SYNOPSYS®

DesignWare DW_ahb Databook

Component of the DesignWare AMBA On-Chip Bus

Version 2.02a

June 21, 2004

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, COSSAP, CSim, DelayMill, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSPICE, Hypermodel, I, iN-Phase, InSpecs, in-Sync, Leda, MAST, Meta, Meta-Software, ModelAccess, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SiVL, SmartLogic, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAII, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert *Plus*, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA *Express*, Frame Compiler, Galaxy, Gatran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, LRC, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLlint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, Progen, Prospector, Proteus OPC, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-OPC, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
AMBA is a trademark of ARM Limited. ARM is a registered trademark of ARM Limited.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

About This Manual

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.

Related Documents

To see a complete listing of documentation related to the DW_ahb within the DesignWare AMBA On-chip Bus platform, refer to the *Guide to DesignWare AMBA OCB Documentation*.

Manual Overview

This manual contains the following chapters and appendixes:

Chapter 1 “Product Overview”	Provides a DesignWare AMBA OCB System Overview, a component block diagram, basic features, and an overview of the verification environment.
Chapter 2 “Building and Verifying Your Core”	Provides getting started information that allows you to walk through the process of using the DW_ahb.
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.

Appendix A “Database Description”	Provides deliverables and reference files generated from the coreConsultant flow.
Appendix B “DesignWare AMBA Connect and QuickStart Designs”	Provides the locations of QuickStart examples that integrate most DesignWare AMBA OCB components into an SoC design that you can simulate.
Appendix C “DesignWare AMBA OCB Constants”	Includes the contents of the DesignWare AMBA OCB bus constants file.
Appendix D “Glossary”	Provides a glossary of general terms.

Typographical and Symbol Conventions

The following conventions are used throughout this document:

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
Monospace	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or *Italic*	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: `pin1 [pin2 ... pinN]` In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > SubMenu	Pulldown menu paths, such as: **File > Save As …**

Getting Help

If you have a question about using Synopsys products, please consult product documentation that is installed on your network or located at the root level of your Synopsys product CD-ROM (if available). You can also access documentation for DesignWare products on the Web:

- Product documentation for many DesignWare products:

  http://www.synopsys.com/products/designware/docs

- Datasheets for individual DesignWare IP components:

  http://www.synopsys.com/products/designware/ipdir

You can also contact a Synopsys Support Center online or by phone:

- http://www.synopsys.com/support/support.html
- United States:
  Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific Time, Mon—Fri.
- Canada:
  Call 1-650-584-4200 from 7 AM to 5:30 PM Pacific Time, Mon—Fri.
- All other countries:
  Find other local support center telephone numbers at the following URL:

  http://www.synopsys.com/support/support_ctr

Additional Information

General information about Synopsys and its products is available at this URL:

http://www.synopsys.com

Comments?

To report errors or make suggestions, please send e-mail to:

doc@synopsys.com

To report an error that occurs on a specific page, select the entire page (including headers and footers), and copy to the buffer. Then paste the buffer to the body of your e-mail message. This will provide us with information to identify the source of the problem.

1 Product Overview

The DesignWare DW_ahb is a programmable Advanced High-performance Bus.

This chapter provides a basic overview of the DW_ahb. The topics included in this chapter are:

DesignWare AMBA OCB System Overview

The Synopsys DesignWare AMBA On-chip Bus environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB and APB components.

DesignWare AMBA OCB System Block Diagram

Figure 1 on page 10 illustrates one example of this environment, including the AHB bus, the APB Bus (includes the APB Bridge), AHB multi-layer interconnect IP, APB peripheral components, verification Master/Slave models, and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in Figure 1.

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.

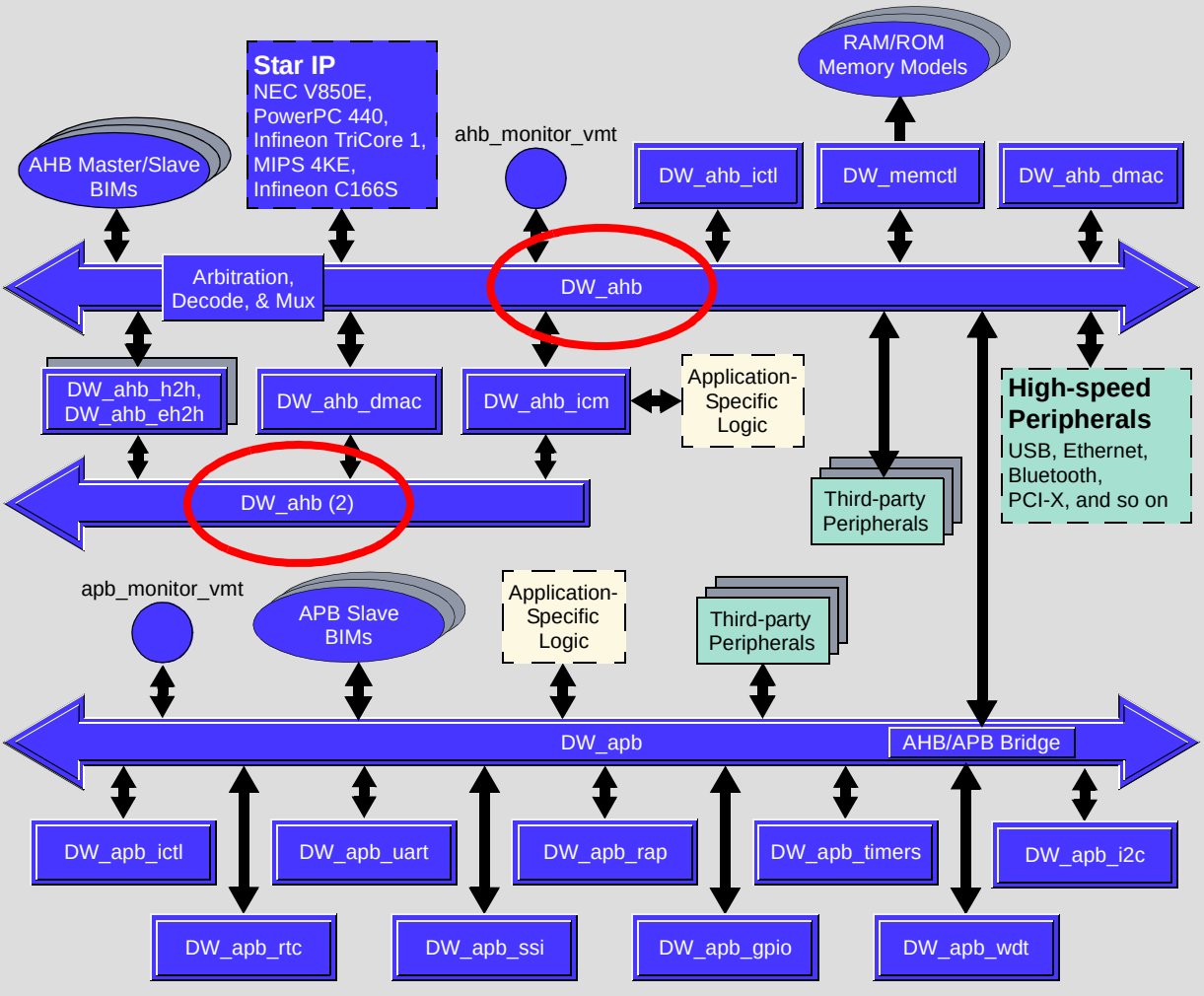


Figure 1: Example of DW_ahb in a Complete System

General Product Description

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

DW_ahb Block Diagram

As shown in Figure 2 on page 12, the DW_ahb consists of the following components:

- "Arbiter" on page 27 – 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 35 – 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 38 – 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 38 – All addresses, data, and control signals from each master are multiplexed.

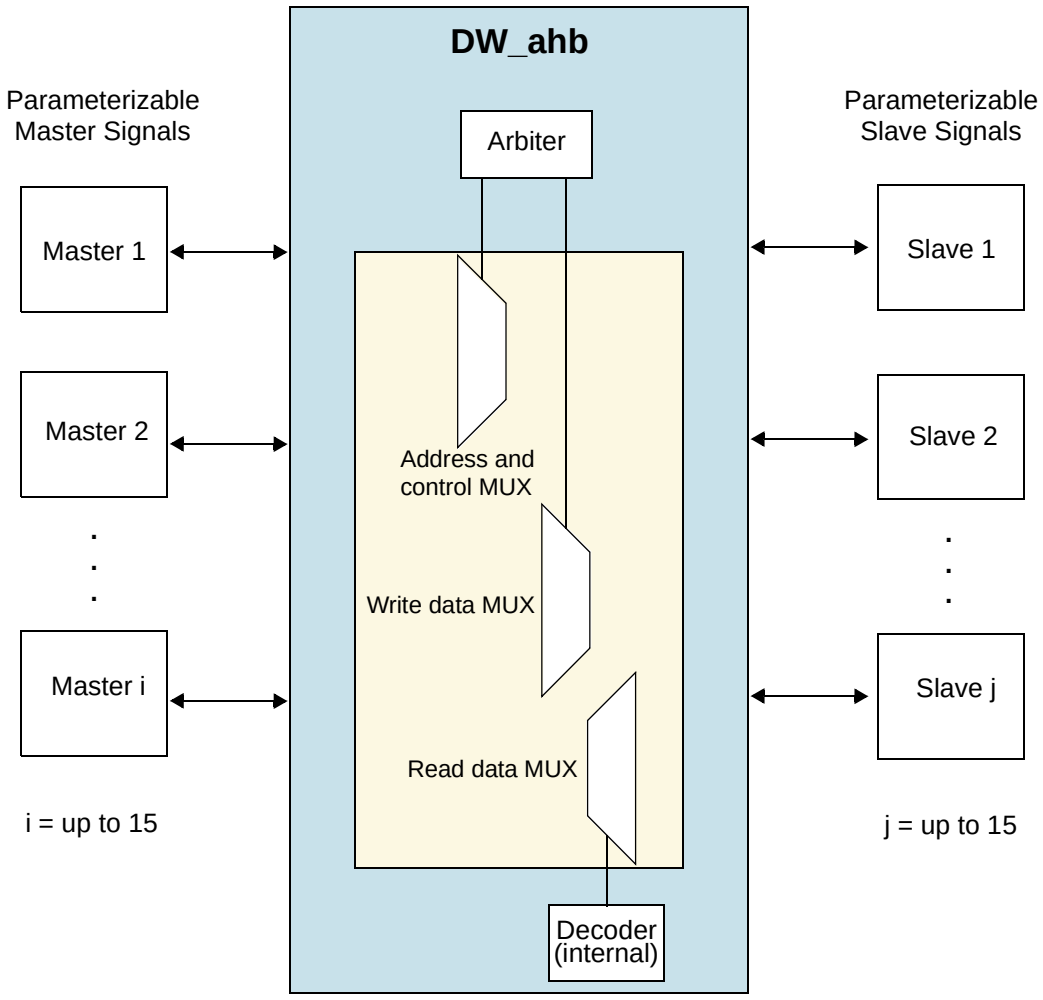


Figure 2: DW_ahb Block Diagram

Features

The DW_ahb supports the following features:

- Compliance with the *AMBA Specification (Rev. 2.0)*
- Configuration of DesignWare AMBA Lite system
- Configuration of up to 15 masters in a DesignWare non-AMBA 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 DesignWare AMBA OCB 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.

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.

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 87 chapter discusses the specific procedures for verifying the DW_ahb.

2 Building and Verifying Your Core

This chapter provides an overview of the step-by-step process you use to configure, synthesize, and verify your DW_ahb component. The topics are as follows:

Setting up Your Environment

Refer to the section "Setting up Your Environment" in the *DesignWare DW_ahb Release Notes.*

Starting coreConsultant

To start the coreConsultant GUI:

1. In a UNIX shell, navigate to a directory where you plan to locate your component Workspace.
2. Invoke coreConsultant:

   ```
   % coreConsultant
   ```

   The DesignWare AMBA On-Chip Bus welcome page is displayed. At this point, you can begin configuring your component or open an exiting workspace.

