SYNOPSYS®

DesignWare® DW_apb_rap

Databook

DW_apb_rap – Product Code

2.09a
December 2020

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.
www.synopsys.com

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.02d onward.

Version	Date	Description
2.09a	December 2020	**Updated:** ■ Version number change for 2020.12a release ■ "Performance" on page 63 ■ "Parameter Descriptions" on page 21, "Signal Descriptions" on page 25, "Register Descriptions" on page 35, and "Internal Parameter Descriptions" on page 65 are auto-extracted with change bars from the RTL **Removed:** ■ Index chapter
2.08a	July 2018	**Updated:** ■ Version number change for 2018.07a release ■ "Overview of Testbench" on page 49 ■ "Overview of Tests" on page 50 ■ "Performance" on page 63 ■ "Parameter Descriptions" on page 21, "Signal Descriptions" on page 25, "Register Descriptions" on page 35, and "Internal Parameter Descriptions" on page 65 are auto-extracted with change bars from the RTL **Removed:** ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.

Version	Date	Description
2.07a	October 2016	■ Version number change to 2016.10a ■ "Parameter Descriptions" on page 21 and "Register Descriptions" on page 35 auto-extracted from the RTL ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Added Entry for the xprop directory in Table 2-1 and Table 2-4. ■ Replaced Figure 2-2 and Figure 2-3 to remove references to Leda ■ Moved "Internal Parameter Descriptions" chapter to Appendix ■ Added "Running VCS XPROP Analyzer"
2.06a	June 2015	■ Added section "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added section "Running SpyGlass on Generated Code with coreAssembler" ■ Added Chapter 4, "Signal Descriptions" that is auto-extracted from the RTL ■ Added Chapter A, "Internal Parameter Descriptions"
2.05a	June 2014	■ Version change for 2014.06a release ■ Added "Performance" section in the "Integration Considerations" chapter ■ Corrected External Input/Output Delay in Signals chapter
2.04e	May 2013	Version change for 2013.05a release. Updated the template.
2.04d	September 2012	Added the product code on the cover and in Table 1-1.
2.04d	March 2012	Corrected reset value for IdCode register.
2.04c	November 2011	Version change for 2011.11a release.
2.04b	October 2011	Version change for 2011.10a release.
2.04a	June 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced "Related Documents" section in Preface.
2.04a	November 2010	Added entry for Ping Test Simulation register.
2.04a	September 2010	Corrected names of include files and vcs command used for simulation
2.03a	December 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks.
2.03a	May 2008	Removed references to QuickStarts, as they are no longer supported.
2.03a	October 2008	Version change for 2008.10a release.
2.02e	June 2008	Version change for 2008.06a release.
2.02d	January 2008	■ Updated for revised installation guide and consolidated release notes titles ■ Changed references of "Designware AMBA" to simply "DesignWare"

Version	Date	Description
2.02d	June 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DW_apb_rap component to the Advanced Peripheral Bus (APB). 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.

DatabookOrganization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_apb_rap.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_apb_rap.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_apb_rap signals.
- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_apb_rap.
- Chapter 6, "Programming the DW_apb_rap" provides information needed to program the configured DW_apb_rap.
- Chapter 7, "Verification" provides information on verifying the configured DW_apb_rap.
- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_apb_rap into your design.
- Appendix A "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Parameters, Signals, and Registers chapters.
- Appendix B, "Glossary" provides a glossary of general terms.

Related Documentation

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

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

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

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

    File > Build Debug Tar-file

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

  a. https://solvnetplus.synopsys.com

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

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

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

  Ensure to include the following:
  - **Product L1:** DesignWare Library IP
  - **Product L2:** AMBA

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

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

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

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

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare APB Peripherals.

Table 1-1 DesignWare APB Peripherals – Product Code: 3771-0

Component Name	Description
DW_apb_gpio	General Purpose I/O pad control peripheral for the AMBA 2 APB bus
DW_apb_rap	Programmable controller for the remap and pause features of the DW_ahb interconnect
DW_apb_rtc	A configurable high range counter with an AMBA 2 APB slave interface
DW_apb_timers	Configurable system counters, controlled through an AMBA 2 APB interface
DW_apb_wdt	A programmable watchdog timer peripheral for the AMBA 2 APB bus

1 Product Overview

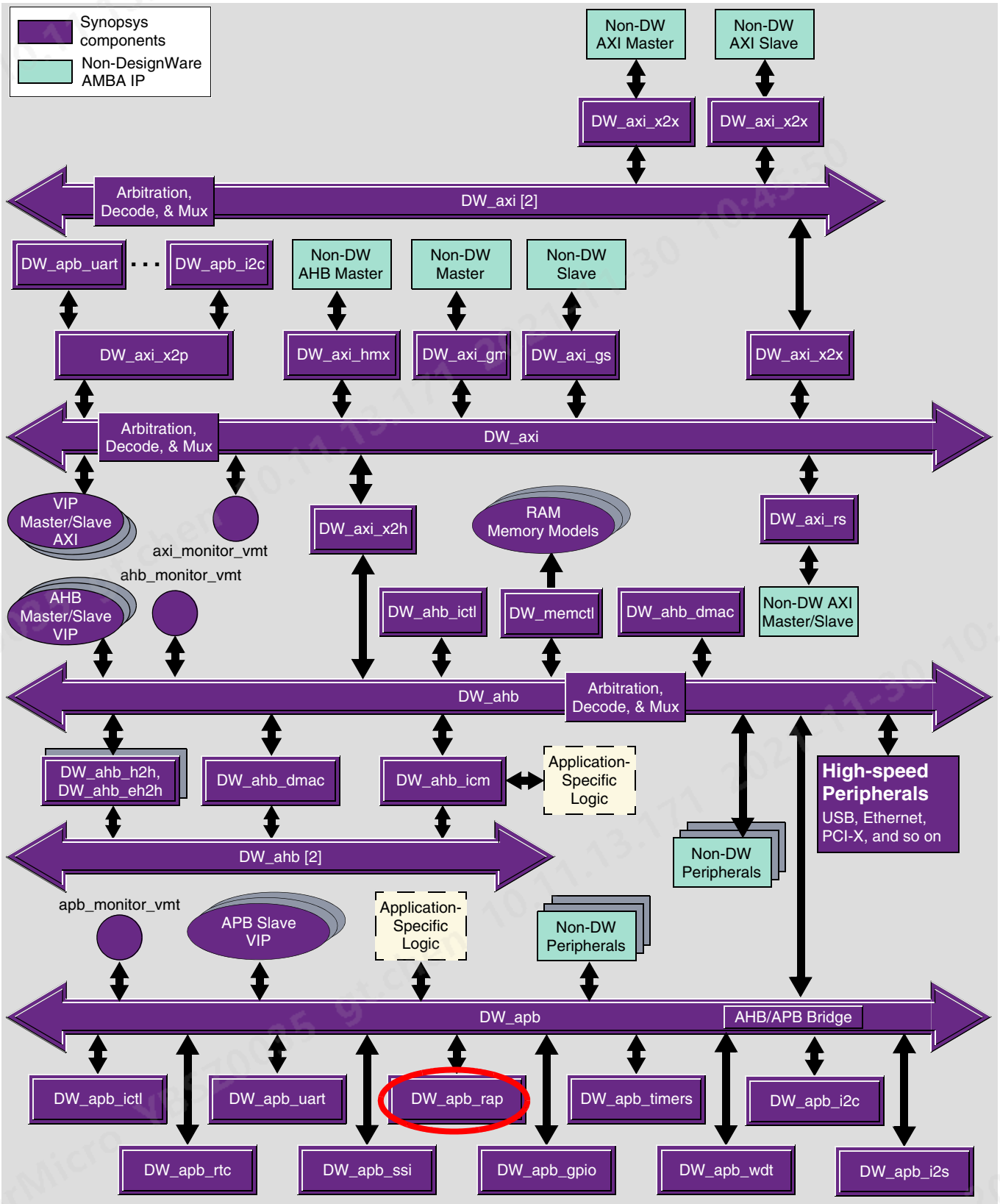
The DW_apb_rap is a programmable Remap and Pause (RAP) controller peripheral. This component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components for AMBA 2.

1.1 DesignWare System Overview

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

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

Figure 1-1 Example of DW_apb_rap in a Complete System



You can connect, configure, synthesize, and verify the DW_apb_rap within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_apb_rap component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

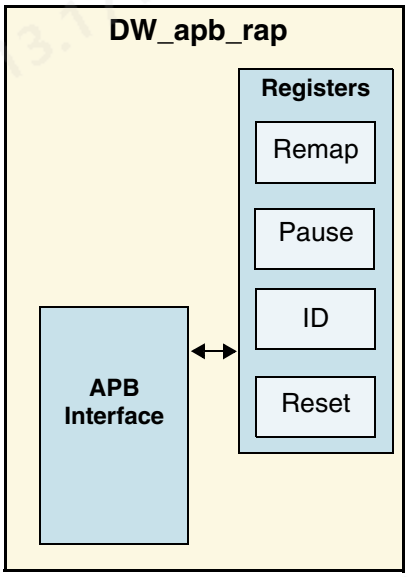
1.2 General Product Description

The Synopsys DW_apb_rap is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the *AMBA Specification, Revision 2.0* from Arm®.

