present in the system in turn requests the bus, gets granted, performs a number of transfers and relinquish the bus. This test is used to prove that each master can gain access to the bus, performing successfully a write and then a read operation.

The following tests are dedicated to arbitration when there is competition to access the bus. Multiple masters request the bus at the same time or in a staggered fashion.

The following checks are performed:

- Write-Read data consistency: each master tries to access some dedicated locations of the same slave. After write operations (with data values randomly generated by the master) reads are performed and the master BFM checks for read data values mismatch.
- Each master eventually gain access to the bus.
- The bus is always granted to the highest priority requesting master.

The test is run for a number of different bus scenarios obtained combining different burst types (SINGLE, INCR, WRAP4), number of BUSY cycles inserted during burst sequential transfers, different slave responses (OKAY ERROR RETRY SPLIT) and wait states (zero or more).

If no split capable slave is retrieved from the configuration of DW_ahb, SPLIT responses are not exercised. If a split capable slave is present in the system that slave is selected for the test. For DW_ahb configurations with AHB_LITE set only the first part of the test is executed.

8.1.8 Masking

These tests are used to prove that masters can be excluded from the arbitration scheme.

The tests are as follows:

- Verify that setting the priority value to zero disables a master.
- Verify that self disable is not possible.
- Verify that after SPLIT a master loses the bus.
- Verify the masking of lower priority masters after RETRY.

The test is not executed when DW_ahb is configured with the AHB_LITE parameter set.

8.1.9 Locking

This is the lock arbitration function of DW_ahb. Test stimuli are similar to those of the granting test, the only exception is that in this case some of the masters are programmed for locked transfers, and locked transfers are competing with non-locked transfers. Test checks are specific to the lock functionality.

The following checks are performed:

- The locked transfer completes before any other non-dummy master is granted the bus. The arbiter does not giver ownership to the bus if the last transfer of the locked sequence receives RETRY/SPLIT response.
- HMASTLOCK signal behavior: when asserted HMASTER must not change.
- Write-Read data consistency
- If a locked transfer is split the dummy master must own the bus until the split is cleared.
- The locked transfer completes before any other non-dummy master is granted the bus.

8.1.10 Token Arbitration

This is the weighted token arbitration function of DW_ahb. This functionality is optional and the corresponding test is executed only when AHB_WTEN is set in the configuration. The test starts programming the AHB_WTEN register to enable the token arbitration mode. The stimuli consist of all masters requesting the bus at the same time for a long (longer than the number of tokens available for the arbitration period) transfer sequence.

The following checks are performed:

- The weighted token (wt) arbitration period length is constant (in terms of clock cycles or bus cycles, depending on the configuration of AHB_TPS_MODE) and always matches the configured value of AHB_TCL.
- During each arbitration period, for each master, the weighted token mask is asserted as soon as all the tokens are consumed (value of AHB_CL_M(j)). If other unmasked masters are requesting the access to the bus the current master must be removed from the bus and the highest priority requesting master should get the bus. The mask stays asserted until the start of the next arbitration period.
- At any hmaster change, verify that the corresponding bus granting is in accordance with the following rules:
  a. Normal priority rules if no master or all the requesting masters are (wt) masked.
  b. Bus granting in favor of a lower priority master if an higher priority one gets masked.
  c. Masters with a token count limit of 0 are treated as if configured for an infinite number of tokens.
- At any token mask change verify that the hgrant value matches the expected one according the following rules:
  a. If the current master is the only one requesting the bus it must stay on the bus.
  b. If there are pending-unmasked requests the current master must be removed from the bus.

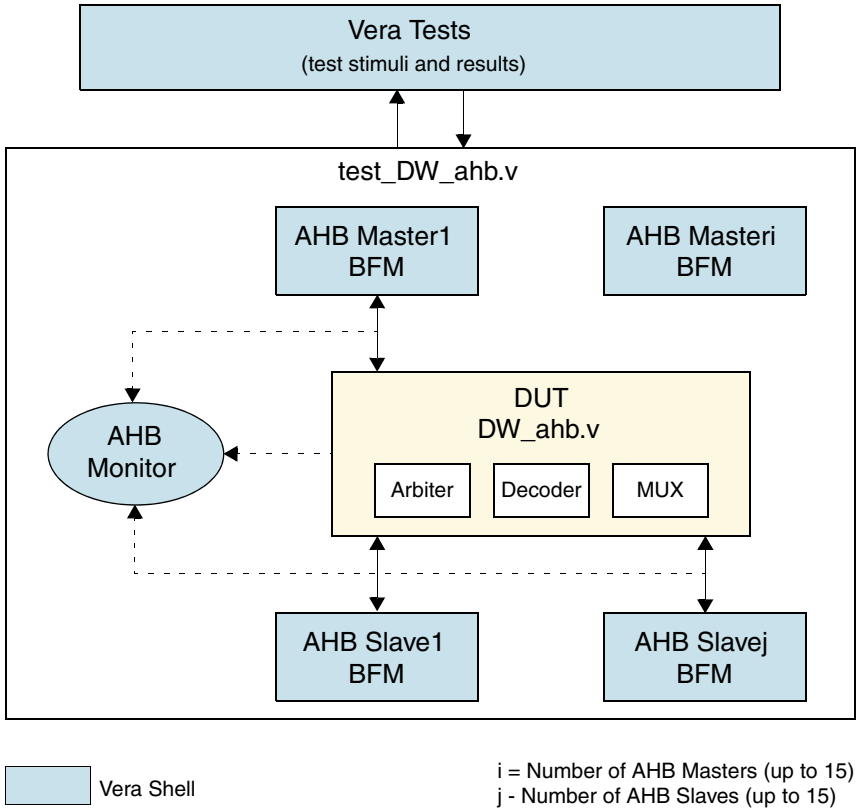
A concise description of the tests can be found in the test.log transcript file generated during simulations.