Checking Your Environment

Before you begin configuring your component, it is recommended that you check your environment to make sure you have the latest tool versions installed and your environment variables set up correctly.

To check your environment, open or create a workspace and select **Help > Check Environment**.

An HTML report is displayed in a separate dialog. This report lists the specific tools and versions installed in your environment. It also displays errors when a specific tool is not installed or if you are using an older version. You will also see an error if your $DESIGNWARE_HOME environment variable has not been set up correctly.

Configuring the DW_ahb

This section steps you through the tasks in the coreConsultant GUI that configure your core. Complete information about the latest version of coreConsultant is available on the web in the *coreConsultant User Guide*. To view documentation specific to your version of coreConsultant, choose the Help pulldown menu from the coreConsultant GUI.

At any time during this process you can click on the Help tab for each activity to activate the coreConsultant online help.

1. In the Getting Started window, click on DW_ahb.

   In the resulting dialog box, specify the Workspace name and root directory, or use the defaults.

2. Under the Create RTL category in the Activity List, the Specify Configuration activity is where you specify the basic configuration of the DW_ahb.

   If you need help with any field in the activity pane, right-click on the field name and then left-click on the What's This box, or you can click on the Help tab to see a listing of all the field descriptions. When you are finished configuring the basic parameters for your core, click the **Apply** button.

   The Report tab lists useful information as you complete each step in the coreConsultant process.

For more information about the specific configuration parameters, refer to "Parameter Descriptions" on page 43.

Synthesizing the DW_ahb

This section provides synthesis information for the DW_ahb component.

The topics are as follows:

Create Gate-Level Netlist

To run synthesis on the DW_ahb to create a gate-level netlist, step through the following tasks in the coreConsultant GUI. At any time, you can click on the **Help** tab for each activity to activate the coreConsultant online help.

1. Specify the tool installation root directories in the Tool Installation Roots dialog which is accessed from the toolbar menu through **Edit > Tool Installation Roots**. Type values directly in the data fields. At a minimum, dc_shell and fc2_shell must have an installation directory defined.
2. Under the "Create Gate-Level Netlist" activity, the Specify Target Technology item displays the Library setup pane where you must specify a target and link library; otherwise, errors occur in coreConsultant. When finished, click Apply.
3. Specify the attributes associated with each of the real and virtual clocks in your design by completing the Specify Clock(s) activity. Click Apply.
4. Specify the attributes relating to the chip environment, by completing the Specify Operating Conditions and Wire Loads activity. Click Apply.
5. Specify the attributes associated with input delay, drive strength, and DRC constraints by completing the Specify Port Intent activity. Click Apply.

6. Specify the common synthesis strategy attributes by completing the Specify Synthesis Methodology activity. Note that these attributes are typically set by the core developer and are not required to be modified by the core integrator. Click Apply.
7. Select the Synthesize activity. Do the following:
   a. Select the Strategy tab.
   b. Turn on the Enable TCL-based implementation tools.
   c. In the Strategy field, select “DCTCL_opto_strategy.”
   d. Select the Options tab.
   e. Set the Run Style option to “local” and maintain the other default settings.
   f. Click Apply in the Synthesize activity pane to start synthesis from coreConsultant. The current status of the synthesis run is displayed in the main window. Click the refresh button to update the status in this screen.

All the synthesis results and log files are created under the syn directory under your workspace. Refer to “Synthesis Files” on page 95.

The top-level script that is executed in the synthesis process is either initial.dcsh, incr1.dcsh, or incr2.dcsh and resides in its respective directory. This top-level script analyzes and elaborates all the RTL files and then sources the top-level constraint file, DW_ahb.cscr, in *workspace*/syn/constrain/ script. The top-level script (.dcsh) and all of the constraint files (*.cscr) are unencrypted.

Running Synthesis from Command Line

To run synthesis from the command line prompt for the files generated by coreConsultant, enter the following command:

```
% run.scr
```

This script resides in your *workspace*/syn directory.

Verifying the DW_ahb

This section provides the steps you use to execute the testbench available for DW_ahb verification. Once the DW_ahb has been configured and the verification environment has been set up, simulations can be automatically run. DW_ahb verification is detailed in the following sections:

- “Creating GTECH Simulation Models”
- “Verify the Simulation Model” on page 19
- “ACT Certification” on page 23

Note

For GTECH Simulations Only. Due to the configurable nature of the component, some ports in the testbench may not be needed for your chosen configuration. Warnings about undriven nets may appear. These warnings are to be expected, and you can ignore them. The verification result files show if the verification ran successfully.

Creating GTECH Simulation Models

DesignWare AMBA On-chip Bus implementation IP (coreKit RTL) is delivered in encrypted format, rather than source code, and some simulators cannot read the encrypted source files. In order for these simulators to have a simulation model of the component, you must perform a conversion to gates, either GTECH gates or technology-specific gates. However, 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.

Note

The Synopsys VCS simulator reads the encrypted files directly and does not require a GTECH conversion. All other supported simulators require a GTECH simulation model. You need a DesignWare license to complete the gate-level netlist generation process.

1. To create a GTECH simulation model, click on the “Generate GTECH Model” activity.
2. Specify values for the fields listed below.

Field Name	Description
Execution Options	
Generate Scripts only?	**Values**: Enable or Disable **Default Value**: Disable **Description**: Writes scripts that run the generation of the GTECH simulation model, but they are not run when you click Apply. To run these scripts, go to the gtech directory of the component workspace and run the run.scr script.
Run Style	**Values**: local, lsf, grd, or remote **Default Value**: local **Description**: Describes how to run the command: locally, through lsf, through grd, or through the remote shell.
Run Style Options	**Values**: user-defined **Default Value**: none **Description**: Additional options for the run style options except local. For remote, specify the hostname. For lsf and grd, specify bsub or qsub commands.
Send e-mail	**Values**: current user’s name **Description**: E-mail is sent when the command script completes or is terminated.
Synthesis Control	
Ungroup Netlist after Compile	**Values**: Enable or Disable **Default Value**: Disable **Description**: Ungroups the design to provide a non-hierarchical netlist

3. Specify your parameters, then click Apply; coreConsultant invokes Design Compiler to perform a low-effort compile (quickmap) of your custom configuration using the Synopsys technology-independent GTECH library. After this activity has completed, an e-mail similar to the following is sent to the specified user name (if selected):

```
Activity:   GenerateGtechModel
Workspace:  workspace path
Design:     DW_ahb
Started:    Wed Jul 24 16:19:48 BST 2002
Finished:   Wed Jul 24 16:21:42 BST 2002
Status:     Completed
Results:    workspace path/gtech/gtech.log
```

Your simulation model is contained in the output file DW_ahb.v that is written to *workspace*/gtech/qmap/db.

Verify the Simulation Model

To verify the DW_ahb, complete the following steps using coreConsultant.

1. **Optional**. You can run formal verification scripts using Synopsys' Formality (fm_shell) to check two designs for functional equivalence. You can check the gate-level design from a selected phase of a previously executed synthesis strategy against either the RTL version of the design or the gate-level design from another stage of synthesis.

   To run this, select Formal Verification under the Verify Component activity. Maintain the default parameter settings and click Apply.

2. Specify the simulation by completing the Setup and Run Simulations activity:

   a. Select the Simulator view list item to view the Simulator pane.

   b. In the Select Simulator area, in the "Simulator" field you can specify one of the supported Verilog simulators from the drop-down.

      For installation instructions, required tools and versions, and general information about the contents of the release, refer to the *DesignWare DW_ahb Release Notes*.

   c. In the Simulator Setup area of the Simulator pane, specify the parameters for the simulator setup as detailed below.

      Specify the parameters for the simulator setup as detailed below.

Field Name	Description
Root Directory of Cadence Installation	The path to the top of the directory tree where the Cadence NC-Verilog executable is found; coreConsultant automatically detects this path. The NC-Verilog executables reside in the ./bin subdirectory.
MTI Include Directory	The path to the include directory contained within your MTI simulator installation area. A valid directory includes the file veriuser.h.
Vera Install Area ($VERA_HOME)	Path to your Vera installation. This parameter defaults to the value of your VERA_HOME environment variable. Changes to this value are propagated as $VERA_HOME in any simulation run.

Field Name	Description
Vera .vro file cache directory	Cache directory used by Vera to store .vro files, which are generated when building the testbench. Encrypted Vera source is compiled and stored in the cache. This is placed within a scratch location.
DW Foundation install area	Path to your Synopsys/DW Foundation installation. This parameter defaults to the value of your SYNOPSYS environment variable. Any change to this value must be made from the Tool Installation Areas coreConsultant dialog box.
C Compiler for (Vera PLI)	**Values**: gcc or cc **Default Value**: gcc **Description**: Invokes the specific C compiler to create a Vera PLI for your chosen non-VCS simulator. Choose cc if you have the platform native ANSI C compiler installed. Choose gcc if you have the GNU C compiler installed.

d. In the Waves Setup area of the Simulator pane, specify the parameters for the waves setup as detailed below.

Field Name	Description
Generates waves file	**Values**: Enable or Disabled **Default Value**: Disabled **Description**: Indicates whether a wave file should be created for debugging with a wave file browser after simulation ends. Uses VPD file format for VCS and VCD format for the other supported simulators.
Depth of waves to be recorded	**Description**: Enter the depth of the signal hierarchy for which to record waves in the dump file. A depth of 0 indicates all signals in the hierarchy are included in the wave file.

e. Select the View list choice.

f. In the View Selection area of the View pane, select the View of the design you want to simulate from the drop-down list. Valid views include:

- RTL – requires a source license or Synopsys VCS
- GTECH – requires that you have completed the Generate GTECH Model activity (see “Creating GTECH Simulation Models” on page 18).

g. Choose the Execution Options and then choose or enter values according to the choices listed below.

Field Name	Description
Do Not Launch Simulation	**Values**: Enable or Disable **Default Value**: Disable **Description**: Determines whether to execute the simulation or just generate the simulation run script. If checked, coreConsultant generates, but does not execute, the simulation run script. You can execute the script at a later time by invoking the run script (workspace/sim/run.scr) directly from the UNIX command line or by repeating the Verification activity with Do Not Launch Simulation unselected.
Run Style	**Values**: local, lsf, grd, or remote **Default Value**: local **Description**: Describes how to run the command: locally, through lsf, through grd, or through the remote shell.
Run Style Options	**Values**: user-defined **Default Value**: none **Description**: Additional options for the run style options except local. For remote, specify the hostname. For lsf and grd, specify bsub or qsub commands.
Send e-mail	**Values**: current user's name **Description**: E-mail is sent when the command script completes or is terminated.

h. Select Testbench and specify the following options:

Field Name	Description
Type of Monitoring	**Values**: Enable AHB Monitor, Enable AHB ACT monitor certification, or Enable AHB ACT monitor recertification **Default Value**: Enable AHB Monitor **Description**: Select the type of monitor to include: Enable AHB Monitor (No ACT checking) Enable AHB ACT monitor certification Enable AHB ACT monitor recertification Unless the AHB data bus width is 32, 64, or 128 bits, and Arbiter Slave Interface is included, you are forced to use the AHB monitor.
Run test_10_token	This runs the specific tests to verify the functionality of the weighted token arbitration.
Run test_11_slave_act	This runs the ACT tests. To run ACT or ACT recertification, you must chose the appropriate monitor in the Type of Monitoring drop-down box. For more information about ACT certification and recertification, refer to "ACT Certification" on page 23 and "ACT Recertification" on page 24, respectively.
Run test_2_token	This runs the specific tests to verify the DW_ahb decoder.