1.2.1 DW_apb_rap Block Diagram

Figure 1-2 shows a block diagram of a fully configured DW_apb_rap peripheral.

Figure 1-2 DW_apb_rap Block Diagram



1.3 Features

- Remap control
- Pause mode
- Identification code register
- Reset Status Register
- Design For Test

1.4 Standards Compliance

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

1.5 Verification Environment Overview

The DW_apb_rap 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 49 section discusses the specific procedures for verifying the DW_apb_rap.

1.6 Licenses

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

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_apb_rap component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, see *Guide to coreTools Documentation*.

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

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

2 Functional Description

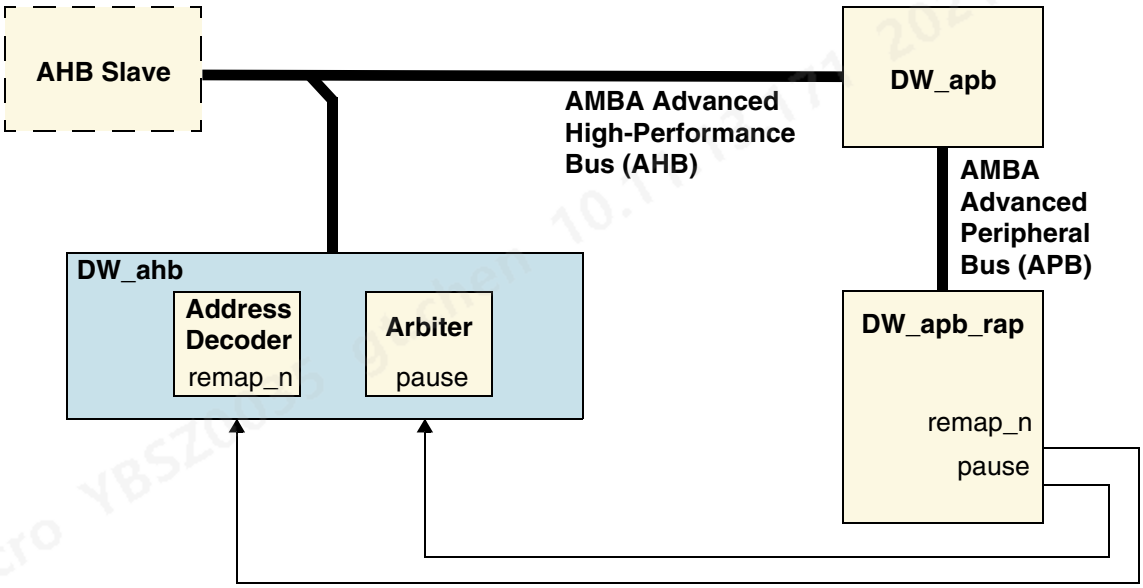
This chapter describes the DW_apb_rap APB slave, which can be configured to support any of four separate functions. This component conforms to the *AMBA Specification, Revision 2.0*, which provides bus-specific and timing information. The following subsections describe the functions of DW_apb_rap.

2.1 Remap Control

Figure 2-1 shows how the DW_apb_rap acts as control registers for the remap and pause functions of the DW_ahb Address Decoder and Arbiter.

The DW_ahb implementation of the AMBA AHB contains an address decoder that supports up to two memory maps: a boot memory map, and a normal memory map. A typical use of this feature is to allow ROM to be mapped to 0x0000 on system reset (boot memory map) and RAM to be remapped to the same memory space after initialization (normal memory map). The DW_apb_rap can be configured to include a remap control register that is used to switch the DW_ahb address decoder from boot mode to normal mode operation.

Figure 2-1 DW_apb_rap Used as Control Registers for Remap and Pause



With power-on reset, the boot memory map is selected. This is reprogrammed some time later to the normal memory map by writing to the remap register in the DW_apb_rap. This in turn sends a signal to the AHB address decoder to change its memory map. The memory map cannot be changed back to the boot map by writing to this register. It is reset only when a power-on reset occurs.

The remap register is cleared by the power-on reset.

2.2 Pause Mode

The arbiter in the DW_ahb implementation of the AMBA AHB bus supports a pause mode that forces other masters off the bus and gives control of the bus to a dummy AHB master. The dummy master reduces power by driving no transactions to the bus.

The DW_apb_rap implements a control register for the DW_ahb pause mode. Writing a 1 to the PauseMode register asserts the pause signal that controls the DW_ahb pause mode. Once the PauseMode register has been set, it can be cleared only by an interrupt. Two interrupt inputs are provided by the DW_apb_rap: one for normal interrupts (irq or irq_n), and one for fast interrupts (fiq or fiq_n).

The PauseMode register is cleared by the power-on reset.

2.3 Identification Code Register

The DW_apb_rap implements a configurable, read-only, identification register. This is typically used to store a processor-accessible system ID code or user version number. The width of the register is configurable, but is limited to the data width of the APB bus interface.

2.4 Reset Status Register

The DW_apb_rap implements a reset status register that is used to capture reset events on any one of up to eight separate system reset signals. After a reset occurs, a processor can query the DW_apb_rap ResetStatus register to determine which reset signal caused the reset event.

Up to eight reset signals are connected to the sys_resets bus of the DW_apb_rap. If any of the reset signals goes active, it asynchronously sets the corresponding bit in the ResetStatus register. This bit remains set until cleared by writing the corresponding bit in the ClrResetStatus register.

The reset status register is not reset by the por_reset_n power-on reset. Up to eight system resets can be monitored.

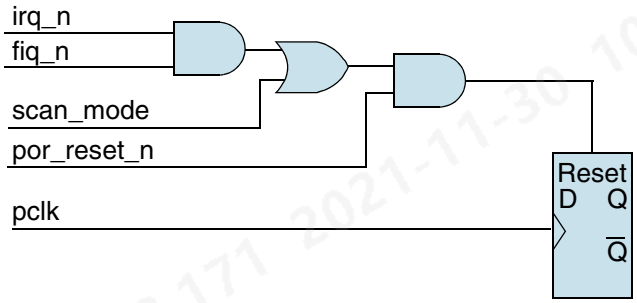
The system resets are not gated with power-on reset in order to ensure all the reset status bits are set initially – it is assumed this is handled at a higher level within the SoC.

2.5 Design For Test

The PauseMode register is asynchronously reset by the interrupt inputs to the DW_apb_rap. During scan testing, the asynchronous reset of this register needs to be isolated from the interrupt inputs.

The DW_apb_rap provides a scan_mode input signal to allow scan logic to control the asynchronous reset, as shown in Figure 2-2.

Figure 2-2 Design For Test – Use of Scan Mode Signal



3 Parameter Descriptions

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

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

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

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
Top Level Parameters	
APB Data Bus Width	Specifies the width of APB Data Bus to which this peripheral is attached. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Enable Pause Mode?	Specifies the pause feature. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** PAUSE
RAP responds to FIQ?	Specifies whether FIQ exists on the I/O. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** PAUSE **Parameter Name:** RAP_HAS_FIQ
Active High Interrupts ?	Generates active-high or active-low interrupts. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** PAUSE == 1 **Parameter Name:** RAP_INT_POL
Enable Remap Mode?	Enables the remap feature. Values must be consistent when the DW_apb_rap is used in conjunction with the DW_ahb. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** REMAP

Table 3-1 Top Level Parameters (Continued)

Label	Description
Enable ID Code?	Enables the ID register feature to provide an ID code value (RAP_ID_NUM) that can be read by the processor. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** RAP_ID
ID Code Size	Specifies the width of the identification code that is configured in RAP_ID_NUM. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** RAP_ID == 1 **Parameter Name:** RAP_ID_WIDTH
Identification Code	Specifies the ID code value that you can set so that it can be read back later by the processor. While the minimum value of the ID code can be 0x0, the maximum value is determined by RAP_ID_WIDTH. **Values:** 0x0, ..., [::DW_apb_rap::calc_rap_idcode_limit RAP_ID_WIDTH] **Default Value:** 0xbeefcafe **Enabled:** RAP_ID == 1 **Parameter Name:** RAP_ID_NUM

