DesignWare DW_apb_wdt Databook

DW_apb_wdt

1.07c
March 2012

Contents

Preface

This databook provides information that you need to interface the DesignWare APB Watchdog Timer, referred to as DW_apb_wdt throughout the remainder of this databook. It is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the *AMBA Specification, Revision 2.0* from ARM.

The information in this databook includes a functional description, signal and parameter descriptions, and a memory map. Also provided are an overview of the component testbench, a description of the tests that are run to verify the coreKit, and synthesis information for the coreKit.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Building and Verifying a Component or Subsystem" introduces you to using the DW_apb_wdt within the coreAssembler and coreConsultant tools.
- Chapter 3, "Functional Description" describes the functional operation of the DW_apb_wdt.
- Chapter 4, "Parameters" identifies the configurable parameters supported by the DW_apb_wdt.
- Chapter 5, "Signals" provides a list and description of the DW_apb_wdt signals.
- Chapter 6, "Registers" describes the programmable registers of the DW_apb_wdt.
- Chapter 7, "Programming the DW_apb_wdt" provides information needed to program the configured DW_apb_wdt.
- Chapter 8, "Verification" provides information on verifying the configured DW_apb_wdt.
- Chapter 9, "Integration Considerations" includes information you need to integrate the configured DW_apb_wdt into your design.
- Appendix A, "Glossary" provides a glossary of general terms.

Related Documentation

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

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

Document Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.03d onward.

Table 1-1 Databook Revision History

Version	Databook Date	Description
1.07c	Mar 2012	Version change for 2012.03a release.
1.07b	Nov 2011	Version change for 2011.11a release.
1.07a	Oct 2011	Added material for WDT_ASYNC_CLK_MODE_ENABLE parameter.
1.06a	Jun 2011	Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface.
1.06a	Sep 2010	Corrected Figure 7-1; corrected names of include files and vcs command used for simulation
1.04a	May 2010	Corrected lack of “no” branch on first counter of Operation Flow diagram.
1.04a	Dec 2009	Clarified wdt_sys_rst propagation in “Functional Description”; corrected width of paddr signal; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.04a	May 2008	Removed references to QuickStarts, as they are no longer supported.
1.04a	Oct 2008	Version change for 2008.10a release.
1.03e	Jun 2008	Version change for 2008.06a release.
1.03d	Jan 2008	Updated to revised installation guide and consolidated release notes; changed references of “Designware AMBA” to simply “DesignWare.”
1.03d	Jun 2007	Version change for 2007.06a release.

Web Resources

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

Customer Support

To obtain support for your product:

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

    File > Build Debug Tar-file

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

    Go to http://solvnet.synopsys.com/EnterACall and click on the link to enter a call. Provide the requested information, including:
    - **Product:** DesignWare Library IP
    - **Sub Product:** AMBA
    - **Tool Version:** *product version number*
    - **Problem Type:**
    - **Priority:**
    - **Title:** DW_apb_wdt
    - **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

    After creating the case, attach any debug files you created in the previous step.
  - Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
    - Include the Product name, Sub Product name, and Tool Version number in your e-mail (as identified above) so it can be routed correctly.
    - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
    - Attach any debug files you created in the previous step.

- Or, telephone your local support center:
  - North America:
    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:
    http://www.synopsys.com/Support/GlobalSupportCenters

1 Product Overview

The DW_apb_wdt is a programmable Watchdog Timer (WDT) 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.