The AHB monitor is always active during all the tests. Bus signals are continuously monitored and checked against AMBA protocol rules.

8.2 DW_ahb Testbench

As illustrated in Figure 8-1, the DW_ahb testbench is a Verilog testbench that includes an instantiation of the design under test (DUT) and a Vera shell.

Figure 8-1 DW_ahb Testbench



The Vera shell consists of AHB master bus functional models (BFMs), AHB slave BFMs, an AHB monitor, test stimuli, BFM configuration, BFMs and DUT.

The testbench tests for all possible user configurations specified in the Configure Component activity of coreConsultant. The testbench also tests that the component is AMBA-compliant and includes a self-checking mechanism.

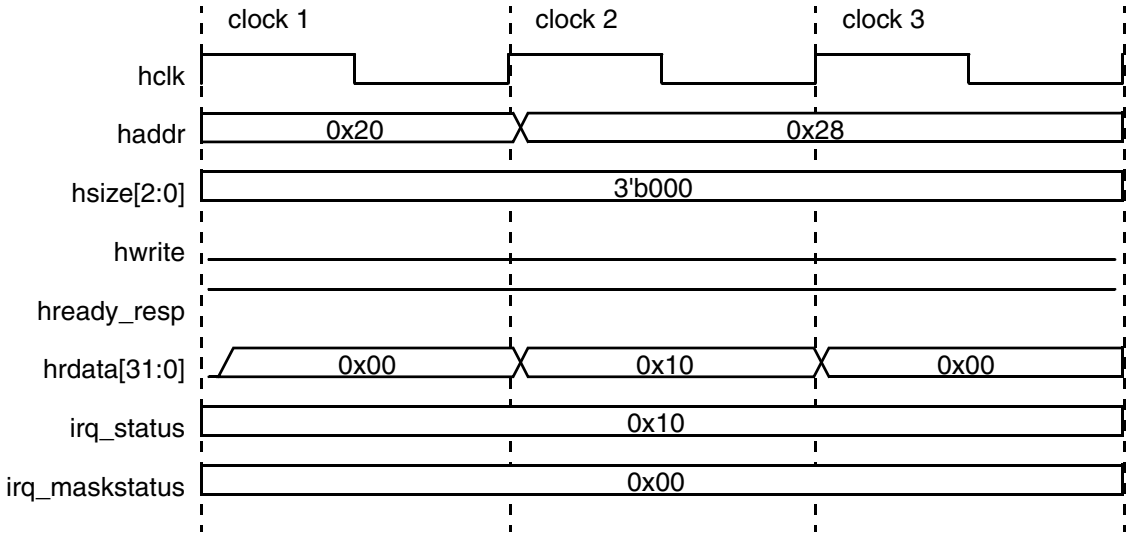