3.2 RAP Reset Configuration Parameters

Table 3-2 RAP Reset Configuration Parameters

Label	Description
RAP Reset Configuration	
Build Reset Status Register?	Enables the reset status feature. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** RAP_RESET
Number of Resets	Specifies the number of resets whose status must appear in the reset register. **Values:** 1, 2, 3, 4, 5, 6, 7, 8 **Default Value:** 1 **Enabled:** RAP_RESET == 1 **Parameter Name:** RAP_NUM_RESETS
Reset x Active High (for x = 0; x <= RAP_NUM_RESETS-1)	Specifies the number of resets whose status must appear in the reset register. The DW_apb_rap module can be configured to accept different polarities of sys_resets. Each polarity can be changed independently. If RAP_RESET_POLn is set to 1, an active-high event on the corresponding sys_resets input causes the associated register bit to be set to 1. If one bit of sys_resets is used to monitor the power-on reset signal (por_reset_n), its polarity must be set to active low. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** (RAP_RESET == 1 && RAP_NUM_RESETS >= (x+1)) **Parameter Name:** RAP_RESET_POL_x

4 Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- APB Interface on page 27
- Miscellaneous on page 29
- Reset Status on page 30
- Pause on page 31
- Remap on page 33

4.1 APB Interface Signals

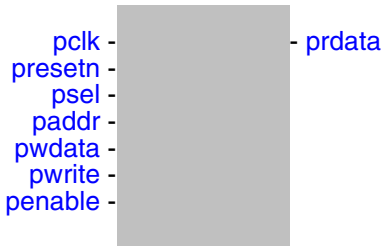


Table 4-1 APB Interface Signals

Port Name	I/O	Description
pclk	I	APB clock. This clock times all bus transfers. All signal timings are related to the rising edge of pclk. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
presetn	I	APB Reset Signal. The bus reset signal is used to reset the system and the bus on the DesignWare interface. Asynchronous APB interface domain reset. This signal resets only the bus interface. The signal is asserted asynchronously, but is deasserted synchronously after the rising edge of pclk. The synchronization must be provided external to this component. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
psel	I	APB peripheral select **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 APB Interface Signals (Continued)

Port Name	I/O	Description
paddr[RAP_ADDR_SLICE_LHS:0]	I	APB address bus **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pwrite	I	APB write control **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
penable	I	APB enable control that indicates the second cycle of the APB frame. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prdata[(APB_DATA_WIDTH-1):0]	O	APB read data **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Miscellaneous Signals

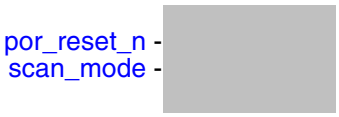


Table 4-2 Miscellaneous Signals

Port Name	I/O	Description
por_reset_n	I	Power-on reset (por) that resets the programming registers. This is an asynchronous power-on reset (por). **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
scan_mode	I	Scan mode used to ensure that test automation tools can control all asynchronous flip-flop signals. This signal must be asserted during scan testing and deasserted at all other times. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.3 Reset Status Signals

sys_resets -

Table 4-3 Reset Status Signals

Port Name	I/O	Description
sys_resets[(RAP_NUM_RESETS-1):0]	I	Each bit corresponds to a different system reset. An active signal on any of these inputs asynchronously sets the corresponding status bit in the ResetStatus register. The active polarity for each bit of this signal is independently configurable using the RAP_RESET_POLn parameter. If you want to maintain the status of the power-on reset, connect it to one bit of this input. **Exists:** RAP_RESET==1 **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.4 Pause Signals

Table 4-4 Pause Signals

Port Name	I/O	Description
irq	I	Active-high interrupt request from the interrupt controller. **Exists:** RAP_INT_POL==1 && PAUSE==1 **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
irq_n	I	Active-Low interrupt request from the interrupt controller. **Exists:** RAP_INT_POL==0 && PAUSE==1 **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
fiq	I	Active-high fast interrupt request from the interrupt controller. **Exists:** RAP_INT_POL==1 && PAUSE==1 && RAP_HAS_FIQ==1 **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
fiq_n	I	Active-Low fast interrupt request from the interrupt controller. **Exists:** RAP_INT_POL==0 && PAUSE==1 && RAP_HAS_FIQ==1 **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Table 4-4 Pause Signals (Continued)

Port Name	I/O	Description
pause	O	Causes the DW_ahb arbiter to grant the AHB bus to the default master until an interrupt is received. **Exists:** PAUSE==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Remap Signals

Table 4-5 Remap Signals

Port Name	I/O	Description
remap_n	O	Selects between two memory maps: one after a system reboot, and the other for normal operation. **Exists:** REMAP==1 **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

5 Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
DW_apb_rap_addr_block1 on page 38	DW_apb_rap address block **Exists:** Always

5.1 DW_apb_rap_mem_map/DW_apb_rap_addr_block1 Registers

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

Table 5-5 Registers for Address Block: DW_apb_rap_mem_map/DW_apb_rap_addr_block1

Register	Offset	Description
RAP_PAUSEMODE on page 39	0x0	Pause Mode Register
RAP_IDCODE on page 40	0x4	ID Code Register
RAP_REMAPMODE on page 41	0x8	Remap Mode Register
RAP_RESETSTAT on page 42	0xc	Reset Status Register
RAP_CLRRESET on page 44	0x10	Clear Reset Status Register
RAP_VER_ID on page 45	0x14	Component Version Register
RAP_PING on page 46	0x18	Ping Test Simulation Register

5.1.1 RAP_PAUSEMODE

- **Name:** Pause Mode Register
- **Description:** This register specifies the pause mode register.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** PAUSE==1

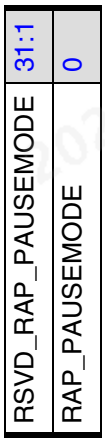


Table 5-6 Fields for Register: RAP_PAUSEMODE

Bits	Name	Memory Access	Description
31:1	RSVD_RAP_PAUSEMODE	R	RAP_PAUSEMODE 31to1 Reserved bits **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	RAP_PAUSEMODE	R/W	This bit controls the pause output. The reset state of the pause output is 0, which indicates that the pause functionality is not active. By writing a 1 to this register, the pause output is asserted. It is not possible to write a 0 into this register. If an attempt is made to write a 0, the register retains its previous state. Only an interrupt causes this register to be reset. **Values:** ■ 0x0 (INACTIVE): Inactive State ■ 0x1 (ASSERT): Assert pause output **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.2 RAP_IDCODE

- **Name:** ID Code Register
- **Description:** This register specifies the ID code.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** RAP_ID==1

Table 5-7 Fields for Register: RAP_IDCODE

Bits	Name	Memory Access	Description
31:y	RSVD_RAP_IDCODE	R	RAP_IDCODE 31toRAP_ID_WIDTH Reserved bits **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** RAP_ID_WIDTH
x:0	RAP_IDCODE	R	This bit specifies the user-specified code that a system can read. It can be used for chip identification, and so on. Reading from this register when the ID feature is not selected (RAP_ID = 0) returns 0. **Value After Reset:** RAP_ID_NUM **Exists:** Always **Range Variable[x]:** RAP_ID_WIDTH - 1

5.1.3 RAP_REMAPMODE

- **Name:** Remap Mode Register
- **Description:** This register specifies the remap mode.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** REMAP==1

Table 5-8 Fields for Register: RAP_REMAPMODE

Bits	Name	Memory Access	Description
31:1	Reserved_REMAP	R	RAP_REMAPMODE 31to1 Reserved bits **Values:** ■ 0x0 (BOOT_MODE): Boot mode memory map ■ 0x1 (NORMAL_MODE): Normal mode memory map **Value After Reset:** 0x0 **Exists:** Always
0	RAP_REMAPMODE	R/W	This bit controls the remap_n output, assuming that the remap functionality is set (REMAP = 1). The reset state of the remap_n output is 0, indicating that the boot mode memory map is active. Writing a 1 into bit 0 causes the DW_ahb address decoder to switch from the boot mode memory map to the normal mode memory map. This register bit can be reset to 0 only by a power-on reset. It is not possible to write a 0 into this register. If an attempt is made to write a 0, the register retains its previous state. Writing to, or reading from, this register when the remap feature is not selected (REMAP = 0) returns a value of 1'b1. **Value After Reset:** "(REMAP==1) ? 0 : 1" **Exists:** Always

5.1.4 RAP_RESETSTAT

- **Name:** Reset Status Register
- **Description:** This register specifies the reset status.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** Always

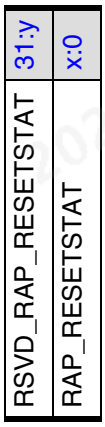


Table 5-9 Fields for Register: RAP_RESETSTAT

Bits	Name	Memory Access	Description
31:y	RSVD_RAP_RESETSTAT	R	RAP_RESETSTAT 31toRAP_NUM_RESETS Reserved bits **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** RAP_NUM_RESETS