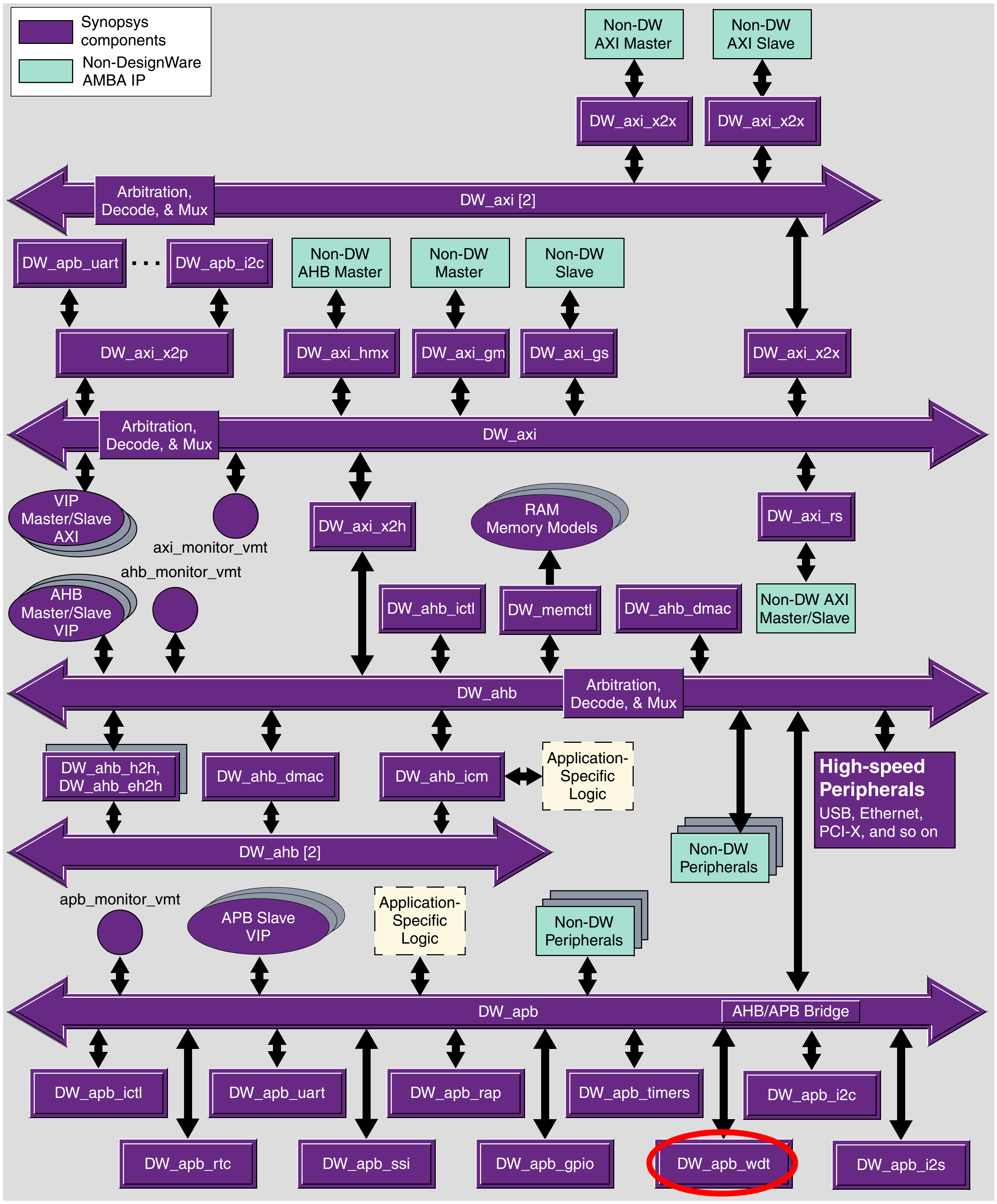
1.1 DesignWare System Overview

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

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

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

Figure 1-1 Example of DW_apb_wdt in a Complete System



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

1.2 General Product Description

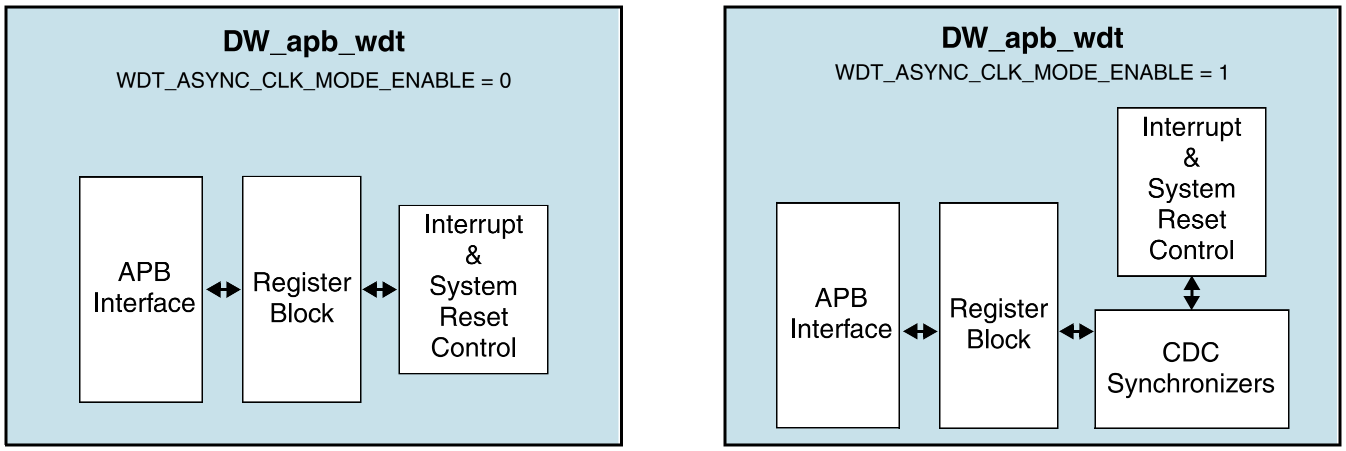
The Synopsys DW_apb_wdt is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the *AMBA Specification, Revision 2.0* from ARM.

1.2.1 DW_apb_wdt Block Diagram

Figure 1-2 shows the following functional groupings of the main interfaces to the DW_apb_wdt block:

- An APB slave interface
- A register block with read coherency for the current count register
- An interrupt/system reset generation block comprising of a decrementing counter and control logic
- Clock domain crossing synchronizers required for asynchronous timer clock support; present only when WDT_ASYNC_CLK_MODE_ENABLE = 1

Figure 1-2 Block Diagram of DW_apb_wdt


DW_apb_wdt
WDT_ASYNC_CLK_MODE_ENABLE = 0
APB Interface
Register Block
Interrupt & System Reset Control
DW_apb_wdt
WDT_ASYNC_CLK_MODE_ENABLE = 1
Interrupt & System Reset Control
APB Interface
Register Block
CDC Synchronizers

1.3 Features

The DW_apb_wdt supports the following features:

- APB interface used to allow easy integration into DesignWare Synthesizable Components for AMBA 2 implementations.
- Configurable APB data bus widths of 8, 16, and 32 bits.
- Configurable watchdog counter width of 16 to 32 bits.
- Counter counts down from a preset value to 0 to indicate the occurrence of a timeout.

- Optional external clock enable signal to control the rate at which the counter counts.
- If a timeout occurs the DW_apb_wdt can perform one of the following operations:
  - Generate a system reset
  - First generate an interrupt and if this is not cleared by the service routine by the time a second timeout occurs then generate a system reset
- Programmable timeout range (period). The option of hard coding this value during configuration is available to reduce the register requirements.
- Optional dual programmable timeout period, used when the duration waited for the first kick is different than that required for subsequent kicks. The option of hard coding these values is available.
- Programmable and hard coded reset pulse length.
- Prevention of accidental restart of the DW_apb_wdt counter.
- Prevention of accidental disabling of the DW_apb_wdt.
- Optional support for Pause mode with the use of external pause enable signal.
- Test mode signal to decrease the time required during functional test.
- Optional support for asynchronous external timer clock. With this feature enabled, the timer interrupt and system reset can be generated, even when the APB bus clock is switched off.