Field Name	Description
Run test_3_default_slave	This runs the specific tests to verify the functionality of the DW_ahb default slave.
Run test_4_dummy_master	This runs the specific tests to verify the functionality of the DW_ahb dummy master.
Run test_5_granting	This runs the specific tests to verify the arbitration scheme.
Run test_6_arbiter_if	This runs the specific tests to verify the functionality of the arbiter slave interface.
Run test_7_multiplexing	This runs the specific tests to verify the DW_ahb's multiplexers.
Run test_8_masking	This runs the specific tests to verify the masking functionality of the DW_ahb.
Run test_9_locking	This runs the specific tests to verify the locking functionality of the DW_ahb.

i. Click Apply.

coreConsultant then performs the following actions:

- Sets up the DW_ahb verification environment to match your selected DW_ahb configuration.
- Generates the simulation run script (run.scr) and writes it to your workspace/sim directory.
- Invokes the simulation run script, unless you enabled the Do Not Launch Simulation option.

The simulation run script, in turn, performs the following actions:

- Invokes your simulator to simulate the specified scenarios.
- Writes the simulation output files to your workspace/sim/test_*name* directory.
- If an e-mail address is specified, sends the simulation completion information to that e-mail address when the simulation is complete.

If you have chosen the "LSF/GRD" option for the Run Style, then the status of the simulation jobs (running or complete) is not correct. Once all the simulation jobs are submitted to the LSF/GRD queue, the status would indicate "complete." You should use "bjobs/qstatus" to see whether all the jobs are completed.

For each test executed, the run script first writes simulation output files to workspace/sim/test_*name*, then overwrites the files with the results of each succeeding scenario. The files that remain in workspace/sim/test_*name* after all the scenarios are complete are the results of the last scenario to be executed.

Checking Simulation Status and Results

To check simulation status and results, click the Report tab for either the GTECH models or for the simulation options; coreConsultant displays a dialog that indicates:

- Your selected Run Style (local, lsf, grd, or remote)
- The full path to the HTML file that contains your simulation results
- The name of the host on which the simulation is running
- The process ID (Job Id) of the simulation

- The status of the simulation job (running or completed)

The Results dialog also enables you to kill the simulation (Kill Job) and to refresh the status display in the Results dialog (Refresh Status).

The Results information includes:

- Vera compile execution messages
- Simulation execution messages
- DW_ahb bus transactions

This information indicates whether the simulation executed successfully, and lists the DW_ahb transactions that occurred during the scenario(s).

Thorough analysis of the scenario execution requires detailed analysis of all simulation output files and inspection of simulation waveforms with a waveform viewer.

Default Verification Attributes

To reset all DW_ahb verification attributes to their default values, use the Default button in the Setup and Run Simulation activity under the Verification tab. To examine default attributes values without resetting the attribute values in your current workspace, create a new workspace; the new workspace has all the default attributes values. Alternatively, use the Default button to reset the values, and then close your current workspace without saving it.

ACT Certification

You can run DesignWare AMBA Compliance Tool (ACT) certification on the DW_ahb provided you have the required license (DesignWare-ACT-VIP) and the data width of 32, 64, or 128 bits. For more information about DesignWare ACT certification, refer to the DesignWare AMBA Compliance Tool data sheet at:

www.synopsys.com/products/designware/docs/ds/v/ahb_act_monitor_vmt.pdf

If you are interested in purchasing the DesignWare AMBA Compliance Tool verification IP product, please contact your local Synopsys Sales Office.

ACT allows designers to certify the compliance of their design with the AMBA 2.0 specification and to generate a certificate of compliance. The ACT certificate has two parts: plain text and encrypted.

The plain-text portion includes the following information:

- Data and time
- ARM ACT version
- User string
- Configuration information

The encrypted portion contains the following information:

- Data and time
- Information on coverage points hit
- ARM ACT version
- Internal DW ACT version

ACT Recertification

Note

You do not need a DesignWare-ACT-VIP license to run ACT recertification.

ACT recertification allows the designer to exercise the IP again with the same transactions and then generate compliance data. The compliance data is compared with the data in the original ACT certificate. If proper coverage points are not hit or if there is a protocol error during recertification, a differences certificate is generated. The designer can use this differences certificate to send back to Synopsys for problem solution.

The differences certificate has two parts similar to the ACT certificate except that the encrypted portion contains information on coverage points missed or incorrectly hit and includes protocol errors.

Certificates for the default configuration are already included and reside in the following directories:

workspace/sim/default_act_certs/act_slave.certificate

If all the coverage points are not hit then a differences certificate, re_act_slave.certificate, is generated and written to the *workspace*/sim/default_act_certs directory.

Simulation Output Files

The runtest.log file in *workspace*/sim includes all of the results of the simulation and presents them in the following categories:

- Summary of All Results – provides the final result either PASSED or FAILED
- Verification Activity Log – a log of the simulation activity
- Testbench Preparation – a list of runtest options that were executed during the simulation
- Simulation Execution – provides the output of the simulator; this information is also saved to test.log in *workspace*/sim/test_*name*.
- Simulation Results – includes the time the simulation completed, the path to test.log, the number of errors, and the overall result (PASSED/FAILED)

The *workspace*/sim/test_*name* directory includes the various logs that are included in runtest.log. The individual log files in *workspace*/sim/test_*name* are:

- test.log – output of the testbench and includes specifics about the simulators used, the tests uses to verify the core, and the simulation results.
- summary – post-processed file that includes the following sections:
  - Testbench Preparation
  - Simulation Execution
  - Profiling Report
  - Test Report
  - Simulation Results
- test.result – The testbench automatically compares the simulation results with the expected results during simulation. If the simulation results match expected results, the simulation completes successfully and the simulation status in the test.result file is PASSED. If the simulation results do not match expected results, the simulation terminates and the simulation status in the test.result file is FAILED.

- test.vcd or vcdplus.vpd – waveform dump files depending on the specific simulator. The test.vcd file is for simulators other than VCS; vcdplus.vpd is the waveform dump file for VCS, which is only generated when the dump depth is specified.

For more information about deliverables that are generated after verification is performed, see “Database Description” on page 93.

3 Functional Description

This chapter describes the functional operation of the DW_ahb and describes the components and functionality of the DW_ahb:

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 DesignWare AMBA Lite (or AHB Lite), which is a subset of the full AHB design where only a single bus master is used. This 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

DW_ahb Components

This section describes the function of the following DW_ahb components:

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.

The following topics are included in this section:

Arbitration Schemes

If the DW_ahb is specified as a DesignWare AMBA Lite system, then there is no arbitration scheme. There is only one master in a DesignWare AMBA 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 the 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 84.

When two or more masters request the bus at the same time, the requesting master with the highest priority is granted the bus. (First tier arbitration)

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. (Second tier arbitration)

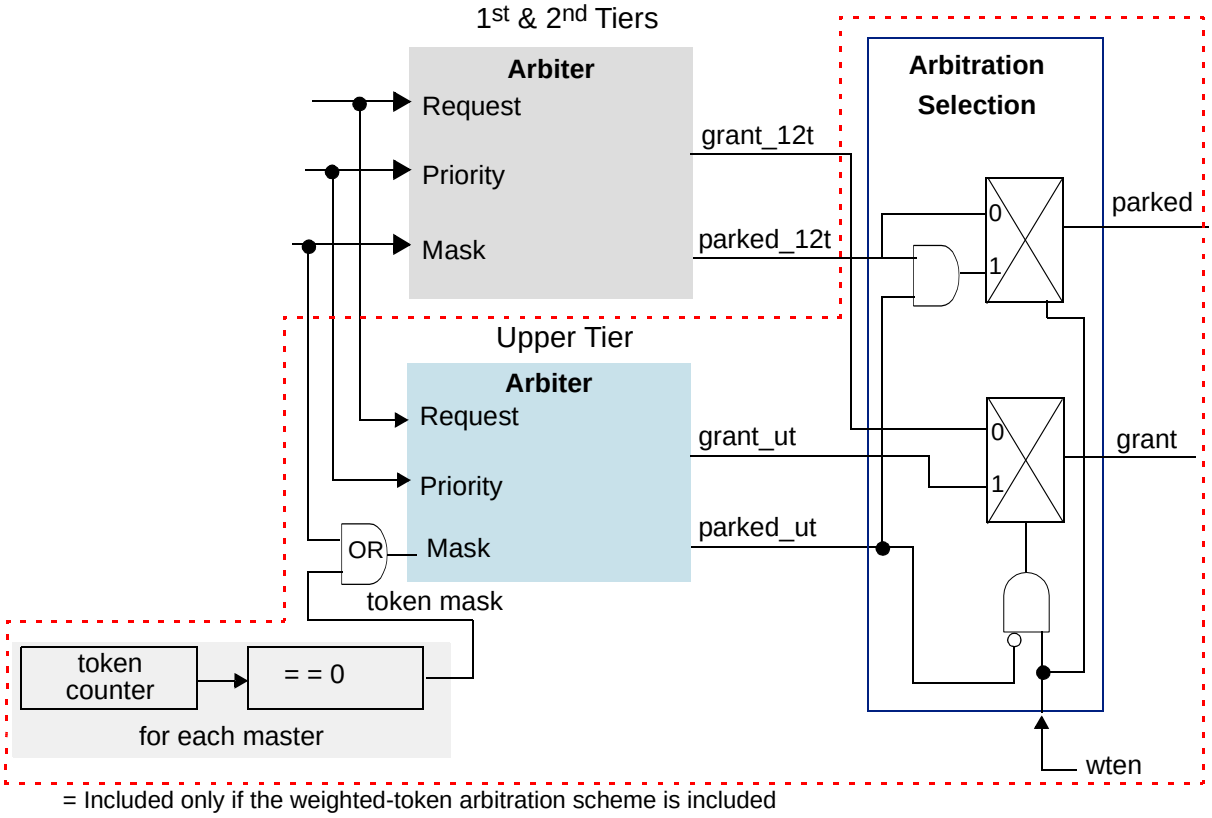
If the DW_ahb is specified with the weighted-token arbitration scheme, then each master in the system is allocated an initial number of clock tokens. This scheme is an extension of the default arbitration scheme (first and second tiers). Additionally, the overall length of the arbitration period needs 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 11 on page 60.

These schemes are illustrated in Figure 3 on page 29.

Note

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



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.

Figure 3: Arbitration Scheme in DW_ahb

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 4 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.

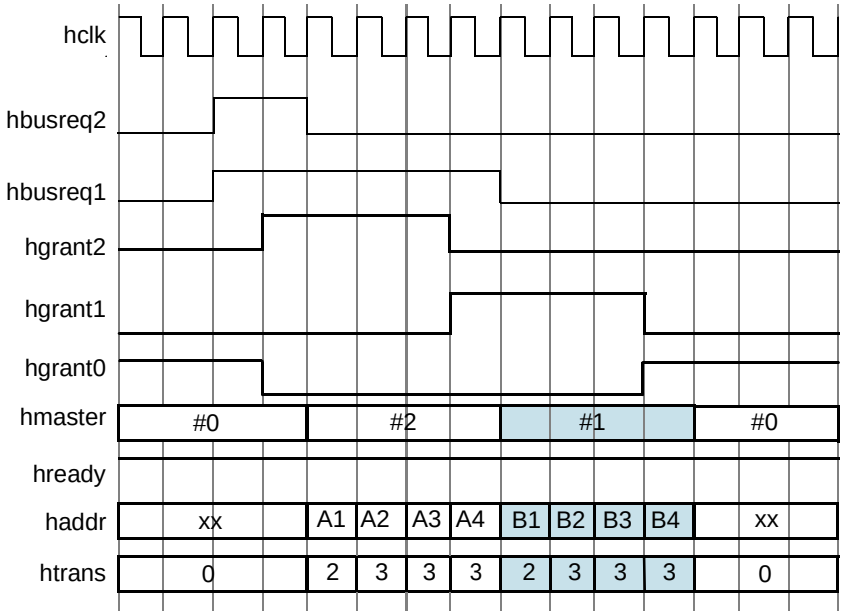


Figure 4: Bursts with Specified Length

Figure 5 on page 30 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.