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

Bits	Name	Memory Access	Description
x:0	RAP_RESETSTAT	R	This register contains the status of all the system resets. If an individual reset occurs, it asynchronously sets the corresponding internal register bit to 1. If 1 is read back from that bit location, it indicates that a reset of that type has occurred since the last time the reset status was cleared. A bit can be set only if the occurring reset sets it. A status bit exists for each reset. Each bit of this register corresponds to a bit of the sys_resets input. The polarity of each individual reset input can be configured using the RAP_RESET_POLn parameter. Reading from this register when the reset status feature is not selected (RAP_RESET = 0) has no effect. **Values:** ■ 0x0 (INACTIVE): Inactive ■ 0x1 (ACTIVE): Active **Value After Reset:** RAP_RESET_DFLT_VAL **Exists:** Always **Range Variable[x]:** RAP_NUM_RESETS - 1

5.1.5 RAP_CLRRESET

- **Name:** Clear Reset Status Register
- **Description:** This register specifies the clear reset status.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always

Table 5-10 Fields for Register: RAP_CLRRESET

Bits	Name	Memory Access	Description
31:y	RSVD_RAP_CLRRESET	W	RAP_CLRRESET 31toRAP_NUM_RESETS Reserved bits **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** RAP_NUM_RESETS
x:0	RAP_CLRRESET	W	These bits clear corresponding individual bits of the Reset Status register. When set to 0, these bits do not have any effect. **Values:** ■ 0x1 (CLEAR_RESET): Clears reset status **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** RAP_NUM_RESETS - 1

5.1.6 RAP_VER_ID

- **Name:** Component Version Register
- **Description:** This register specifies the component version.
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** Always

Table 5-11 Fields for Register: RAP_VER_ID

Bits	Name	Memory Access	Description
31:0	RAP_VER_ID	R	This bit specifies the ASCII value for each number in the version. **Value After Reset:** RAP_VERSION_ID **Exists:** Always

5.1.7 RAP_PING

- **Name:** Ping Test Simulation Register
- **Description:** This register specifies the ping test simulation.
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always

Table 5-12 Fields for Register: RAP_PING

Bits	Name	Memory Access	Description
31:1	RSVD_RAP_PING	R	RAP_PING 31to1 Reserved bits **Value After Reset:** 0x0 **Exists:** Always
0	RAP_PING	R/W	This bit specifies is a single read / write bit that can be used for subsystem connectivity tests. It does not perform any other function in the DW_apb_rap component. **Value After Reset:** 0x0 **Exists:** Always

6 Programming the DW_apb_rap

This chapter provides information on the programmable features of the DW_apb_rap.

6.1 Programming Considerations

The following sections give details about programming the DW_apb_rap.

6.1.1 Reset Status Register Operation

The Reset Status Register is a read-only register. It is cleared through a write-to-clear operation on Clear Reset Status Register. The reset inputs to the block asynchronously set the individual register bits, so it always takes precedence over a simultaneous write-to-clear. To ensure that the register status is cleared, it is recommended that a read operation occur after a write-to-clear operation in order to obtain the status. The write through the DW_apb does not stall the AHB bus, allowing AHB transfers to run.

Placing a read after a write forces the write to complete before any further transfers occur. The reset state of this register is all ones, which indicates that all resets have occurred. This means the user should do a write-to-clear on this register (through the Clear Reset Status Register) immediately after a power-on reset.

6.1.2 Remap Operation

The remap operation causes the system memory map to change. The APB does not hold up the AHB bus following a write operation; that is, the AHB bus can continue with other instructions while the APB actually does the write. Compounding this is the fact that the APB can operate at a much slower clock than the AHB, so the actual write to the Remap register may occur a number of cycles after the write is initiated on the AHB.

It takes another cycle before the resultant remap_n output signal is registered in the AHB bus itself, which actually causes the decoder to switch memory maps. Therefore this operation has to be handled carefully in software.

The following procedure is recommended:

1. Write a 1 to bit 0 (only bit) of the RemapMode register, to switch the memory map from boot to normal.
2. Put N IDLE cycles on the bus, where N = Freq(AHB) / Freq(APB) × 2 +1.
3. Read the RemapMode register to make sure the remap operation occurred.

7 Verification

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

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

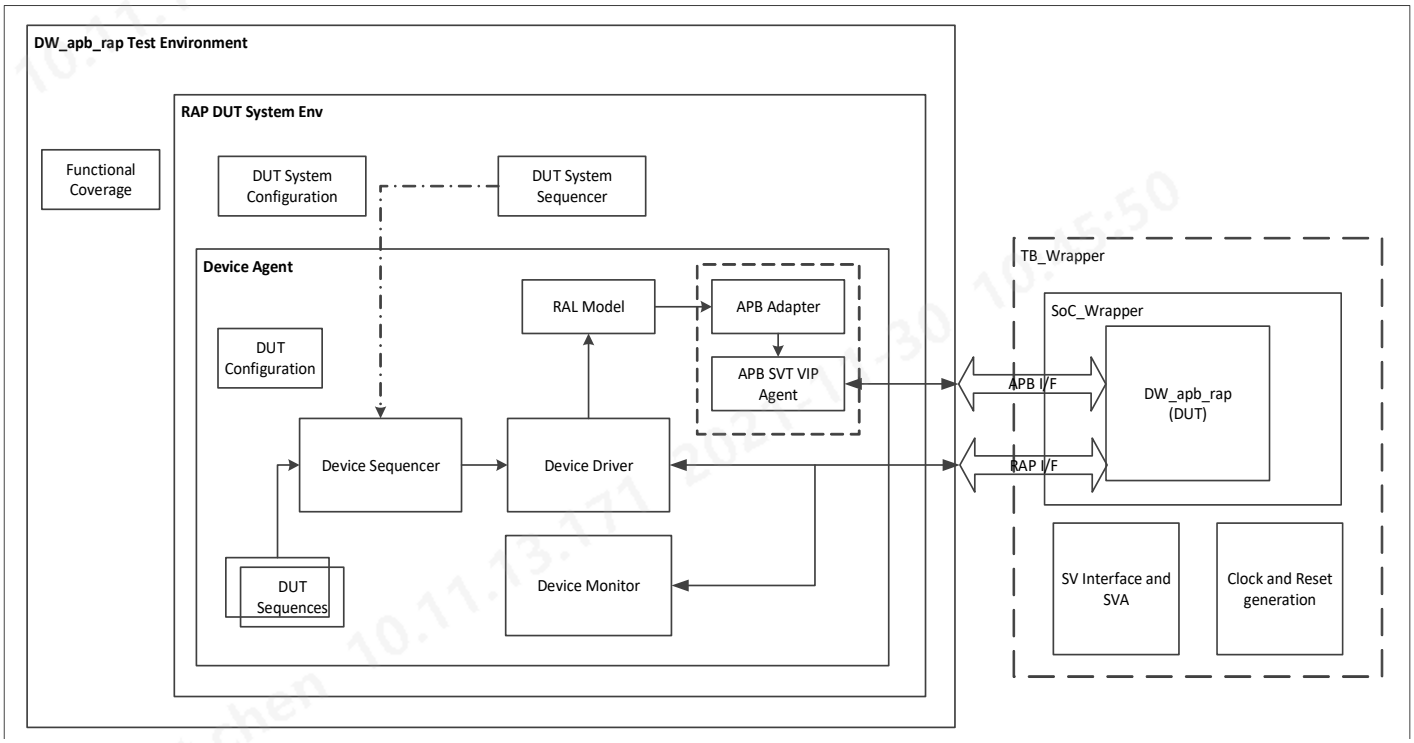
Note The packaged test benches are only for validating the IP configuration in coreConsultant GUI. It is not for system level validation.
IPs that have the Vera test bench packaged, these test benches are encrypted.

7.1 Overview of Testbench

The verification environment is SV-UVM based and makes use of the following verification IP to program the APB Slave interface:

- SVT APB VIP – For APB Interface

Figure 7-1 Verification Environment



The device agent is responsible to interact between the APB Interface and the RAL model present in the testbench. The device agent receives transfers from the Sequencer and schedules them for execution by driving the corresponding APB interface of the DW_apb_rap through RAL model and APB SVT VIP.

The RAP agent is responsible to drive and monitor the RAP related signals like pause, remap_n, and so on.

The reference model keeps track of the programming that is being done through the device agent and compares the actual activity through the information received from the RAP agent. The checker compares the whole activity and makes sure the integrity of the DUT.

7.2 Overview of Tests

The DW_apb_rap peripheral offers these functional features: remap mode, pause mode, and reset status. The DW_apb_rap peripheral also has an optional ID Code register and a coreKit version ID register. The tests have been written to exhaustively verify the functionality and have also achieved maximum RTL code coverage.