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

1.4 Standards Compliance

The DW_apb_wdt 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_wdt 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 65 section discusses the specific procedures for verifying the DW_apb_wdt.

1.6 Licenses

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

1.7 Where To Go From Here

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

For more information about configuring, synthesizing, and verifying just your DW_apb_wdt component, refer to "Overview of the coreConsultant Configuration and Integration Process" on page 18.

For more information about implementing your DW_apb_wdt component within a DesignWare subsystem using coreAssembler, refer to "Overview of the coreAssembler Configuration and Integration Process" on page 21.

2 Building and Verifying a Component or Subsystem

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

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

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

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

2.1 Setting up Your Environment

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

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

2.2 Overview of the coreConsultant Configuration and Integration Process

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

2.2.1 coreConsultant Usage

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

Figure 2-1 coreConsultant Usage Flow

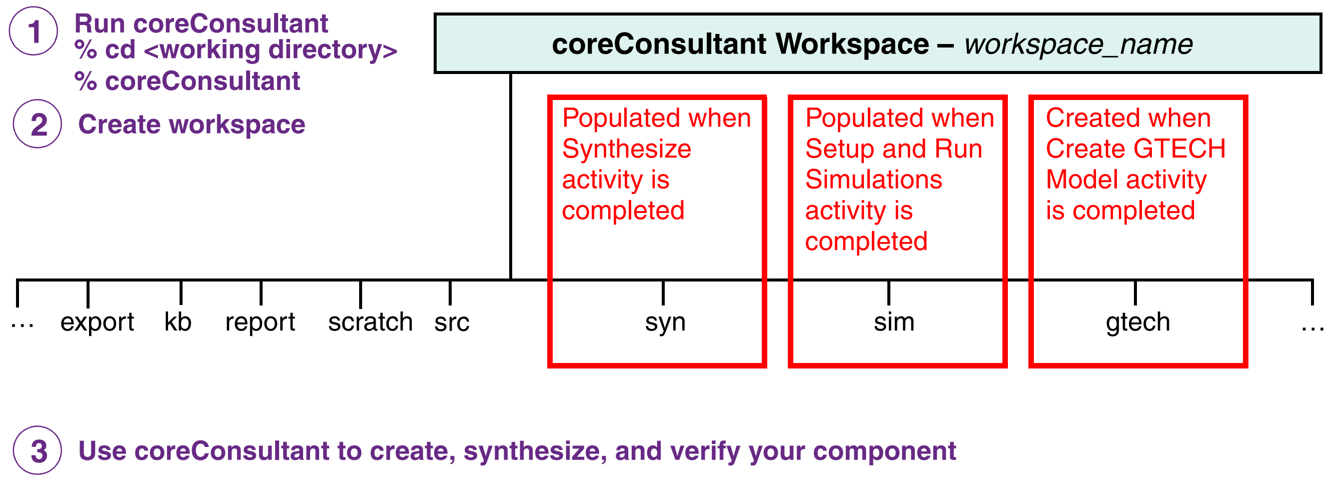

1 Run coreConsultant
% cd <working directory>
% coreConsultant
2 Create workspace
coreConsultant Workspace – workspace_name
Populated when Synthesize activity is completed
Populated when Setup and Run Simulations activity is completed
Created when Create GTECH Model activity is completed
…
export
kb
report
scratch
src
syn
sim
gtech
…
3 Use coreConsultant to create, synthesize, and verify your component

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

Table 2-1 coreConsultant Implementation Workspace Directory Contents

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

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

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

For details on some key files created during coreConsultant activities, refer to "Database Files" on page 25.

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

2.2.2 Configuring the DW_apb_wdt within coreConsultant

The "Parameters" chapter on page 35 describes the DW_apb_wdt hardware configuration parameters that you configure using the coreConsultant GUI.

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

2.2.3 Creating Gate-Level Netlists within coreConsultant

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

2.2.4 Verifying the DW_apb_wdt within coreConsultant

The "Verification" chapter on page 65 provides an overview of the testbench available for DW_apb_wdt verification using the coreConsultant GUI.