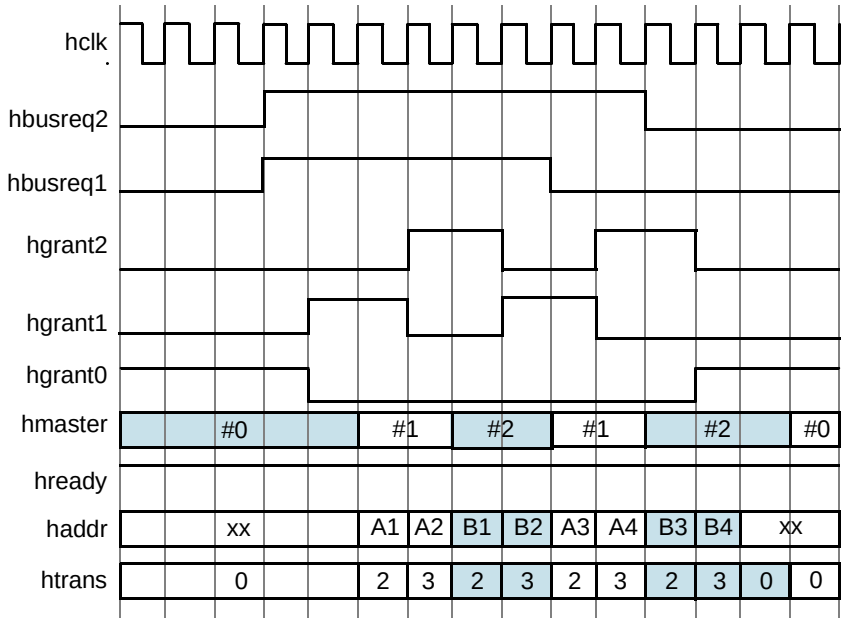


Figure 5: Bursts with Unspecified Length and Equal Priority Level

In Figure 5 on page 30, 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 6 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.

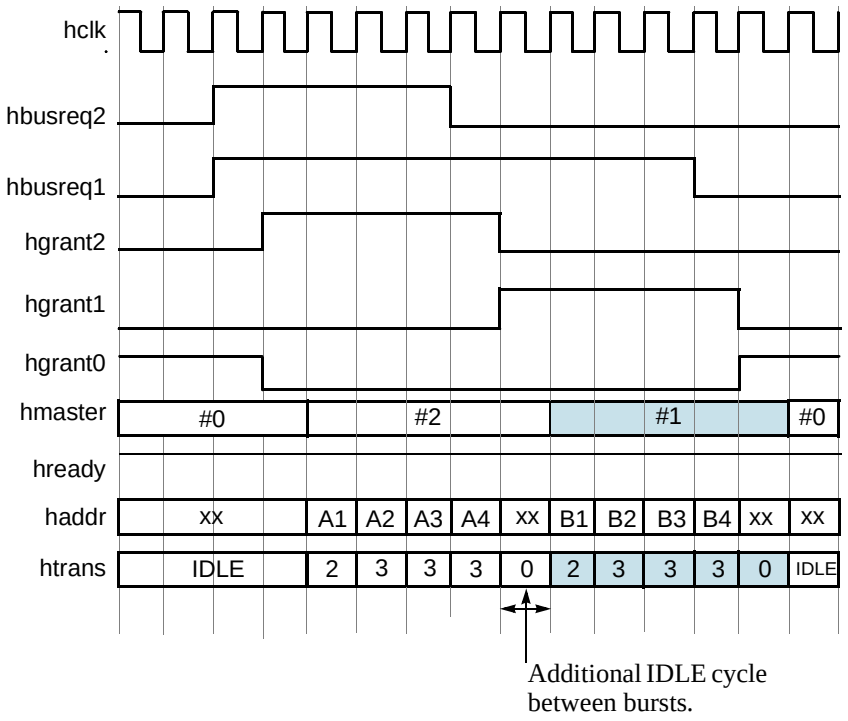


Figure 6: Bursts with Unspecified Length and Different Priority Level

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.

Note

For a DesignWare AMBA 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.

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 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 83.

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 DesignWare AMBA Lite, the arbiter slave interface is not included.

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 DesignWare AMBA Lite, the default master number is changed to 1.

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

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 DesignWare AMBA Lite, pause mode is disabled.

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.

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:

- Back-to-back transfers when a higher priority master requests in the same cycle as when the second NSEQ is on the bus.
- 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 cyles 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 occured.

For more information about programming the early burst termination registers, refer to “Early Burst Termination Registers” on page 85.

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.

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 7 on page 35 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 85). 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.

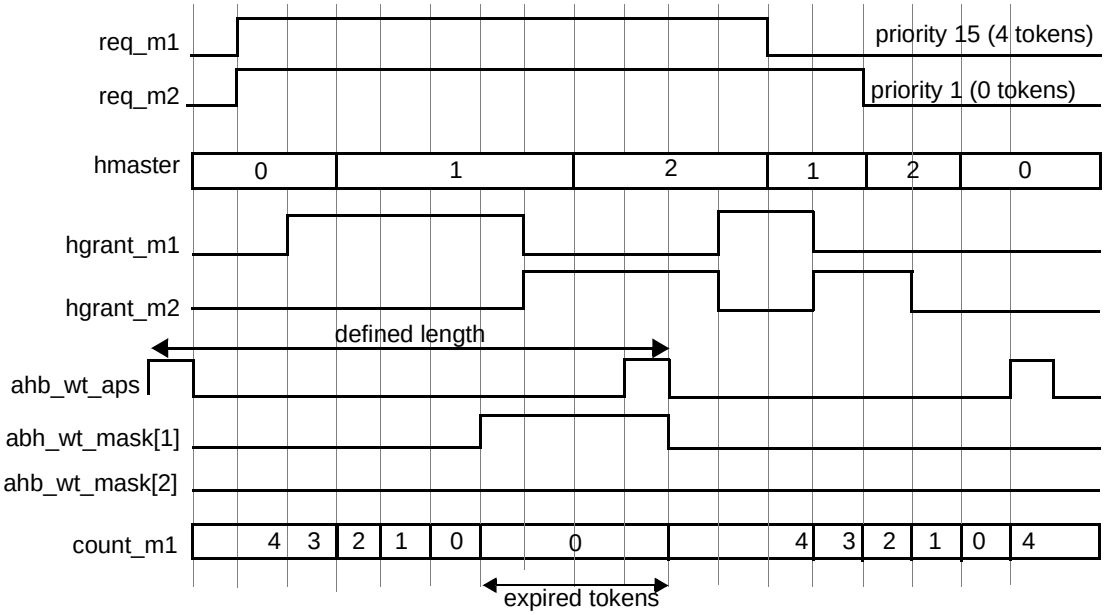


Figure 7: Masters Competing for Bus Ownership with Weighted Tokens

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 38). 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 37
- "Slave Aliasing" on page 37
- "Slave Visibility" on page 37
- "Default Slave" on page 38

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 AMBA 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 8 on page 36 illustrates memory maps for both Boot and Normal modes. 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).

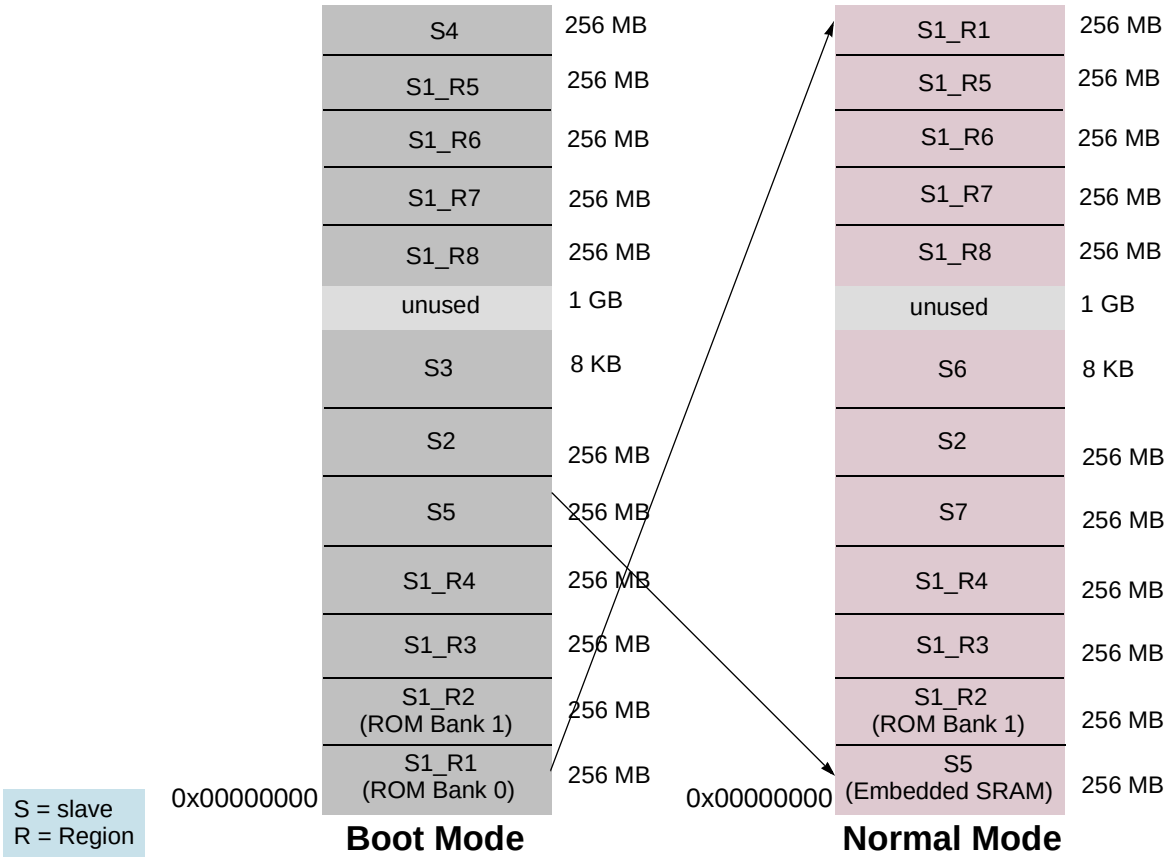


Figure 8: Example DW_ahb Memory Map – Boot Mode and Normal Mode

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 8) 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 8 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.

Slave Aliasing

The DW_ahb also includes an alias feature, shown in Figure 9 on page 37, 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 9. 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.

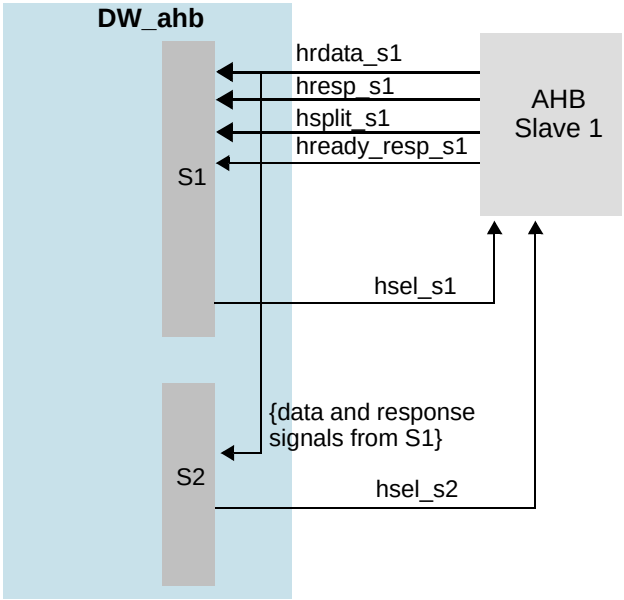


Figure 9: DW_ahb with Aliased Slave

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 8 on page 36, 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.

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.

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.