9 Integration Considerations

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

9.1 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 9-1 AHB Read

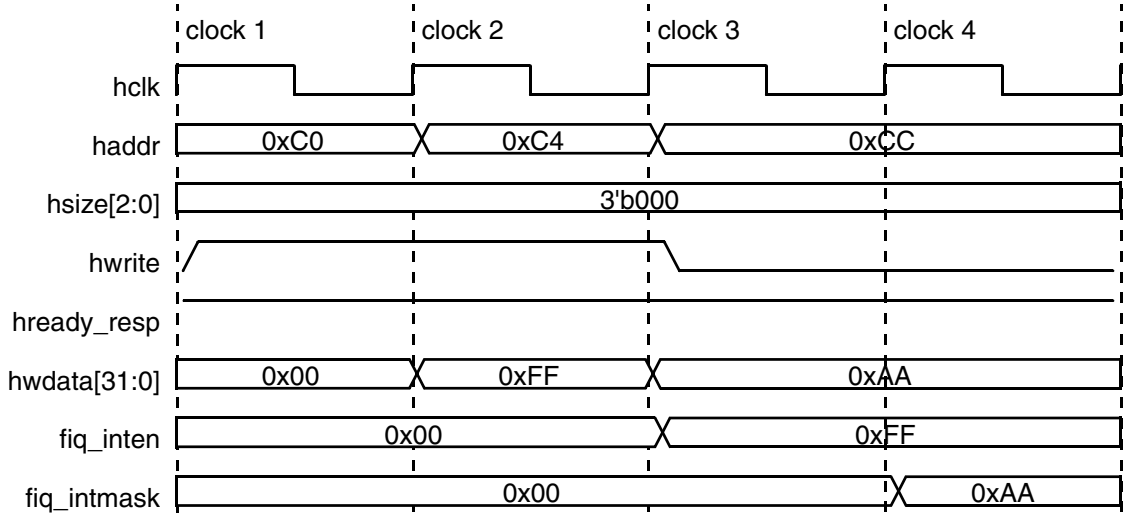


9.2 Write Accesses

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 9-2 AHB Write

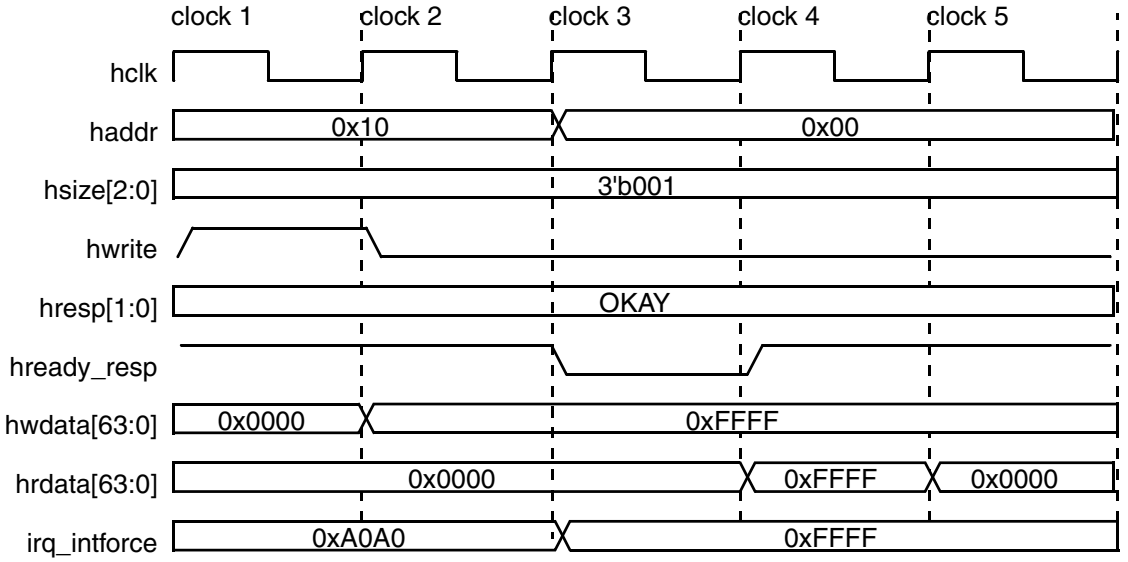


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

9.3 Consecutive Write-Read

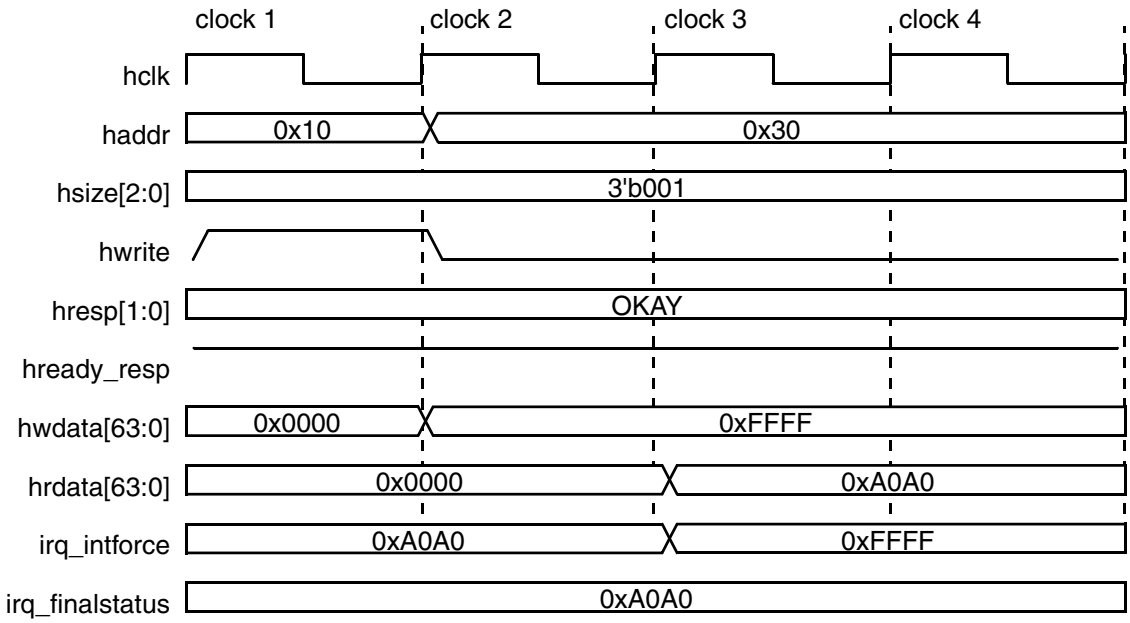
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_ictl inserts a “wait state” when it detects a read immediately 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 9-3 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 9-4 AHB Consecutive Read/Write