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

2.2.5 Running Leda on Generated Code with coreConsultant

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

2.3 Overview of the coreAssembler Configuration and Integration Process

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

2.3.1 coreAssembler Usage

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

Figure 2-2 coreAssembler Usage Flow

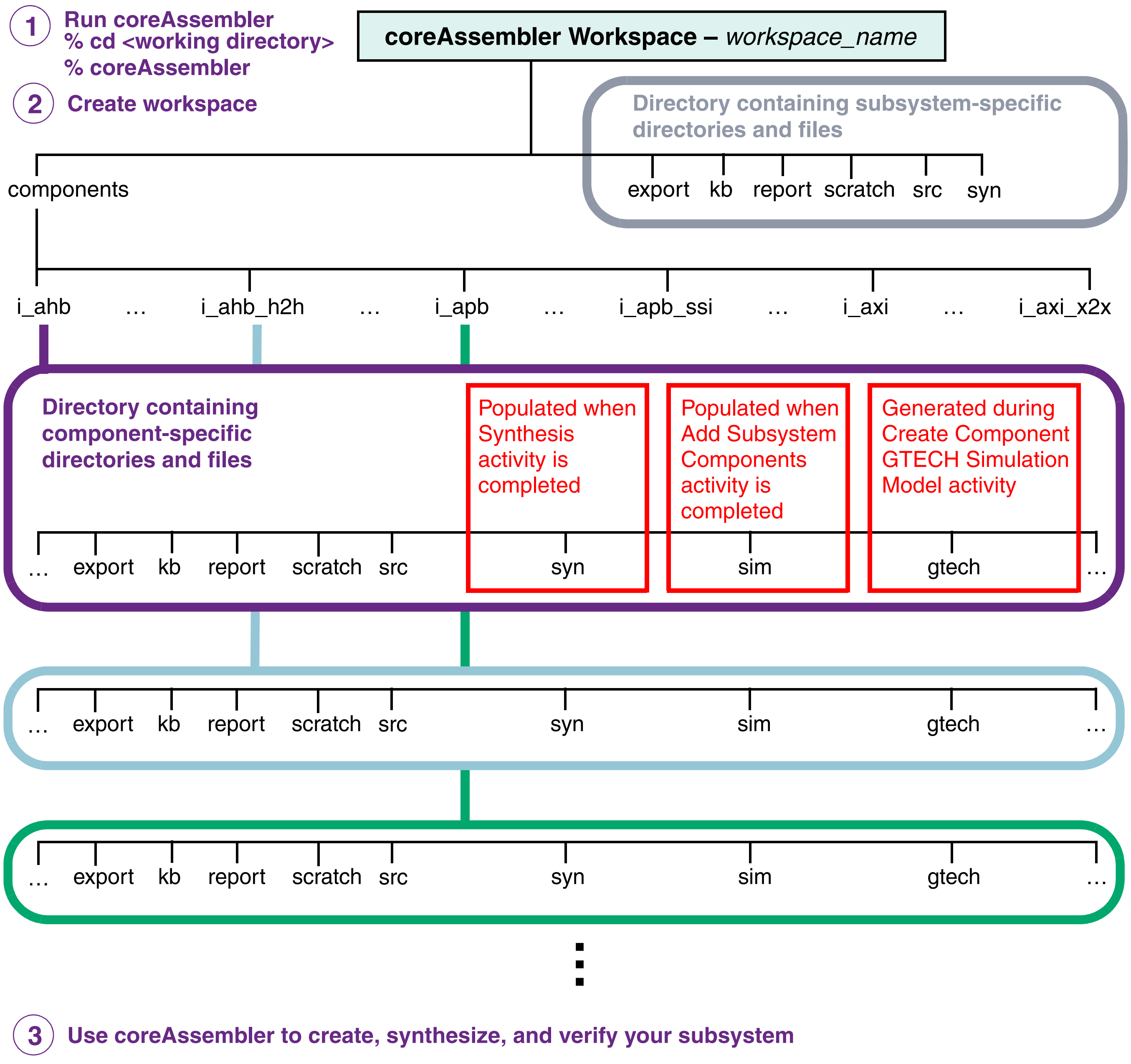

1 Run coreAssembler
% cd <working directory>
% coreAssembler
2 Create workspace
coreAssembler Workspace – *workspace_name*
Directory containing subsystem-specific directories and files
export kb report scratch src syn
components
i_ahb … i_ahb_h2h … i_apb … i_apb_ssi … i_axi … i_axi_x2x
Directory containing component-specific directories and files
Populated when Synthesis activity is completed
Populated when Add Subsystem Components activity is completed
Generated during Create Component GTECH Simulation Model activity
… export kb report scratch src syn sim gtech …
3 Use coreAssembler to create, synthesize, and verify your subsystem

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

Table 2-2 coreAssembler Implementation Workspace Directory Contents

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

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

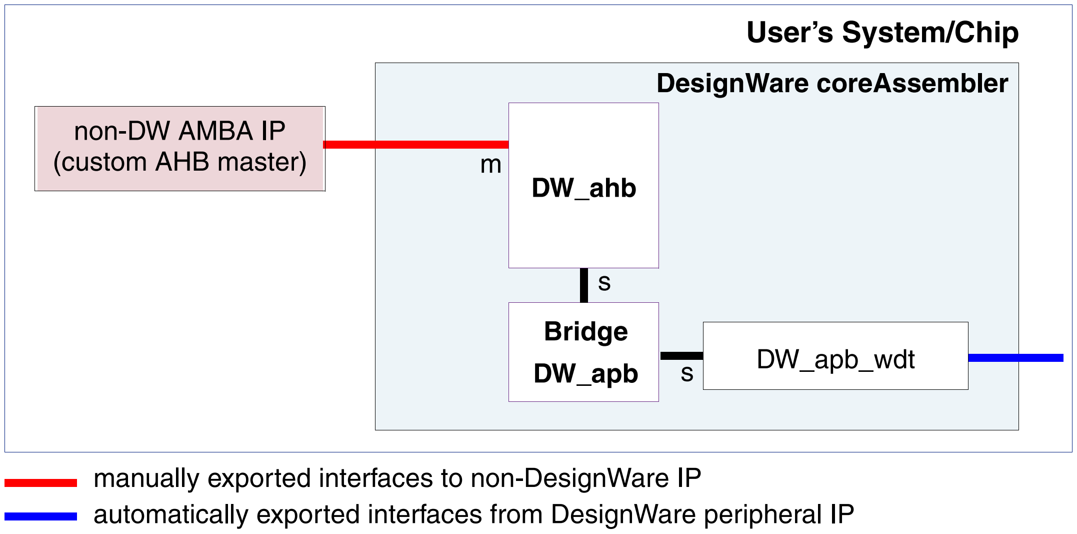
Directory/Subdirectory	Description
i_*component*/src	Includes the top-level RTL file, *design_name*.v. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated during Add Subsystem Components activity; populated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

For details on some key files created during coreAssembler activities, refer to "Database Files" on page 25.