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.

Timing Diagrams

For timing, refer to the following diagrams:

- Bursts with specified length (Figure 4 on page 30)
- Bursts with unspecified length and equal priority levels (Figure 5 on page 30)
- Bursts with unspecified length and different priority levels (Figure 6 on page 31)

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

- Simple transfer (Figure 10 on page 39)
- Transfer with wait states (Figure 11 on page 39)
- Multiple transfers (Figure 12 on page 40)
- Granting access with no wait states (Figure 13 on page 40)
- Granting access with wait states (Figure 14 on page 40)
- Data bus ownership (Figure 15 on page 41)
- Hand-over after burst (Figure 16 on page 41)

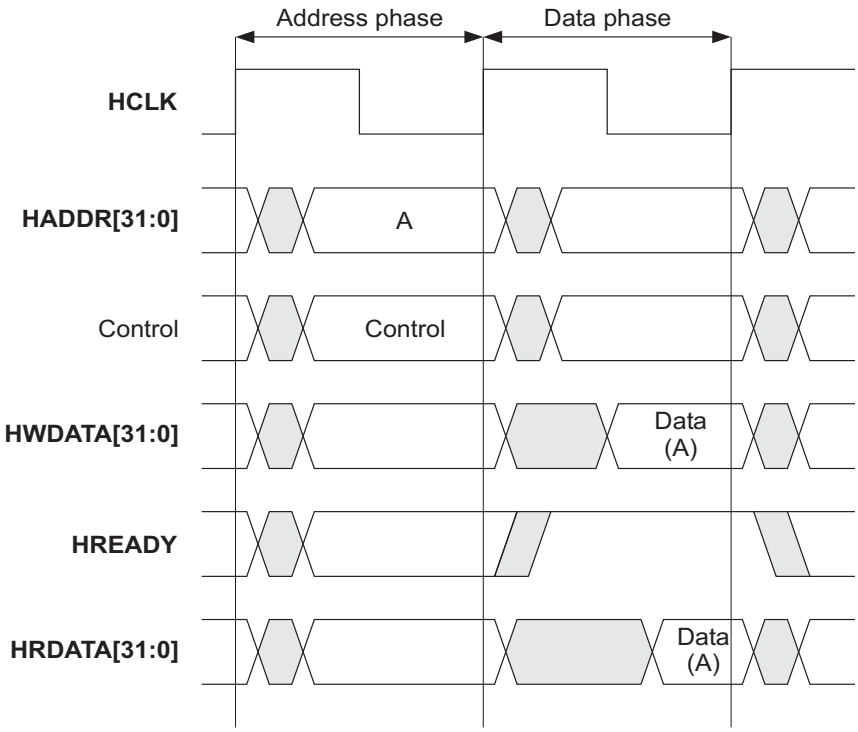


Figure 10: Simple Transfer

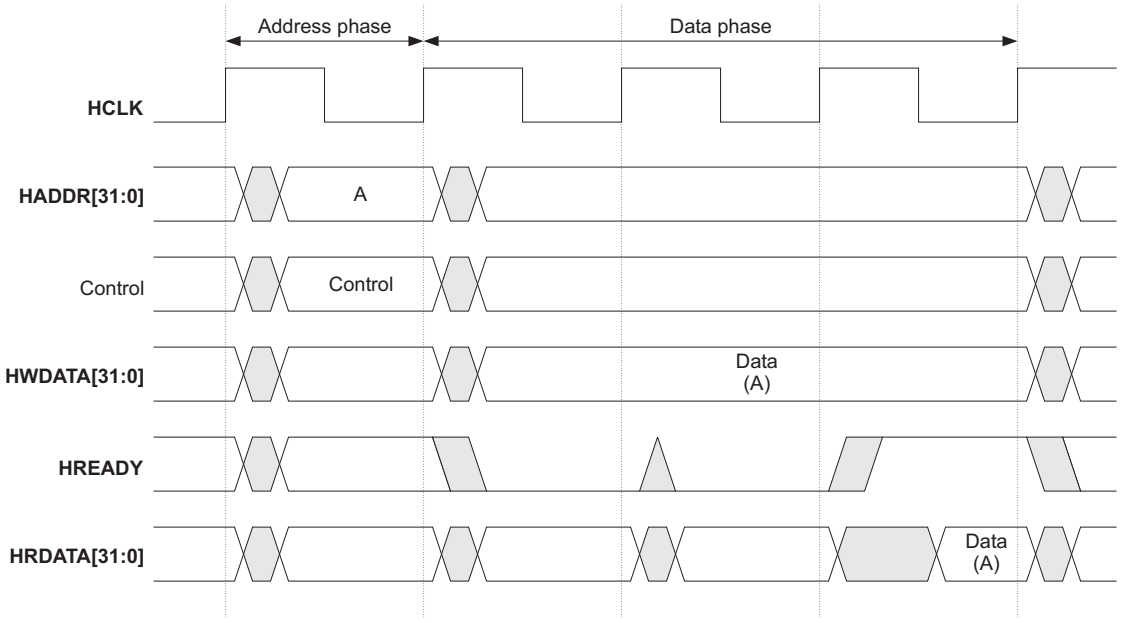


Figure 11: Transfer with Wait States

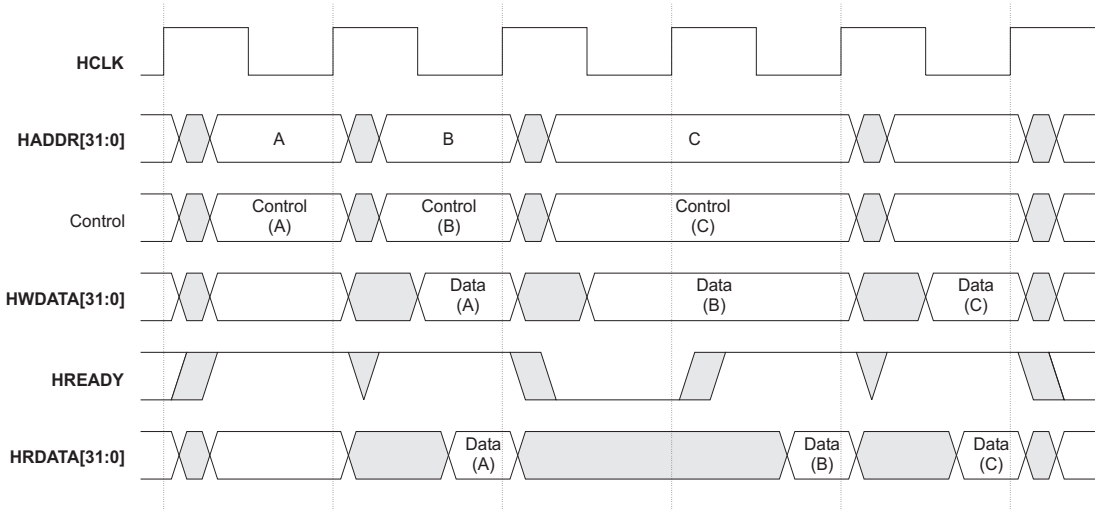


Figure 12: Multiple Transfers

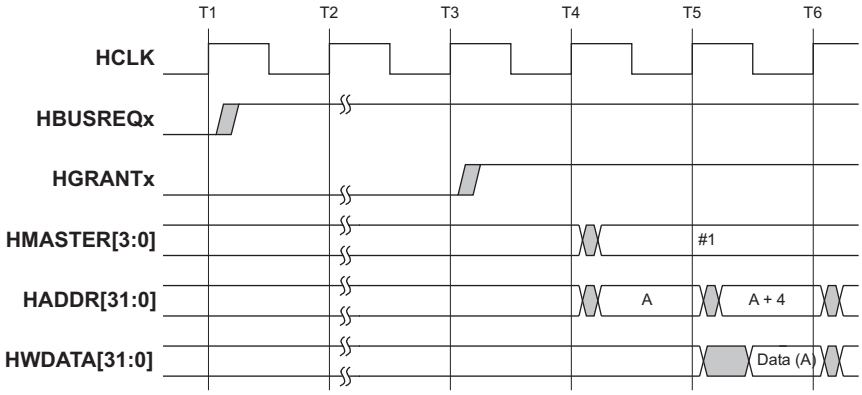


Figure 13: Granting Access with No Wait States

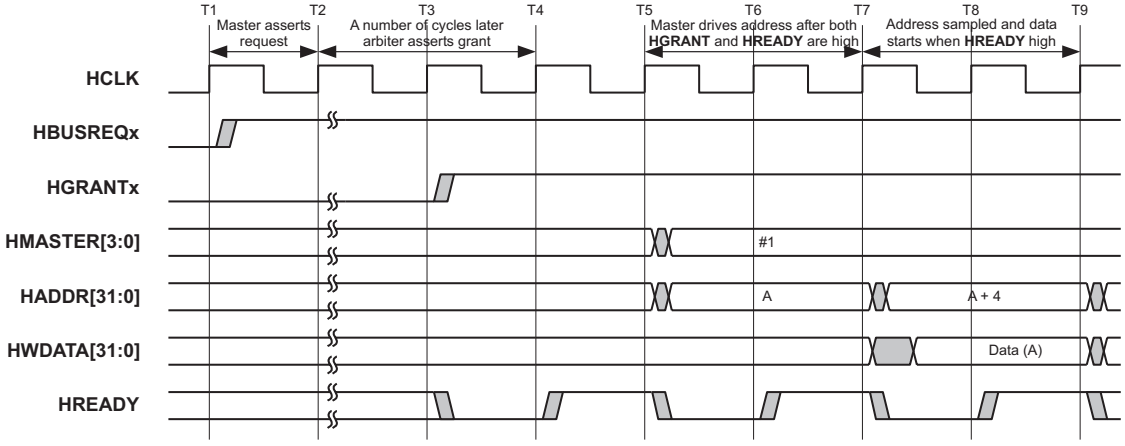


Figure 14: Granting Access with Wait States

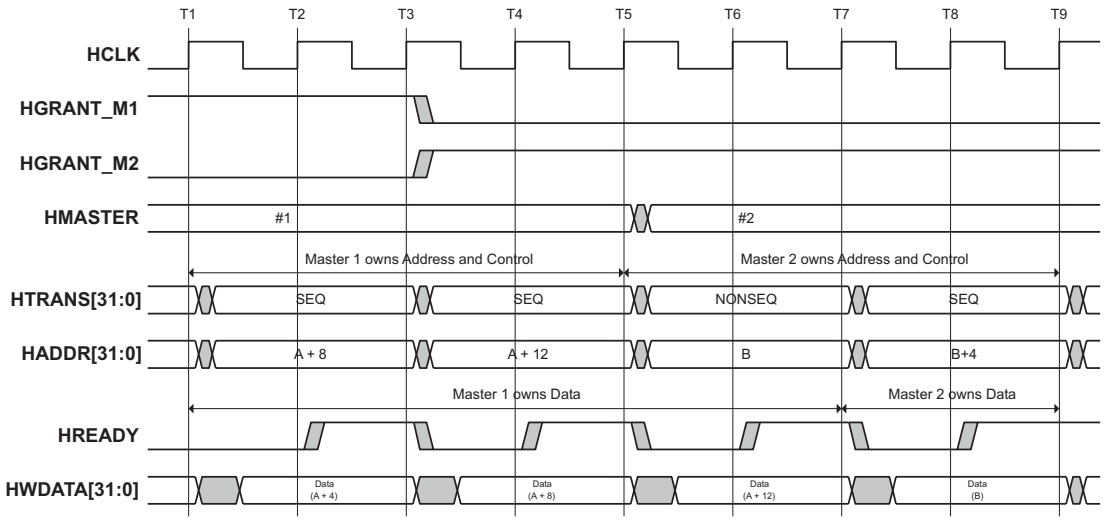


Figure 15: Data Bus Ownership

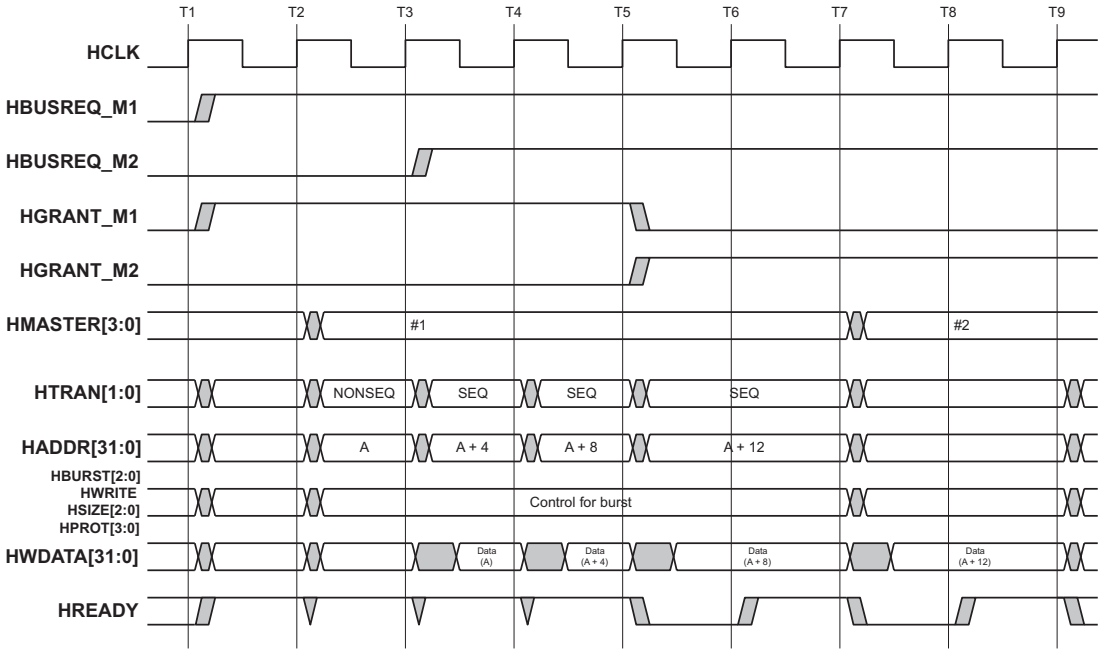


Figure 16: Handover after Burst

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.

Parameter Descriptions

You use coreConsultant to configure the following parameters and generate the configured code. Parameters are listed in the following tables:

Table 2 lists the parameters that you configure using the Synopsys coreConsultant tool.

Table 2: DW_ahb Top-Level Parameters

Label	Parameter Definition
DesignWare 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.

Table 2: DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
Number of AHB Master Ports	**Parameter Name:** NUM_AHB_MASTERS **Legal Values:** 1 to 15 **Default Value:** 2 **Dependencies:** When DesignWare AMBA 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.
External endian control?	**Parameter Name:** AHB_XENDIAN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: If False (0), the endian type of the DW_ahb is controlled by an external signal. If True (1), the endian type is set at configuration by the BIG_ENDIAN parameter. If controlled externally, the endian type is deemed bootable. If controlled internally, there is no way to change the endian type after the component has been configured.
System Endianness	**Parameter Name:** BIG_ENDIAN **Legal Values:** Little-Endian (0) or Big-Endian (1) **Default Value:** Little-Endian (0) **Dependencies:** This is relevant only when AHB_HAS_ARBIF = 1. **Description**: By default, the DW_ahb is configured as a Little-Endian system. You can choose the endianness of the system. The setting affects the byte lane routing to/from the arbiter slave.

Table 2: DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
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 AMBA 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 36.
Support Arbiter Pause Mode?	**Parameter Name:** PAUSE **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** Not available in a DesignWare AMBA 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.

Table 2: DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
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 a DesignWare AMBA 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 a DesignWare AMBA 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.
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.

Table 2: DW_ahb Top-Level Parameters (Continued)

Label	Parameter Definition
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.
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_HSELS2ABH **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.

Arbiter Slave Interface Parameters

Table 3 provides the parameters for the DW_ahb arbiter slave interface. The information in the tables shows the coreConsultant field name and the parameter definition.

Table 3: 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
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.

Table 3: Slave Arbiter Configuration Parameters (Continued)

Label	Parameter Definition
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 32.
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 32.
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 33.
Generate slave select on the interface	**Parameter Name:** GEN_HSEL0 **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 No. If the DW_ahb is configured as DesignWare AMBA Lite, then this signal is removed from the I/O. **Description:** Select line for the internal arbiter slave. When there is an internal decoder, the select is hsel_s0. When there is an external decoder, the select is xhsel_s0.

Arbiter Priority Parameters

Specify the priority level for each DW_ahb master by setting the configuration parameter described in Table 4. 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: 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.

Slave Memory Region Definition

Table 5describes 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 6 on page 52 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 5: Slave 1 Memory Region Definition

Label	Parameter Definition
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 37.

Table 5: 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 37.
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 37.
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 37.
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.

Table 5: Slave 1 Memory Region Definition (Continued)

Label	Parameter Definition