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

7.2.1 Remap Mode

Test Name: *test_remap_mode*

The Remap function is an optional feature of DW_apb_rap. This feature allows you to swap the system memory map between Boot Mode operation and Normal Mode operation. The remap_n output port is activated by a power-on reset and cleared by a write to the remap register.

These tests verify that the following requirements are implemented:

- Remap Mode can be activated only by a power-on reset.
- Remap Mode can be reset only by a write (1'b1) to the remap register.
- It is not possible to write (1'b0) into the 1-bit wide remap register.
- It is not possible to revert back into Remap Mode without powering down the system and rebooting.
- If the Remap feature is not selected, the remap register is not present in the design, and 1'b1 is returned for read accesses to the Remap address offset.

7.2.2 Pause Mode

Test Name: *test_pause_mode*

The Pause function is an optional feature of DW_apb_rap. This feature allows you to place the system into low power mode. The pause output port is activated by a write (1'b1) to the pause register and cleared by an active interrupt.

These tests verify that the following requirements are implemented:

- Pause Mode can be entered only by writing 1'b1 to the 1-bit pause register.
- Pause Mode can be reset only by an active FIQ or IRQ interrupt.
- Writing (1'b0) has no effect on the pause register.
- Pause Mode is reset at power-on reset.

7.2.3 Reset Status

Test Name: *test_resetstat*

The Reset Status function is an optional feature of DW_apb_rap. This feature allows you to monitor up to eight system resets. If any reset occurs, the corresponding bit in the reset status register is asynchronously set. The reset status register bits are cleared by writing (1'b1) to the corresponding bit position of the reset clear register.

These tests verify that the following requirements are implemented:

- The reset status register is a read-only register; writing to this register has no effect.
- The clear reset status register is a write-only register; reading this register returns zero.
- The reset signal is asserted according to polarity; and checked that the reset status register reflects the same.
- The reset status register bits can be cleared only by writing (1'b1) to corresponding bit position in the clear reset status register.

7.2.4 Register Access

Test Name: *test_reg_access*

This test aims at iterating over all valid registers and for each valid configuration check the proper register reset value, register access policy and register reset operation.

8 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment. The following sections discuss general integration considerations.

8.1 Accessing Top-level Constraints

To get SDC constraints out of coreConsultant, you need to first complete the synthesis activity and then use the "write_sdc" command to write out the results:

1. This cC command sets synthesis to write out scripts only, without running DC:

   ```
   set_activity_parameter Synthesize ScriptsOnly 1
   ```

2. This cC command autocompletes the activity:

   ```
   autocomplete_activity Synthesize
   ```

3. Finally, this cC command writes out SDC constraints:

   ```
   write_sdc <filename>
   ```

8.2 Coherency

Coherency is where bits within a register are logically connected. For instance, part of a register is read at time 1 and another part is read at time 2. Being coherent means that the part read at time 2 is at the same value it was when the register was read at time 1. The unread part is stored into a shadow register and this is read at time 2. When there is no coherency, no shadow registers are involved.

A bus master may need to be able to read the contents of a register, regardless of the data bus width, and be guaranteed of the coherency of the value read. A bus master may need to be able to write a register coherently regardless of the data bus width and use that register only when it has been fully programmed. This may need to be the case regardless of the relationship between the clocks.

Coherency enables a value to be read that is an accurate reflection of the state of the counter, independent of the data bus width, the counter width, and even the relationship between the clocks. Additionally, a value written in one domain is transferred to another domain in a seamless and coherent fashion.

Throughout this appendix the following terms are used:

- **Writing**. A bus master programs a configuration register. An example is programming the load value of a counter into a register.
- **Transferring**. The programmed register is in a different clock domain to where it is used, therefore, it needs to be transferred to the other clock domain.
- **Loading**. Once the programmed register is transferred into the correct clock domain, it needs to be loaded or used to perform its function. For example, once the load value is transferred into the counter domain, it gets loaded into the counter.

8.2.1 Writing Coherently

Writing coherently means that all the bits of a register can be written at the same time. A peripheral may have programmable registers that are wider than the width of the connected APB data bus, which prevents all the bits being programmed at the same time unless additional coherency circuitry is provided.

The programmable register could be the load value for a counter that may exist in a different clock domain. Not only does the value to be programmed need to be coherent, it also needs to be transferred to a different clock domain and then loaded into the counter. Depending on the function of the programmable register, a qualifier may need to be generated with the data so that it knows when the new value is currently transferred and when it should be loaded into the counter.

Depending on the system and on the register being programmed, there may be no need for any special coherency circuitry. One example that requires coherency circuitry is a 32-bit timer within an 8-bit APB system. The value is entirely programmed only after four 8-bit wide write transfers. It is safe to transfer or use the register when the last byte is currently written. An example where no coherency is required is a 16-bit wide timer within a 16-bit APB system. The value is entirely programmed after a single 16-bit wide write transfer.

Coherency circuitry enables the value to be loaded into the counter only when fully programmed and crossed over clock domains if the peripheral clock is not synchronous to the processor clock. While the load register is being programmed, the counter has access to the previous load value in case it needs to reload the counter.

Coherency circuitry is only added in cores where it is needed. The coherency circuitry incorporates an upper byte method that requires users to program the load register in LSB to MSB order when the peripheral width is smaller than the register width. When the upper byte is programmed, the value can be transferred and loaded into the load register. When the lower bytes are being programmed, they need to be stored in shadow registers so that the previous load register is available to the counter if it needs to reload. When the upper byte is programmed, the contents of the shadow registers and the upper byte are loaded into the load register.

The upper byte is the top byte of a register. A register can be transferred and loaded into the counter only when it has been fully programmed. A new value is available to the counter once this upper byte is written into the register. The following table shows the relationship between the register width and the peripheral bus width for the generation of the correct upper byte. The numbers in the table represent bytes, Byte 0 is the LSB and Byte 3 is the MSB. NCR means that no coherency circuitry is required, as the entire register is written with one access.

Table 8-1 Upper Byte Generation

	Upper Byte Bus Width		
Load Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	1	NCR	NCR
17 - 24	2	2	NCR
25 - 32	3	2 (or 3)	NCR

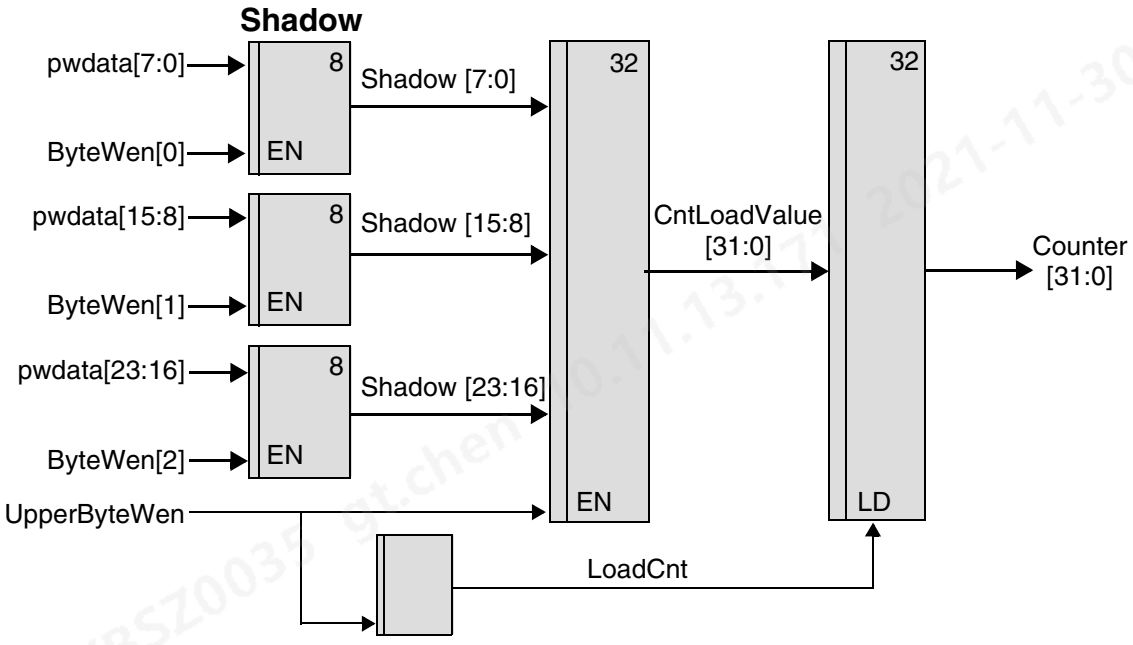
There are three relationship cases to be considered for the processor and peripheral clocks:

- Identical
- Synchronous (phase coherent but of an integer fraction)
- Asynchronous

8.2.1.1 Identical Clocks

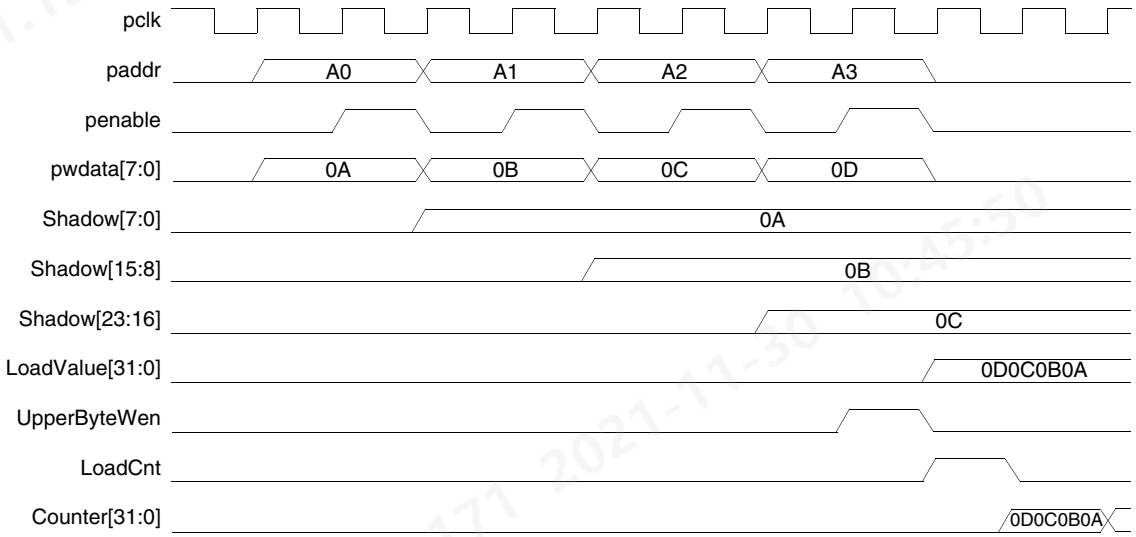
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 8-1 Coherent Loading – Identical Synchronous Clocks



The following figure shows a 32-bit register that is written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal lasts for one cycle and is used to load the counter with CntLoadValue.

Figure 8-2 Coherent Loading – Identical Synchronous Clocks



Each of the bytes that make up the load register are stored into shadow registers until the final byte is written. The shadow register is up to three bytes wide. The contents of the shadow registers and the final byte are transferred into the CntLoadValue register when the final byte is written. The counter uses this register to load/initialize itself. If the counter is operating in a periodic mode, it reloads from this register each time the count expires.

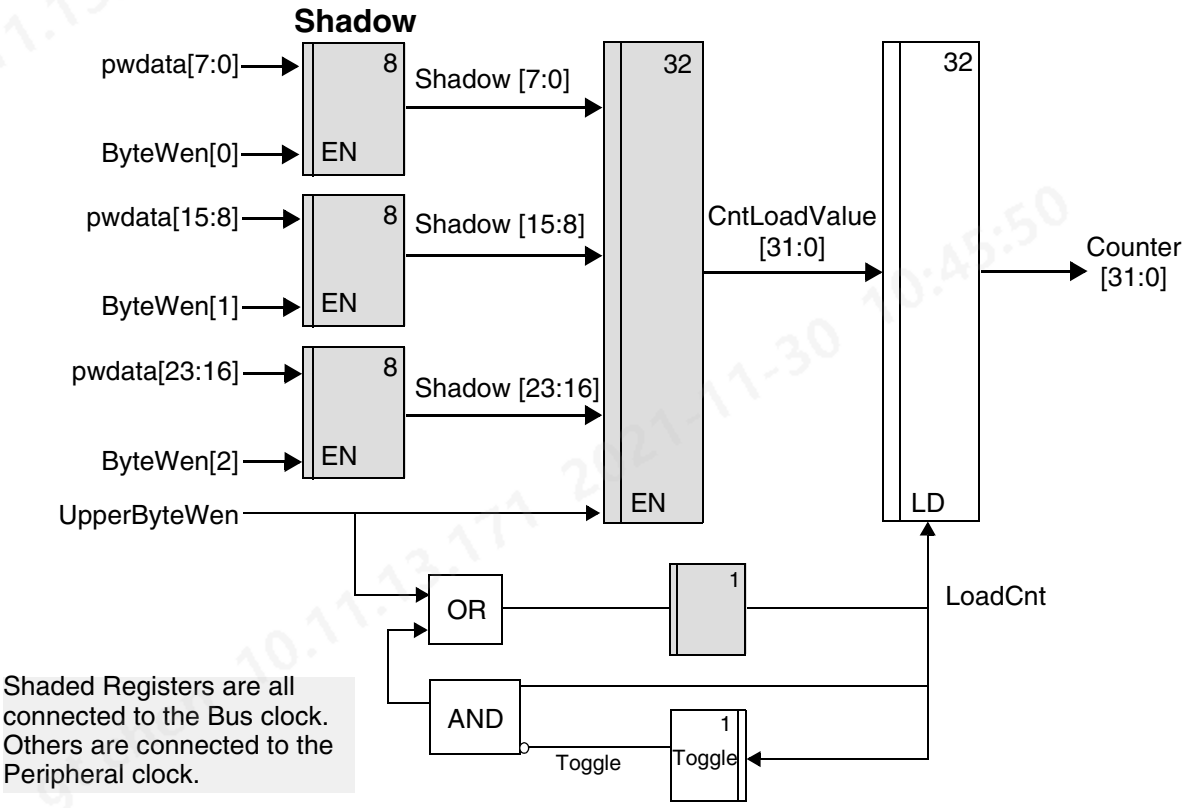
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this is signaled by LoadCnt = 1. After the upper byte is written, the LoadCnt goes to zero.

8.2.1.2 Synchronous Clocks

When the clocks are synchronous but do not have identical periods, the circuitry needs to be extended so that the LoadCnt signal is kept high until a rising edge of the counter clock occurs. This extension is necessary so that the value can be loaded, using LoadCnt, into the counter on the first counter clock edge. At the rising edge of the counter clock if LoadCnt is high, then a register clocked with the counter clock toggles, otherwise it keeps its current value. A circuit detecting the toggling is used to clear the original LoadCnt by looking for edge changes. The value is loaded into the counter when a toggle has been detected. Once it is loaded, the counter should be free to increment or decrement by normal rules.

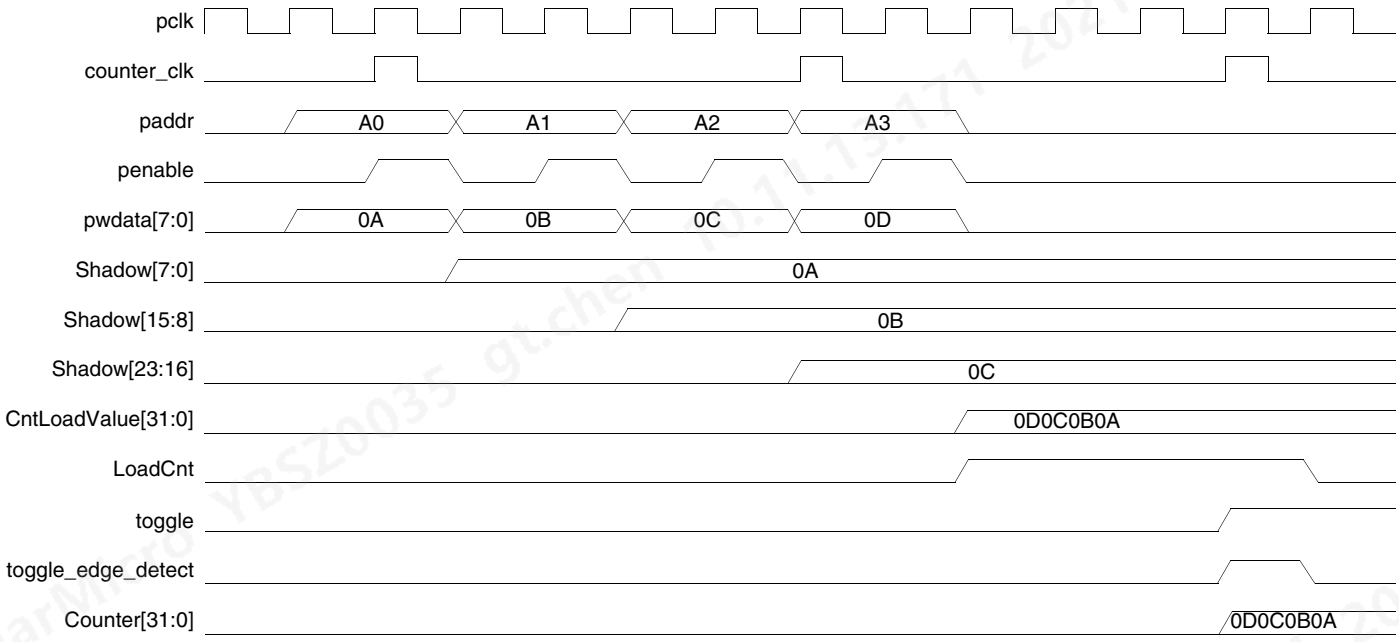
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 8-3 Coherent Loading – Synchronous Clocks



The following figure shows a 32-bit register being written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal is extended until a change in the toggle is detected and is used to load the counter.