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

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

Figure 2-3 DW_apb_wdt in Simple Subsystem


User's System/Chip
DesignWare coreAssembler
non-DW AMBA IP (custom AHB master)
m
DW_ahb
s
Bridge DW_apb
s
DW_apb_wdt
manually exported interfaces to non-DesignWare IP
automatically exported interfaces from DesignWare peripheral IP

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

- DW_apb_wdt
- DW_ahb
- DW_apb
- AHB Master

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

2.3.2 Configuring the DW_apb_wdt within a Subsystem

The "Parameters" chapter on page 35 describes the DW_apb_wdt hardware configuration parameters that you configure using the coreAssembler GUI. Corresponding databooks for the other components in a subsystem contain "Parameters" chapters that describe their respective configuration parameters.

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

2.3.3 Creating Gate-Level Netlists within coreAssembler

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

2.3.4 Verifying the DW_apb_wdt within coreAssembler

The "Verification" chapter on page 65 provides an overview of the testbench available for DW_apb_wdt verification using the coreAssembler GUI.

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

2.3.5 Running Leda on Generated Code with coreAssembler

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

2.4 Database Files

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

2.4.1 Design/HDL Files

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

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

2.4.1.1 RTL-Level Files

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

Table 2-3 RTL-Level Files

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

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

./src/*component_submodule*.v	Yes	Sub-modules of component
./src/*component*_constants.v	No	Includes the constants used internally in the design.

Table 2-3 **RTL-Level Files (Continued)**

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

2.4.1.2 Simulation Model Files

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

Table 2-4 **Simulation Model Files**

Files	Encrypted?	Purpose

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

2.4.2 Synthesis Files

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

Table 2-5 **Synthesis Files**

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

2.4.3 Verification Reference Files

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

Table 2-6 Verification Reference Files

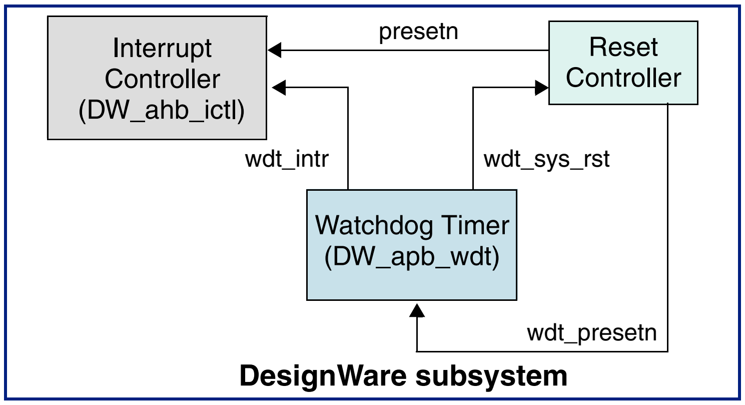
Files	Encrypted?	Purpose
./sim/runtest	No	Perl script that runs the Setup and Run Simulations activity from the command line.
./sim/runtest.log	No	The overall result of simulation, including pass/fail results.
./sim/test_*testname*/test.result	No	Pass/fail of individual test.
./sim/test_*testname*/test.log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

3 Functional Description

This chapter describes the functional operation of the DW_apb_wdt.

The DW_apb_wdt is an APB slave peripheral that can be used to prevent system lockup that may be caused by conflicting parts or programs in an SoC. This component can be configured, synthesized, and programmed based on user-defined options. An example of the DW_apb_wdt peripheral used in a system is illustrated in Figure 3-1.

Figure 3-1 Example of DW_apb_wdt in a DesignWare System



The generated interrupt is passed to an interrupt controller. The generated reset is passed to a reset controller, which in turn generates a reset for the components in the system. The WDT may be reset independently to the other components.

Note Propagating the generated reset output (wdt_sys_rst) back into the presetn input—for example, through the external reset controller—could result in truncated wdt_sys_rst assertion. When the presetn is asserted, the wdt_sys_rst output is immediately de-asserted (asynchronously reset). You can avoid truncation by the external reset controller by either not resetting the DW_apb_wdt after a wdt_sys_rst assertion or waiting for wdt_sys_rst to de-assert before resetting the DW_apb_wdt.

3.1 Counter

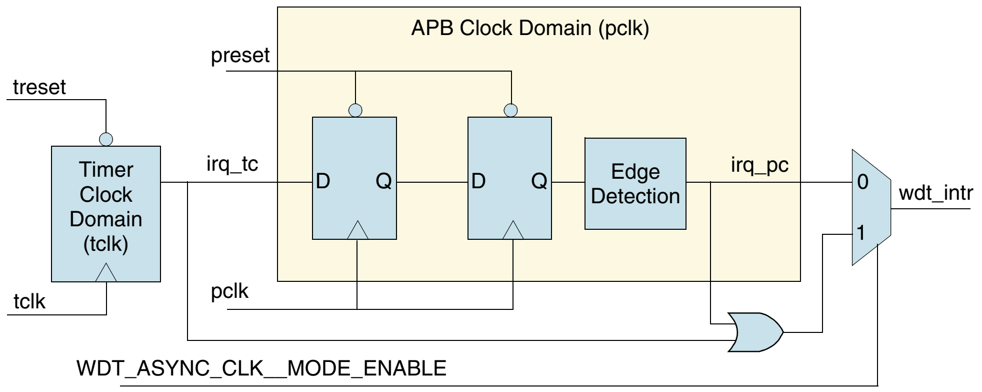
The DW_apb_wdt counts from a preset (timeout) value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. When the counter reaches zero, it wraps to the selected timeout value and continues decrementing. The user can restart the counter to its initial value. This is programmed by writing to the restart register at any time. The process of restarting the watchdog counter is sometimes referred to as *kicking the dog*. As a safety feature to prevent accidental restarts, the value 0x76 must be written to the Current Counter Value Register (WDT_CRR).

3.2 Interrupts

The DW_apb_wdt can be programmed to generate an interrupt (and then a system reset) when a timeout occurs. When a 1 is written to the response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the DW_apb_wdt generates an interrupt. If it is not cleared by the time a second timeout occurs, then it generates a system reset. If a restart occurs at the same time the watchdog counter reaches zero, an interrupt is not generated.