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. **Description:** If the slave has an hsplit bus, then set this parameter to True (1).

Table 6: Slaves 2–15 Memory Region Definition

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 37.
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 37.
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 37.

Table 6: Slaves 2–15 Memory Region Definition (Continued)

Label	Parameter Definition
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 37.
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.
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. **Description:** If the slave has an hsplit bus, then set this parameter to True (1).

Normal Mode Address Map Parameters

Table 7 on page 54 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 8 on page 55 provides the normal mode address map configuration parameters for slaves 2 through 15.

Table 7: 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: 0x2000000 Region 2: 0x3000000 Region 3: 0x4000000 Region 4: 0x5000000 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.
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:** 0x00000000 to 0xffffffff **Default Value:** for regions 1 to 8, the default value is: Region 1: 0x200ffff Region 2: 0x300ffff Region 3: 0x400ffff Region 4: 0x500ffff Region 5: 0x600ffff Region 6: 0x700ffff Region 7: 0x800ffff Region 8: 0x900ffff **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 8: 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:** 0x000003ff 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.
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.

Table 8: Normal Mode Address Map (Slaves 2 – 15) Configuration (Continued)

Label	Parameter Definition
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.
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.

Boot Mode Address Map Parameters

Table 9 on page 57 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 10 on page 58 provides the boot mode address map configuration parameters for slaves 2 through 15.

Table 9: 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: 0x27000000 Region 2: 0x28000000 Region 3: 0x29000000 Region 4: 0x2a000000 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: 0x2900ffff Region 4: 0x2a00ffff 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 10: 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 10: 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.

Parameters for Weighted-Token Control Signals

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

Table 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 a DesignWare AMBA 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 a DesignWare AMBA 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 a DesignWare AMBA 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 a DesignWare AMBA 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 11: Configuration Parameters for Weighted-Token Arbitration

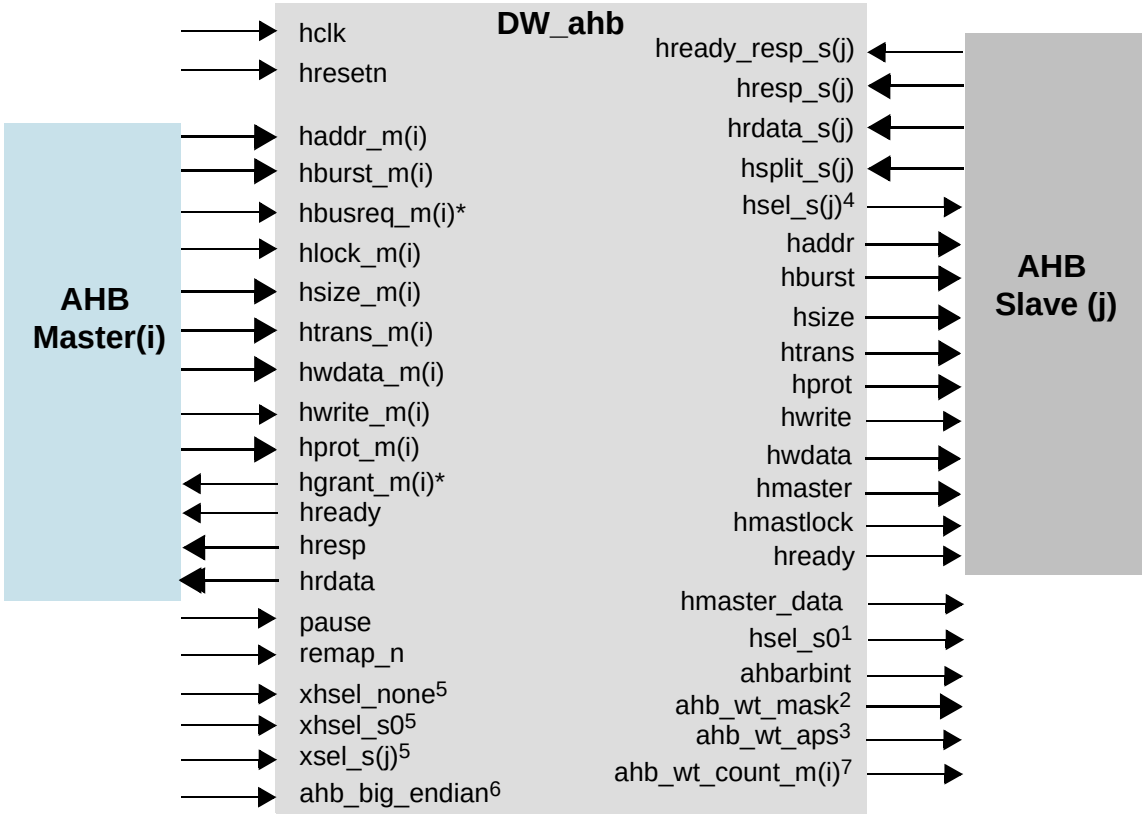
Label	Parameter Definition
Total Cycle Limit	**Parameter Name:** AHB_TCL **Legal Values:** 0x0 to 0xffffffff **Default Value:** 0x0000ffff **Dependencies:** Configured AHB is not a DesignWare AMBA Lite AHB. Enabled if an arbiter slave interface is included and weighted-token arbitration has been enabled. 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 a DesignWare AMBA 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

The following subsections describe the DW_ahb I/O signals:

DW_ahb Interface Diagram

Figure 17 shows the I/O signals for DW_ahb.



for i = 1; i <= Number of AHB masters
for j = 1; j <= Number of AHB slaves

1 (optional) select line for arbiter slave interface registers
2 (optional) weighted token mask
3 (optional) weighted-token arbitration period start
4 (optional) slave select lines when internal decoder configured
5 (optional) slave select lines when external decoder configured
6 (optional) external endianness control is configured
7 (optional) clock token counter outputs if weighted-token arbitration scheme is configured
* optional signals

Figure 17: DW_ahb Interface Diagram

DW_ahb Signal Descriptions

Table 12 identifies the signals that are associated with the DW_ahb.

Table 12: DW_ahb Signal Description

Name	Width	I/O	Description
hclk	1 bit	Input	AHB Clock Signal. This clock times all bus transfers. All signal timings are related to the rising edge of hclk.
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 AMBA OCB interface (asynchronous assertion, synchronous de-assertion). NOTE: 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. 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.
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.
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.
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 a DesignWare AMBA Lite configuration.
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.
hsize_m(i)	3 bits	Input	Indicates size of transfer. There is one hsize_m(i) bus for each master in the system.
htrans_m(i)	2 bits	Input	Indicates the type of transfer being performed.
hwdata_m(i)	[w–1:0][1]	Input	Transfer write data. Each master has its own hwdata_m(i) signal.
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.
hprot_m(i)	4 bits	Input	Protection Control Signals. There is one hprot_m(i) bus for each master in the system.
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 a DesignWare AMBA Lite configuration.
hready	1 bit	Output	Ready response from selected slave. This signal is passed to all AHB masters and slaves.
hresp	2 bits	Output	Transfer response. This signal is passed to all AHB masters.