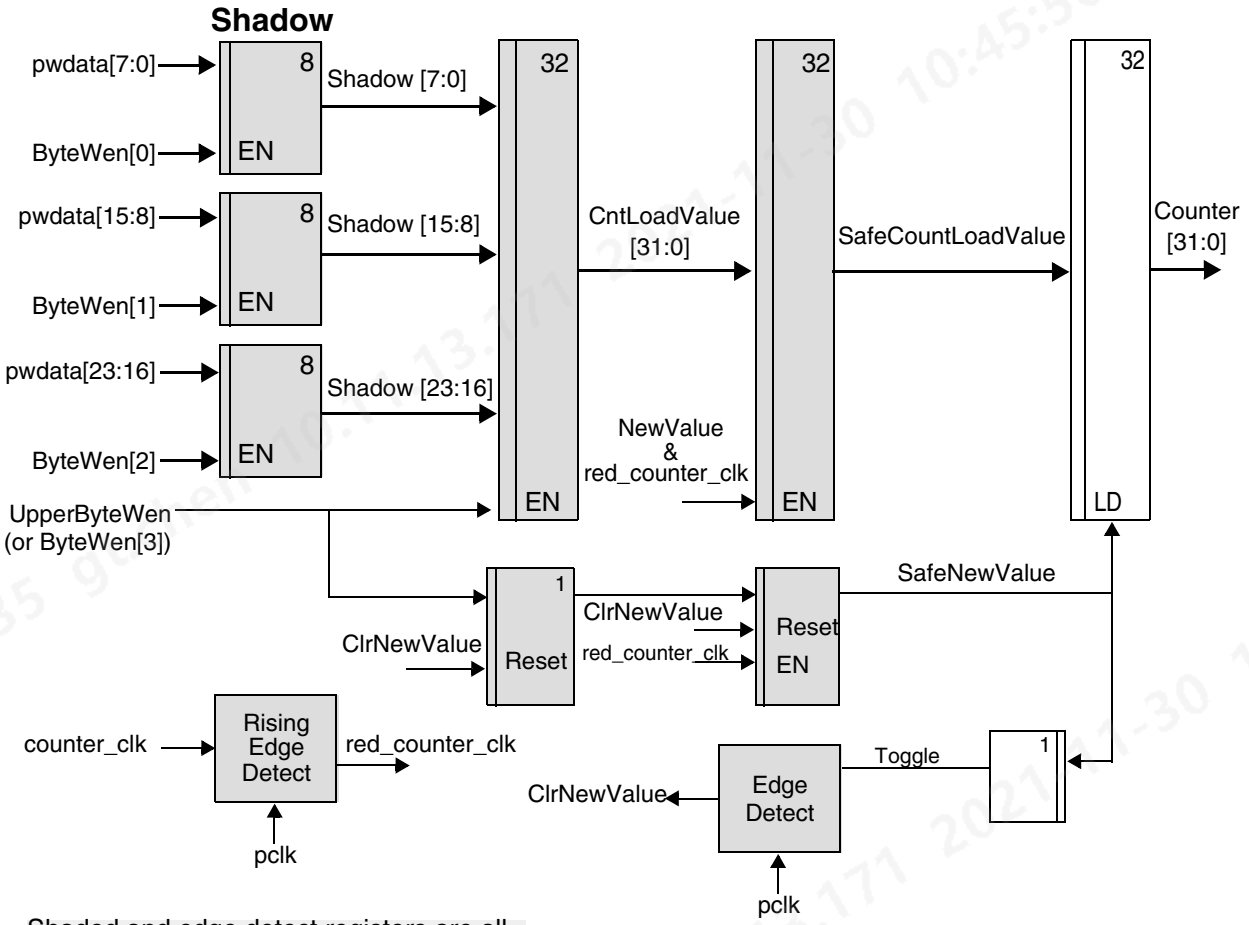
Figure 8-4 Coherent Loading – Synchronous Clocks



8.2.1.3 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three-times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock. The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are asynchronous.

Figure 8-5 Coherent Loading – Asynchronous Clocks



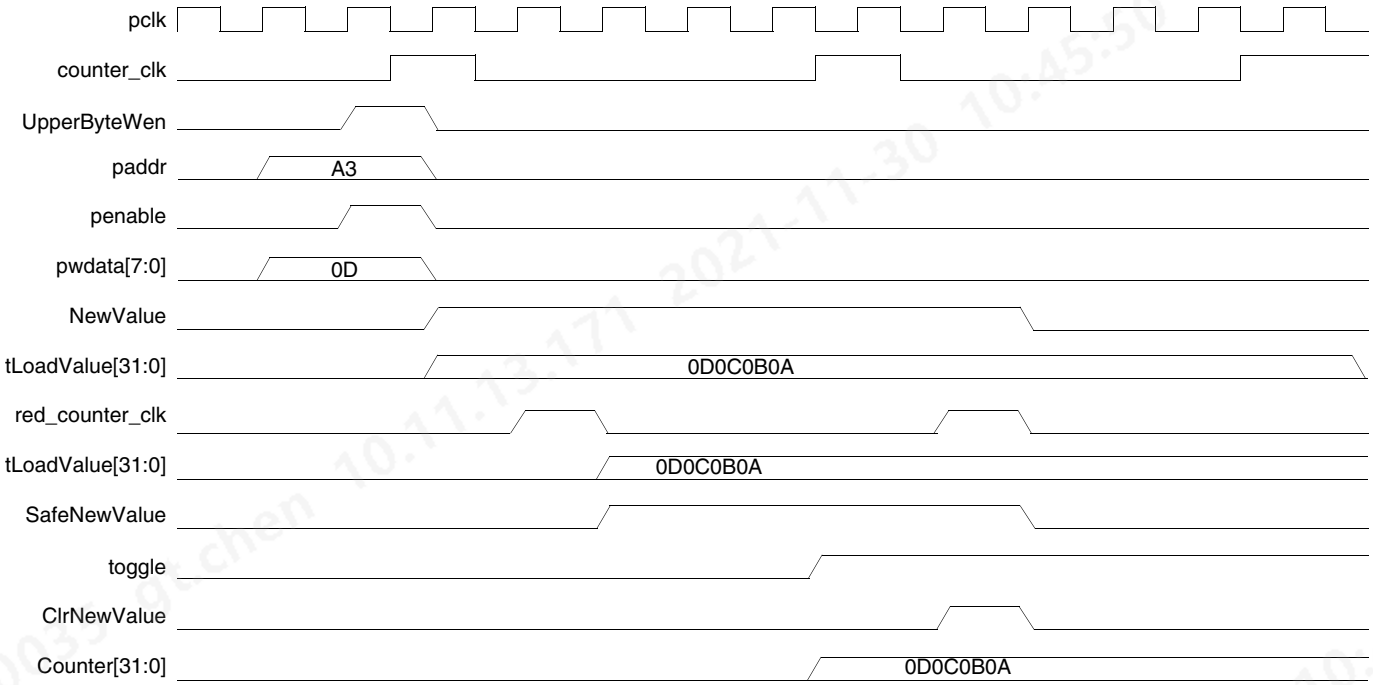
When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock to be used to re-time the peripheral clock. Upon a rising edge of the re-timed clock, the new value signal, NewValue, is transferred into a safe new value signal, SafeNewValue, which happens after the edge of the peripheral clock has occurred.

Every time there is a rising edge of the peripheral clock detected, the CntLoadValue is transferred into a SafeCntLoadValue. This value is used to transfer the load value across the clock domains. The SafeCntLoadValue only changes a number of bus clock cycles after the peripheral clock edge changes. A counter running on the peripheral clock is able to use this value safely. It could be up to two peripheral

clock periods before the value is loaded into the counter. Along with this loaded value, there also is a single bit transferred that is used to qualify the loading of the value into the counter.

The timing diagram depicted in the following figure does not show the shadow registers being loaded. This is identical to the loading for the other clock modes.