Figure 3-2 hows how the top-level Watchdog Timer Interrupt (wdt_intr) is generated based on the value of WDT_ASYNC_CLK_MODE_ENABLE. When the DW_apb_wdt is configured to support the asynchronous timer clock (WDT_ASYNC_CLK_MODE_ENABLE = 1), there are two clock domains in the DW_apb_wdt design; that is, the timer clock domain (tclk) and the APB clock domain (pclk). Thus, interrupt generation also depends on the value configured for the WDT_ASYNC_CLK_MODE_ENABLE parameter.

Figure 3-2 Timer Interrupt Generation



When WDT_ASYNC_CLK_MODE_ENABLE = 0, wdt_intr is asserted only after the internal timer clock domain interrupt signal (irq_tc) is edge-detected in the pclk domain (irq_pc). Therefore in this configuration, the WDT interrupt cannot be generated when pclk is off.

When WDT_ASYNC_CLK_MODE_ENABLE = 1, wdt_intr is asserted along with the internal timer clock domain interrupt (irq_tc) when the timer counter rolls over to its maximum value. The timer clock domain interrupt is cleared internally once it is synchronized and edge-detected in the pclk domain (irq_pc).

The timer clock domain interrupt and the edge-detected APB clock domain interrupt are ORed together in order to generate the final interrupt output (wdt_intr). Therefore, wdt_intr asserts as soon as the timer counter rolls over to its maximum value, and it stays asserted until it is cleared from the pclk domain by either a read of the WDT_EOI register or by writing 0x76 to the WDT_CRR register (watchdog counter restart).

This logic allows the watchdog timer interrupt to be generated, even when pclk is disabled. The wdt_intr remains asserted until pclk is restarted and the interrupt is serviced.

Therefore, with this configuration, it is not necessary to have the system clock (pclk) active in order to detect a watchdog timer interrupt.

Figure 3-3 shows the timing diagram of the interrupt being generated and cleared when WDT_ASYNC_CLK_MODE_ENABLE = 0.

Figure 3-3 Interrupt Generation When WDT_ASYNC_CLK_MODE_ENABLE = 0

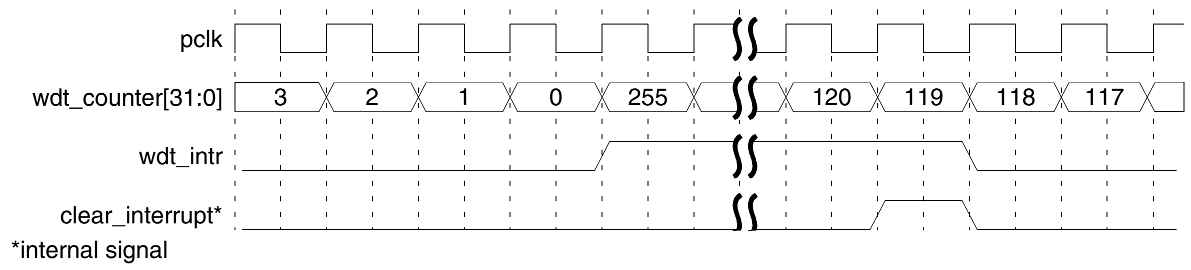
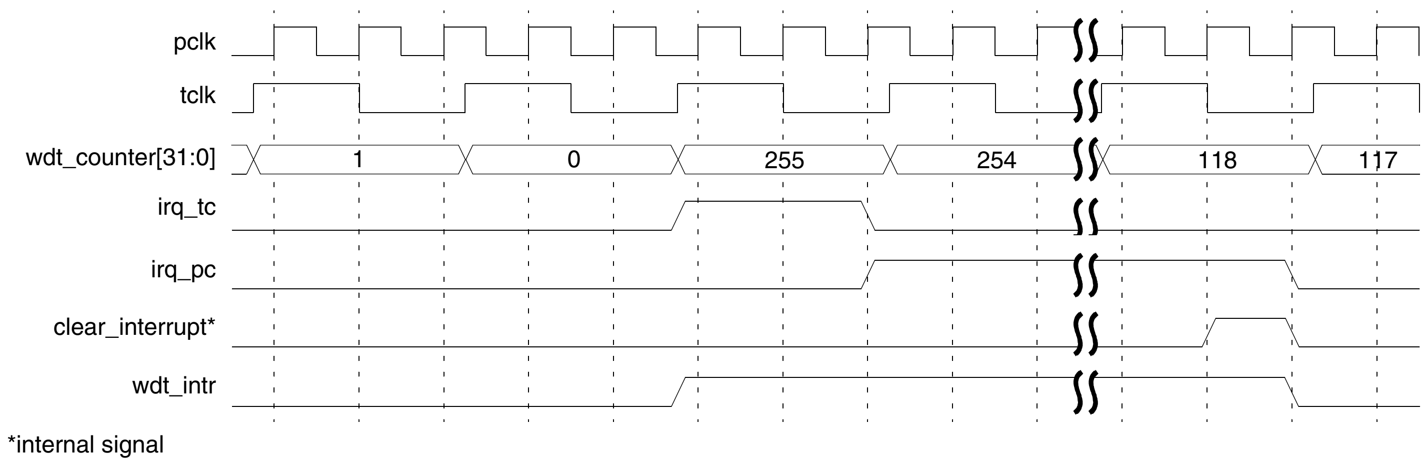


Figure 3-4 shows the timing diagram of the interrupt being generated and cleared when WDT_ASYNC_CLK_MODE_ENABLE = 1.