1: w = the width of the AHB data bus. Maximum width is 256 bits.

Table 12: DW_ahb Signal Description

Name	Width	I/O	Description
hrdata	[w–1:0][1]	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.
Slave Signals generated for j = 1; j <= Number of AHB Slaves (1 to 15)			
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. Exists only if slave is split capable.
hrdata_s(j)	[w–1:0][1]	Input	Readback data from slaves. Each slave has its own hrdata bus.
hresp_s(j)	2 bits	Input	Transfer response from individual slave. There is one hresp_s(j) bus for each slave in the system.
hready_resp_s(j)	1 bit	Input	Response from slave. When asserted, current transfer has completed. There is one hready_resp_s(j) signal for each slave in the system.
hsel_s(j)	1 bit	Output	*Optional*. Included when internal decoder is used (parameter AHB_HAS_XDCDR = 0). 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.
xhsel_s(j)	1 bit	Input	*Optional*. Slave select line. This signal is included if the DW_ahb is configured to use an external decoder (AHB_HAS_XDCDR = 1).
haddr	32 bits	Output	Address bus from selected master. This is passed to all AHB slaves.
hburst	3 bits	Output	Burst type from selected master. This is passed to all AHB slaves.
1: w = the width of the AHB data bus. Maximum width is 256 bits.			
hsize	3 bits	Output	Transfer size from selected master. Indicates the size of the transfer. This is passed to all AHB slaves.
htrans	2 bits	Output	Transfer type from selected master. This is passed to all AHB slaves.
hprot	4 bits	Output	Transfer protection information from selected master. Used only by slaves that can support this feature.
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.
hwdata	[w–1:0][1]	Output	Write data bus from selected master. This signal is passed to all AHB slaves.
hmaster	4 bits	Output	Indicates which master currently has ownership of the address and control bus. This is generated by the arbiter.
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.

Table 12: DW_ahb Signal Description

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.
pause	1 bit	Input	*Optional*. Asserted to put the arbiter into low-power mode by granting the dummy master ownership of the bus. This signal is Active High. This signal is not included if the PAUSE parameter is not enabled
ahbarbint	1 bit	Output	*Optional*. Interrupt signal to Interrupt Controller. The arbiter will flag an interrupt when an Early Burst Termination occurs. This signal is not included if the EBTEN parameter is set to No.
hmaster_data	4 bits	Output	Indicates which master currently has ownership of the data bus.
hsel_s0	1 bit	Output	*Optional*. When asserted, indicates that the arbiter slave has been selected. This signal is not included if ABH_HAS_ARBIF is set to False or if GEN_HSEL0 is set to False.
xhsel_s0	1 bit	Input	*Optional*. 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. This signal is included if the DW_ahb is configured to use an external decoder (AHB_HAS_XDCDR = 1).
ahb_wt_mask	[*y*:1][2]	Output	*Optional*. This signal is included when the weighted token mask is configured. 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.
ahb_wt_aps	1 bit	Output	*Optional*. This signal is included when the weighted token mask is configured and indicates when the weighted-token arbitration period starts. 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.
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. They are included on the interface when the configuration parameter AHB_WTEN = True (1) and AHB_WTEN_DEBUG = True (1).

1: w = the width of the AHB data bus. Maximum width is 256 bits.

2: y = NUM_AHB_MASTERS

6 Registers

This chapter describes the programmable registers of the DW_ahb and contains the following sections:

Register Memory Map

Table 13 shows the memory map for the DW_ahb peripheral.

Table 13: 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

Table 13: Memory Map – Arbiter Slave Interface Registers (Continued)

Name	Address Offset	R/W	Width	Description
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
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:** 0x000
EBT_EN	0x40	R/W	1 bit	Early burst termination enable **Reset Value:** 0x0 (disabled)
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

Table 13: Memory Map – Arbiter Slave Interface Registers (Continued)

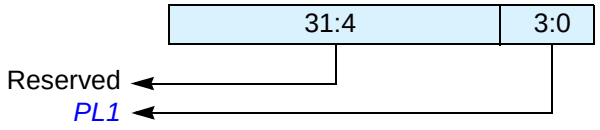
Name	Address Offset	R/W	Width	Description
Address locations for master tokens				
AHB_CL_M(i)	for i in 1-15 0x50 + (0x04*i)	R/W	See Description	Master clock tokens. **Width:** CCL = AHB_CCL_WIDTH
AHB_COMP_VERSION	0x90	R	32 bits	Component version ID **Reset Value:** See the Releases table in the DW_ahb Release Notes

Register and Field Descriptions

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

PL1

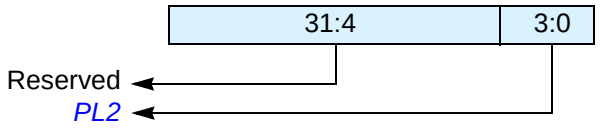
- **Name:** Arbitration Priority Master 1 Register
- **Size:** 4 bits
- **Address Offset:** 0x00
- **Read/Write Access:** Read/Write



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

PL2

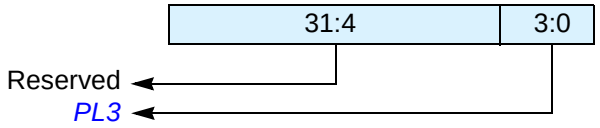
- **Name:** Arbitration Priority Master 2 Register
- **Size:** 4 bits
- **Address Offset:** 0x04
- **Read/Write Access:** Read/Write



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

PL3

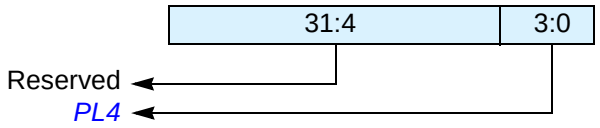
- **Name:** Arbitration Priority Master 3 Register
- **Size:** 4 bits
- **Address Offset:** 0x08
- **Read/Write Access:** Read/Write



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

PL4

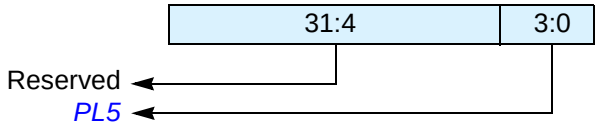
- **Name:** Arbitration Priority Master 4 Register
- **Size:** 4 bits
- **Address Offset:** 0x0c
- **Read/Write Access:** Read/Write



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

PL5

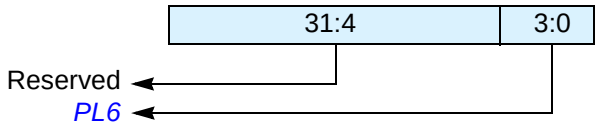
- **Name:** Arbitration Priority Master 5 Register
- **Size:** 4 bits
- **Address Offset:** 0x10
- **Read/Write Access:** Read/Write



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

PL6

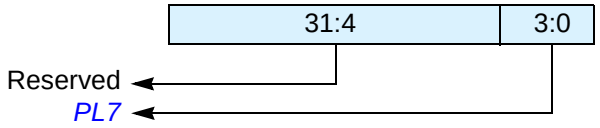
- **Name:** Arbitration Priority Master 6 Register
- **Size:** 4 bits
- **Address Offset:** 0x14
- **Read/Write Access:** Read/Write



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

PL7

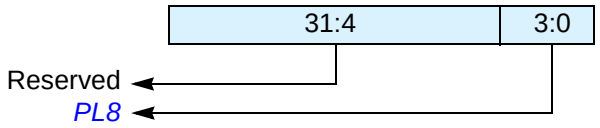
- **Name:** Arbitration Priority Master 7 Register
- **Size:** 4 bits
- **Address Offset:** 0x18
- **Read/Write Access:** Read/Write



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

PL8

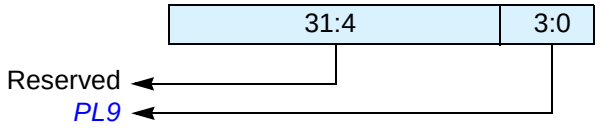
- **Name:** Arbitration Priority Master 8 Register
- **Size:** 4 bits
- **Address Offset:** 0x1c
- **Read/Write Access:** Read/Write



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

PL9

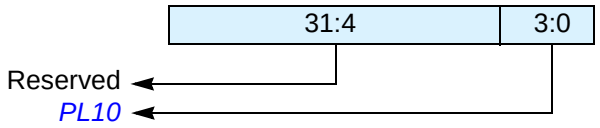
- **Name:** Arbitration Priority Master 9 Register
- **Size:** 4 bits
- **Address Offset:** 0x20
- **Read/Write Access:** Read/Write



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

PL10

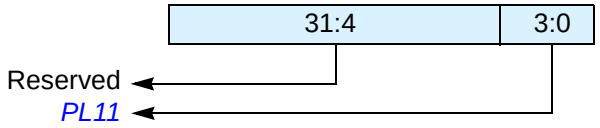
- **Name:** Arbitration Priority Master 10 Register
- **Size:** 4 bits
- **Address Offset:** 0x24
- **Read/Write Access:** Read/Write



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

PL11

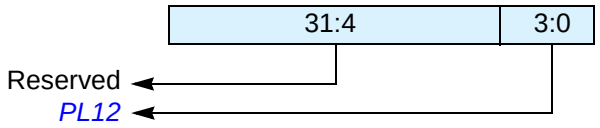
- **Name:** Arbitration Priority Master 11 Register
- **Size:** 4 bits
- **Address Offset:** 0x28
- **Read/Write Access:** Read/Write



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

PL12

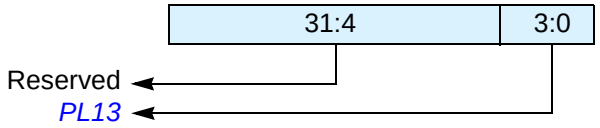
- **Name:** Arbitration Priority Master 12 Register
- **Size:** 4 bits
- **Address Offset:** 0x2c
- **Read/Write Access:** Read/Write



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

PL13

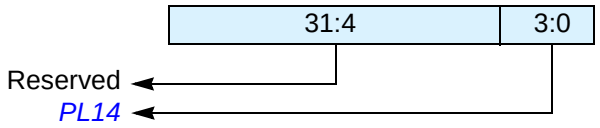
- **Name:** Arbitration Priority Master 13 Register
- **Size:** 4 bits
- **Address Offset:** 0x30
- **Read/Write Access:** Read/Write



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

PL14

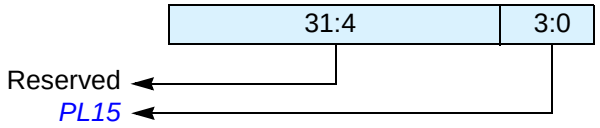
- **Name:** Arbitration Priority Master 14 Register
- **Size:** 4 bits
- **Address Offset:** 0x34
- **Read/Write Access:** Read/Write



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

PL15

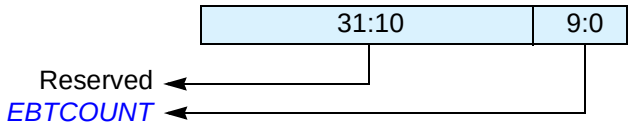
- **Name:** Arbitration Priority Master 15 Register
- **Size:** 4 bits
- **Address Offset:** 0x38
- **Read/Write Access:** Read/Write



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

EBTCOUNT

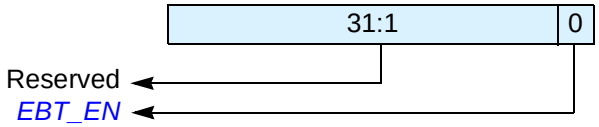
- **Name:** Early Burst Termination Count Register
- **Size:** 10 bits
- **Address Offset:** 0x3c
- **Read/Write Access:** Read/Write