9.4 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>
   ```

9.5 Performance

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

9.5.1 Area

This section provides information to help you configure area for your configuration.

The following table includes synthesis results that have been generated using the TSMC 65nm technology library.

Table 9-1 Synthesis Results Using TSMC 65nm Technology Library

Configuration	Operating Frequency	Gate Count
Default Configuration	166.67 MHz	1777 gates
Minimum Configuration: NUM_AHB_MASTERS=1 NUM_IAHB_SLAVES=1 REMAP=0 AHB_DATA_WIDTH=8 AHB_LITE=1	166.67 MHz	64 gates
Maximum Configuration: NUM_AHB_MASTERS=15 NUM_IAHB_SLAVES=15 AHB_DATA_WIDTH=256	166.67 MHz	15933 gates

The following table includes synthesis results that have been generated using the TSMC 28nm technology library.

Table 9-2 Synthesis Results Using TSMC 28nm Technology Library

Configuration	Operating Frequency	Gate Count
Default Configuration	166.67 MHz	1735 gates
Minimum Configuration: NUM_AHB_MASTERS=1 NUM_IAHB_SLAVES=1 REMAP=0 AHB_DATA_WIDTH=8 AHB_LITE=1	166.67 MHz	66 gates
Maximum Configuration: NUM_AHB_MASTERS=15 NUM_IAHB_SLAVES=15 AHB_DATA_WIDTH=256	166.67 MHz	14439 gates

9.5.2 Power Consumption

The following table provides information about the power consumption of the DW_ahb using the TSMC 65nm technology library and how it affects performance.

Table 9-3 Power Consumption for DW_ahb Using TSMC 65nm Technology Library

Configuration	Operating Frequency	Static Power Consumption	Dynamic Power Consumption
Default Configuration	166.67 MHz	469.0464 nW	173.8591 µW
Minimum Configuration: NUM_AHB_MASTERS=1 NUM_IAHB_SLAVES=1 REMAP=0 AHB_DATA_WIDTH=8 AHB_LITE=1	166.67 MHz	18.1558 nW	8.3473 µW
Maximum Configuration: NUM_AHB_MASTERS=15 NUM_IAHB_SLAVES=15 AHB_DATA_WIDTH=256	166.67 MHz	5.9192 µW	555.9502 µW

The following table provides information about the power consumption of the DW_ahb using the TSMC 28nm technology library and how it affects performance.

Table 9-4 Power Consumption for DW_ahb Using TSMC 28nm Technology Library

Configuration	Operating Frequency	Static Power Consumption	Dynamic Power Consumption
Default Configuration	166.67 MHz	183.3421 µW	117.2563 µW
Minimum Configuration: NUM_AHB_MASTERS=1 NUM_IAHB_SLAVES=1 REMAP=0 AHB_DATA_WIDTH=8 AHB_LITE=1	166.67 MHz	6.3821 µW	5.2355 µW
Maximum Configuration: NUM_AHB_MASTERS=15 NUM_IAHB_SLAVES=15 AHB_DATA_WIDTH=256	166.67 MHz	1.5592 mW	369.3810 µW

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

Table A-1 DesignWare Synthesizable Components Constant Definitions (Continued)

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

Table A-1 DesignWare Synthesizable Components Constant Definitions (Continued)

DesignWare Synthesizable Components Constant	Value
KBYTE	1024

B
Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
AHB	Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (ARM Limited specification).
AMBA	Advanced Microcontroller Bus Architecture — a trademarked name by ARM Limited that defines an on-chip communication standard for high speed microcontrollers.
APB	Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (ARM Limited specification).
APB bridge	DW_apb submodule that converts protocol between the AHB bus and APB bus.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
arbiter	AMBA bus submodule that arbitrates bus activity between masters and slaves.
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocking command	A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.

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

DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWare Synthesizable Components.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.

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

Index

D