Figure 8-6 Coherent Loading – Asynchronous Clocks



The NewValue signal is extended until a change in the toggle is detected and is used to update the safe value. The SafeNewValue is used to load the counter at the rising edge of the peripheral clock. Each time a new value is written the toggle bit is flipped and the edge detection of the toggle is used to remove both the NewValue and the SafeNewValue.

8.2.2 Reading Coherently

For writing to registers, an upper-byte concept is proposed for solving coherency issues. For read transactions, a lower-byte concept is required. The following table provides the relationship between the register width and the bus width for the generation of the correct lower byte.

Table 8-2 Lower Byte Generation

	Lower Byte Bus Width		
Counter Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	0	NCR	NCR
17 - 24	0	0	NCR

Table 8-2 Lower Byte Generation

	Lower Byte Bus Width		
25 - 32	0	0	NCR

Depending on the bus width and the register width, there may be no need to save the upper bits because the entire register is read in one access, in which case there is no problem with coherency. When the lower byte is read, the remaining upper bytes within the counter register are transferred into a holding register. The holding register is the source for the remaining upper bytes. Users must read LSB to MSB for this solution to operate correctly. NCR means that no coherency circuitry is required, as the entire register is read with one access.

There are two cases regarding the relationship between the processor and peripheral clocks to be considered as follows:

- Identical and/or synchronous
- Asynchronous

8.2.2.1 Synchronous Clocks

When the clocks are identical and/or synchronous, the remaining unread bits (if any) need to be saved into a holding register once a read is started. The first read byte must be the lower byte provided in the previous table, which causes the other bits to be moved into the holding register, SafeCntVal, provided that the register cannot be read in one access. The upper bytes of the register are read from the holding register rather than the actual register so that the value read is coherent. This is illustrated in the following figure and in the timing diagram after it.

Figure 8-7 Coherent Registering – Synchronous Clocks

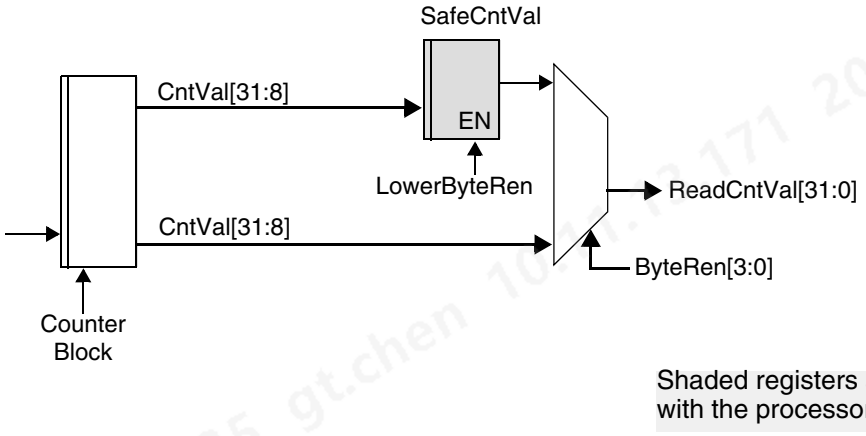
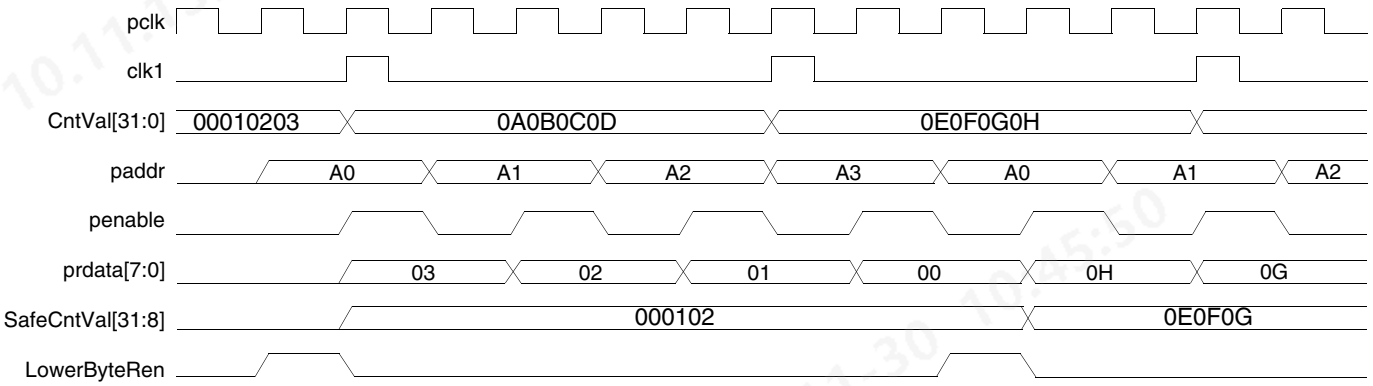


Figure 8-8 Coherent Registering – Synchronous Clocks



8.2.2.2 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock.

To safely transfer a counter value from the counter clock domain to the bus clock domain, the counter clock signal should be transferred to the bus clock domain. When the rising edge detect of this re-timed counter clock signal is detected, it is safe to use the counter value to update a shadow register that holds the current value of the counter.

While reading the counter contents it may take multiple APB transfers to read the value.

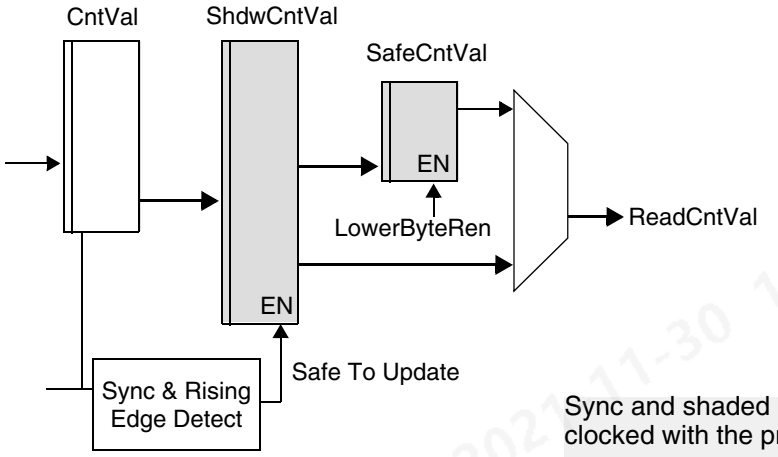
Note You must read LSB to MSB when the bus width is narrower than the counter width.

Once a read transaction has started, the value of the upper register bits need to be stored into a shadow register so that they can be read with subsequent read accesses. Storing these upper bits preserves the coherency of the value that is being read. When the processor reads the current value it actually reads the contents of the shadow register instead of the actual counter value. The holding register is read when the bus width is narrower than the counter width. When the LSB is read, the value comes from the shadow register; when the remaining bytes are read they come from the holding register. If the data bus width is wide enough to read the counter in one access, then the holding registers do not exist.

The counter clock is registered and successively pipelined to sense a rising edge on the counter clock. Having detected the rising edge, the value from the counter is known to be stable and can be transferred into the shadow register. The coherency of the counter value is maintained before it is transferred, because the value is stable.

The following figure illustrates the synchronization of the counter clock and the update of the shadow register.

Figure 8-9 Coherency and Shadow Registering – Asynchronous Clocks



8.3 Performance

8.3.1 Power Consumption, Frequency, Area and DFT Coverage

Table 8-3 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_apb_rap using the industry standard 7nm technology library.

Table 8-3 Synthesis Results for DW_apb_rap

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	pclk= 100 MHz	275	0.7 nW	0.002 mW	100	100	98
Typical Configuration APB_DATA_WIDTH = 32 PAUSE = 1 RAP_HAS_FIQ = 1 RAP_INT_POL = 1 REMAP = 1 RAP_ID = 1 RAP_ID_WIDTH = 32 RAP_ID_NUM = 0x12345678 RAP_RESET = 1 RAP_NUM_RESETS = 8 RAP_RESET_POL_0 = 1 RAP_RESET_POL_1 = 0 RAP_RESET_POL_2 = 1 RAP_RESET_POL_3 = 0 RAP_RESET_POL_4 = 1 RAP_RESET_POL_5 = 0 RAP_RESET_POL_6 = 1 RAP_RESET_POL_7 = 0	pclk= 100 MHz	306	0.8 nW	0.002 mW	100	100	96.9

A
Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

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

Table A-1 Internal Parameters

Parameter Name	Equals To
RAP_ADDR_SLICE_LHS	4
RAP_RESET_DFLT_VAL	=(RAP_RESET ==1 ? [::DW_apb_rap::calc_2pow_value RAP_NUM_RESETS] : 0)
RAP_VERSION_ID	32'h3230392a

B Glossary

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

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

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