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

EBT_EN

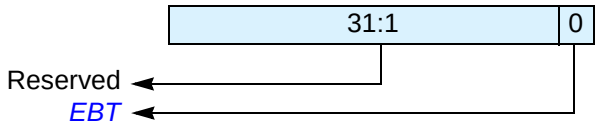
- **Name:** Early Burst Termination Enable
- **Size:** 1 bit
- **Address Offset:** 0x40
- **Read/Write Access:** Read/Write



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

EBT

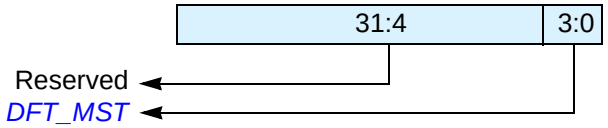
- **Name:** Early Burst Termination Register
- **Size:** 1 bit
- **Address Offset:** 0x44
- **Read/Write Access:** Read to clear



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

DFT_MST

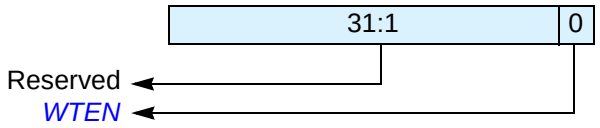
- **Name:** Default Master ID Number Register
- **Size:** 4 bits
- **Address Offset:** 0x48
- **Read/Write Access:** Read/Write



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. **Reset Value:** 0x0

WTEN

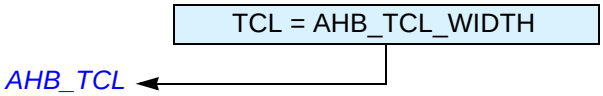
- **Name:** Weighted-Token Arbitration Scheme Enable Register
- **Size:** 1 bit
- **Address Offset:** 0x4c
- **Read/Write Access:** Read/Write



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

AHB_TCL

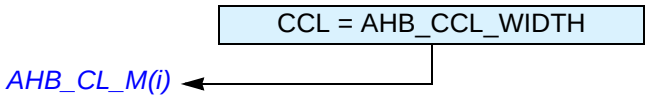
- **Name:** Default Master Register
- **Size:** TCL = AHB_TCL_WIDTH
- **Address Offset:** 0x50
- **Read/Write Access:** Read/Write



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

AHB_CL_M(*i*)

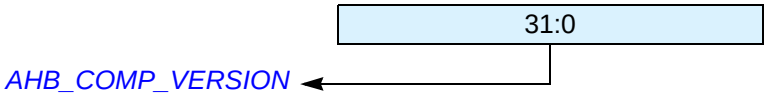
- **Name:** Master Clock Token Register
- **Size:** CCL = AHB_CCL_WIDTH
- **Address Offset:** For *i* in 1-15, 0x50 + (0x04**i*)
- **Read/Write Access:** Read/Write



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

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 Releases table in the DW_ahb Release Notes

7 Programming the DW_ahb

This chapter describes the programmable features of the DW_ahb in the following sections:

Programming Considerations

The following sections outline programming considerations for the DW_ahb:

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 AMBA 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 8 on page 36.

Arbiter Slave Interface Registers

Table 13 on page 69 describes the memory map for the programmable registers that make up the optional arbiter slave interface. 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.

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.

Note

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

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

Note

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

If you specify the gap between the start and end address for the arbiter slave interface as greater than 1 KB, the memory map detailed in Table 13 on page 69 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.

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.

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.

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.

Software Drivers

The DesignWare AMBA On-Chip Bus family includes software drivers for some of the synthesizable IP components. These low-level drivers allow you to easily program a DesignWare AMBA OCB component and integrate your code into a larger software system. The software drivers allow rapid, low-level code development by eliminating programming interfaces (APIs), example code, and header files.

The software drivers are currently under limited availability. Contact Synopsys at designware@synopsys.com for details.

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. The following sections describe the testbench:

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.

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.

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:

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

External Decoder

Exercises all xhsel_s(i) and xhsel_none lines.

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.

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.

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)

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.

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.

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.

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:
    1. Normal priority rules if no master or all the requesting masters are (wt) masked.
    2. Bus granting in favor of a lower priority master if an higher priority one gets masked.
    3. 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:
    4. If the current master is the only one requesting the bus it must stay on the bus.
    5. If there are pending-unmasked requests the current master must be removed from the bus.

Note

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.

Slave ACT Compliance

These tests exercise all of the required sequences of transfers to the slave to allow it to be ACT compliant. For more information about running these tests, see “Verify the Simulation Model” on page 19.

DW_ahb Testbench

As illustrated in Figure 18, the DW_ahb testbench is a Verilog testbench that includes an instantiation of the design under test (DUT) and a Vera shell. 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.

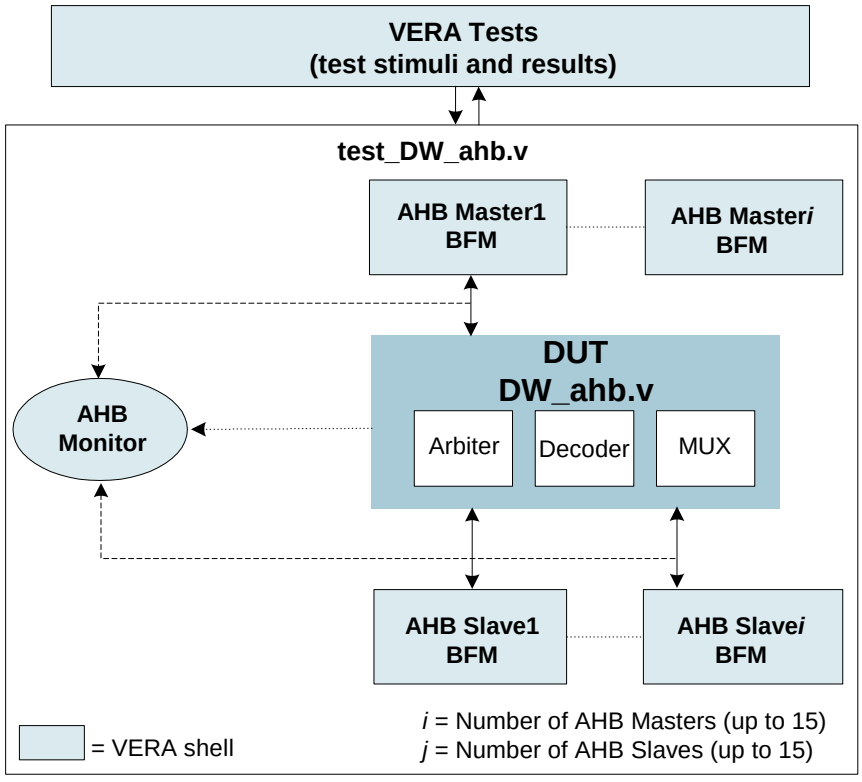


Figure 18: DW_ahb Testbench

A
Database Description

This appendix lists the deliverables and other reference files that are generated from the coreConsultant flow.

This appendix includes the following sections:

Design/HDL Files

The following sections describe the design and HDL files that are produced by coreConsultant when configuring and verifying a DesignWare AMBA component.

RTL-Level Files

Table 14 describes the RTL files that are generated by the Create RTL activity of the coreConsultant GUI. They are encrypted except where otherwise noted.

Note

Any Synopsys synthesis tool or simulator can read encrypted RTL files.

Table 14: 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.
		When you include the component in your simulation, you must include the DesignWare libraries by using the following options in your simulator invocation:
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`
		For an example of this process, refer to the *DW_AMBA QuickStart SingleLayer Example Guide*.

./src/*component_submodule*.v	Yes	Sub-modules of component
./src/*component*_constants.v	No	Includes the constants used internally in the design.
./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 -f component.lst`
./src/*.update	Yes	Ignore these files. Used for VHDL generation
./export/*component*_inst.v	No	Instantiation of configured component for use in design

Simulation Model Files

Table 15 includes the simulation model files generated for the component during the Generate GTECH Simulation activity in coreConsultant. These files are needed when you are using a non-Synopsys simulator (when you can not use the encrypted RTL).

Table 15: 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.
		When you use this simulation model in your simulation, you must include the DesignWare libraries by using the following options in your simulator invocation:
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`
		For an example of this process, refer to the *DW_AMBA QuickStart SingleLayer Example Guide*.

Register Map Files

These files only pertain to DW_ahb and DW_apb slaves, basically components that have a programming interface. The DesignWare AMBA components that do not have register map files are the DW_apb, DW_ahb_icm, and DW_ahb_h2h components. These files include address definitions (memory map) for the component. Table 16 includes a description of the C and Verilog header files generated for components with programming interfaces.

Table 16: Header Files

Files	Encrypted?	Purpose
./c_headers/*component*_defs.h	No	For use when programming the component in a C environment.
./verilog_headers/*component*_defs.v	No	For use when programming the component in a Verilog environment.

Synthesis Files

Table 17 includes the files that are generated after the Create Gate-Level Netlist activity in coreConsultant is performed on a component.

Table 17: Synthesis Files

Files	Encrypted?	Purpose
./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.

Table 17: Synthesis Files

Files	Encrypted?	Purpose
./syn/constrain/script/*.*	No	Constraint files for the components.
./syn/final/report/*.*	No	Synthesis result files.

Verification Reference Files

The files described in Table 18 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 18: Verification Reference Files

Files	Encrypted?	Purpose
./sim/runtest	No	Perl script that runs the coreConsultant Verify Component 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.
./sim/test_*testname*/test.log	No	Log file for individual test.

For more information about performing verification on your component, see the chapter titled Verification in this databook.

B

DesignWare AMBA Connect and QuickStart Designs

The DesignWare AMBA On-chip Bus environment provides many templates and examples to help you be successful with your own design creation process. This section summarizes these system design aids, and points you to more information about them.

DesignWare AMBA Connect

The DesignWare AMBA Connect “designer” allows you to do is construct, modify, and simulate any single- or multi-layer AMBA system that you can conceive in *about an hour.* DesignWare AMBA Connect accelerates the design process in the following ways:

- Initializes the address map for the subsystem
- Generates the top-level subsystem RTL code
- Provides and executes a subsystem testbench using your chosen simulator

Instead of starting with a blank sheet, you can use any of the pre-assembled and partially configured design starts included in DesignWare AMBA Connect and quickly produce your own subsystem testbench and RTL design.

For details on using DesignWare AMBA Connect, and a description of the pre-assembled DesignWare AMBA Connect templates that are available, refer to the *DesignWare AMBA Connect Databook*.

QuickStart Example Designs

QuickStart examples are provided with the DesignWare AMBA On-chip Bus synthesizable components and verification models to help you learn about these products. The QuickStart examples show how to connect the DesignWare AMBA OCB components to the DW_apb and DW_ahb bus IP, and how to set up a verification environment. These are simulation-only subsystems to view waveforms, and not for use in synthesis. Each example design includes the following information:

- Block diagram of subsystem design, showing connections and ports
- Purpose of the example, and features included
- Example directory structure
- Important configuration and parameter information
- Overview of the testbench and tests that are provided
- Instructions on how to quickly perform a simulation run

For more information about QuickStart examples, refer to the *DesignWare AMBA QuickStart_SingleLayer Guide* and the *DesignWare AMBA QuickStart_MultiLayer Guide*.

C
DesignWare AMBA OCB Constants

Table 19 provides the DesignWare AMBA OCB constant definitions. These definitions can also be found in DW_amba_constants.v file in the src directory of your DW_ahb coreKit.

Table 19: DesignWare AMBA OCB Constant Definitions

DesignWare AMBA OCB 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
HRESP Values	
OKAY	2‘b00
ERROR	2‘b01
RETRY	2‘b10

Table 19: DesignWare AMBA OCB Constant Definitions (Continued)

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

D
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 AMBA On-chip Bus	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 AMBA On-chip Bus 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 AMBA OCB 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

W

Z