Figure 3-4 Interrupt Generation When WDT_ASYNC_CLK_MODE_ENABLE = 1



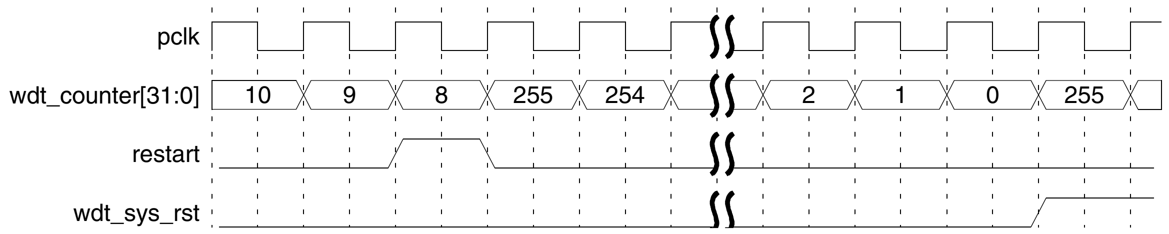
Note The wdt_intr interrupt, generated when WDT_ASYNC_CLK_MODE_ENABLE = 1, is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

3.3 System Resets

When a 0 is written to the output response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the DW_apb_wdt generates a system reset when a timeout occurs. The WDT can be configured so that it is always enabled upon reset of the DW_apb_wdt. If this is the case, it overrides whatever has been written in bit 0 of the WDT_CR register (the WDT enable field).

Figure 3-5 shows the timing diagram of a counter restart and the generation of a system reset.

Figure 3-5 Counter Restart and System Reset



If a restart occurs at the same time the watchdog counter reaches zero, a system reset is not generated.

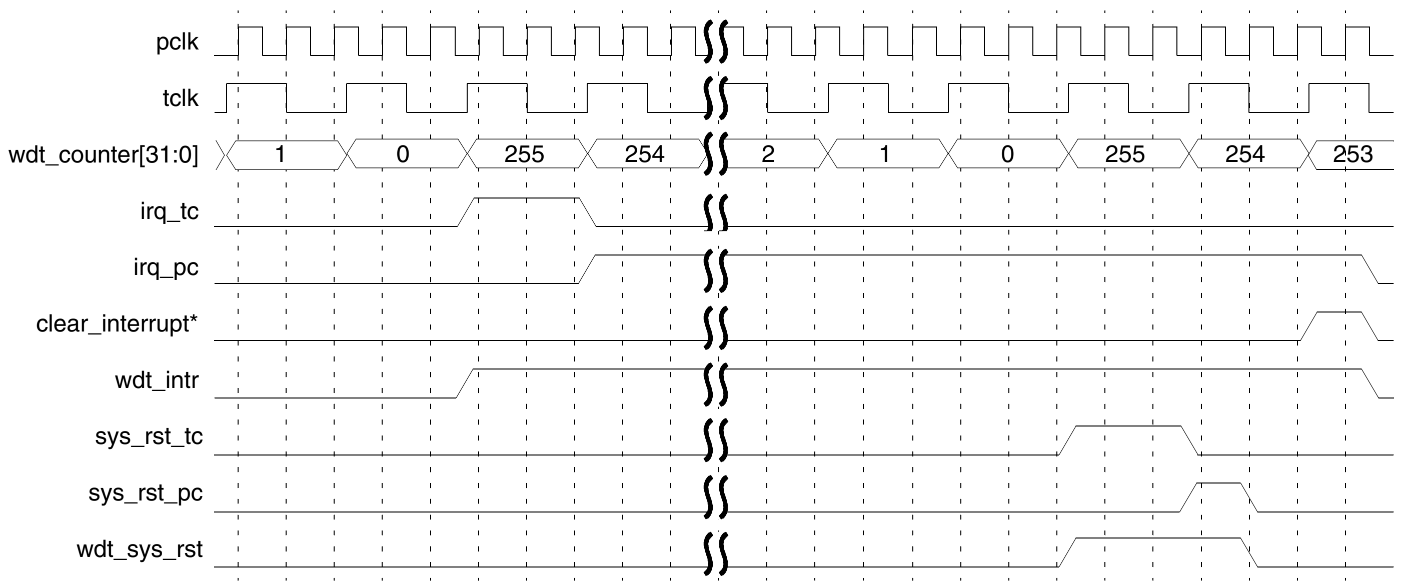
When WDT_ASYNC_CLK_MODE_ENABLE = 1, the system reset is generated in the timer clock domain (sys_rst_tc) and synchronized over to the pclk domain. Once it is edge-detected in the pclk domain (sys_rst_pc), the timer domain reset is cleared. The final output wdt_sys_rst is the output of an OR of the timer clock domain reset and edge-detected pclk domain reset. It remains asserted for a duration as long as the following:

2 pclk cycles of *synchronization_delay* + WDT_CR.RPL (programmed Reset Pulse Length)

This logic allows the wdt reset to be detected, even when pclk is disabled. The wdt_sys_rst remains asserted until pclk is restarted, and the expected reset pulse duration expires thereafter. Therefore, the time taken to make pclk available after reset would further add to the total reset pulse length.

Figure 3-6 shows the timing diagram of the generation of a system reset when WDT_ASYNC_CLK_MODE_ENABLE = 1.

Figure 3-6 System Reset Generation



*internal signal

Note The wdt_sys_rst system reset, generated when WDT_ASYNC_CLK_MODE_ENABLE = 1, is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

3.4 Pause Mode

The DW_apb_wdt can be configured to include a pause signal on the interface, which “freezes” the watchdog counter while the system is being paused. During pause mode, if the counter is frozen at the zero count, no interrupt or system reset is generated. When the pause is removed, the interrupt or system reset is asserted on the next rising edge of the clock. If wdt_clk_en is present, the interrupt or reset is generated only when the wdt_clk_en is asserted. If the counter is not zero when the pause is removed, the interrupt or system reset is not generated.

3.5 External Clock Enable

The DW_apb_wdt can be configured to include an external clock enable (wdt_clk_en) that controls the rate at which the counter decrements. When the counter reaches zero, the wdt_clk_en must be asserted for the interrupt or system reset to be generated.

If a restart occurs when the wdt_clk_en is low, the restart is internally extended until the next rising edge of the wdt_clk_en so that it may be seen and the counter can be restarted. Clearing of interrupts is independent to the clock enable, but both the interrupt and reset are generated only when the clock enable is high.

Note The external clock enabled cannot be included when WDT_ASYNC_CLK_MODE_ENABLE = 1.

3.6 Reset Pulse Length

The reset pulse length is the number of pclk cycles for which a system reset is asserted. When a system reset is generated, it remains asserted for the number of cycles specified by the reset pulse length plus two cycles of synchronization delay, or until the system is reset (by the Reset Controller, see Figure 3-1 on page 29). A counter restart has no effect on the system reset once it has been asserted.

3.7 Timeout Period Values

The DW_apb_wdt can be configured to have a fixed or user-defined timeout period range. In both cases, the values that may be selected are limited by the WDT counter width. If the timeout period range that is selected is greater than the counter width, the timeout period is truncated to fit to the counter width. In the case of user-defined timeout period ranges, the value is limited at the time of configuration, and values greater than the count are not accepted.

4 Parameters

This chapter describes the configuration parameters used by the DW_apb_wdt. The settings of the configuration parameters determine the I/O signal list of the DW_apb_wdt peripheral. You use coreConsultant or coreAssembler to configure the following parameters and generate the configured code.

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

4.1 Parameter Descriptions

In the following tables, the values 0 and 1 occasionally appear in parentheses in the descriptions for the parameters. These are the logical values for parameter settings that appear in the coreConsultant GUI as check boxes, drop-down lists, a multiple selection, and so on.

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

Table 4-1 Top-Level Parameters

Field Label	Parameter Definition
System Configuration	
APB Data bus width	**Parameter Name:** APB_DATA_WIDTH **Legal Values:** 8, 16, or 32-bit **Default Value:** 32-bit **Dependencies:** None **Description**: Width of the APB data bus to which the component is attached.
WDT Counter	
WDT counter width	**Parameter Name:** WDT_CNT_WIDTH **Legal Values:** 16 to 32 bits **Default Value:** 32 bits **Dependencies:** None. **Description**: The Watchdog Timer counter width.

Table 4-1 Top-Level Parameters (Continued)

Field Label	Parameter Definition
Enable WDT from reset?	**Parameter Name:** WDT_ALWAYS_EN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None **Description**: Configures the WDT to be enabled from reset. If this setting is 1, the WDT is always enabled and a write to the WDT_EN field (bit 0) of the Watchdog Timer Control Register (WDT_CR) to disable it has no effect.
Interrupt and System Restart	
Active High Interrupt Polarity?	**Parameter Name:** WDT_INT_POL **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** This parameter is not needed if the configuration parameter WDT_HC_RMOD = 1 and the default response mode WDT_DFLT_RMOD = 0 so that a system reset is generated when a timeout occurs. **Description**: This sets the polarity of the generated interrupt. When set to 1, wdt_intr is included on the interface. When set to 0, wdt_intr_n is included on the interface. When this option is dimmed, wdt_intr or wdt_intr_n is not generated.
Active high reset polarity?	**Parameter Name:** WDT_RST_POL **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** None **Description**: This option sets the polarity of the generated reset. When set to 1, wdt_sys_rst is included on the interface. When set to 0, wdt_sys_rst_n is included on the interface.
Hard code reset pulse length?	**Parameter Name:** WDT_HC_RPL **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: Configures the reset pulse length to be hard coded.

Table 4-1 Top-Level Parameters (Continued)

Field Label	Parameter Definition
Default reset pulse length	**Parameter Name:** WDT_DFLT_RPL **Legal Values:** 0 to 7 **Default Value:** 0 **Dependencies:** None. **Description**: The reset pulse length that is available directly after reset. If WDT_HC_RPL is 1, then the default reset pulse length is the only possible reset pulse length ($2^{(WDT\_DFLT\_RPL+1)}$ pclk cycles). The range of values available for the reset pulse length are as follows: 0 – 2 pclk cycles 1 – 4 pclk cycles 2 – 8 pclk cycles 3 – 16 pclk cycles 4 – 32 pclk cycles 5 – 64 pclk cycles 6 – 128 pclk cycles 7 – 256 pclk cycles **Note** When WDT_SYNC_CLK_MOPE_ENABLE = 1, the total reset pulse length also includes the reset synchronization delay and the time taken for pclk to be made available. For details, refer to “System Resets” on page 31.
Hard code output response mode?	**Parameter Name:** WDT_HC_RMOD **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: Configures the output response mode to be hard coded.
Output response mode default value	**Parameter Name:** WDT_DFLT_RMOD **Legal Values:** 0 or 1 **Default Value:** 0 **Dependencies:** If WDT_HC_RMOD is 1, then the default response mode is the only possible response mode. **Description**: Describes the output response mode that is available directly after reset. Indicates the output response the WDT gives if a zero count is reached; that is, a system reset if it equals 0, and an interrupt followed by a system reset (if not cleared) if it equals 1. The values are: 0 – System reset only 1 – Interrupt and system reset

Table 4-1 Top-Level Parameters (Continued)

Field Label	Parameter Definition
External Configuration	
Are the Counter clock and the Peripheral clock asynchronous?	**Parameter Name:** WDT_ASYNC_CLK_MODE_ENABLE **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: To operate the counter with a clock that is asynchronous from the peripheral clock (pclk), select this parameter. When this parameter is selected, the tclk and tresetn ports are added to the top-level port list.
Include WDT counter clock enable input on I/F?	**Parameter Name:** WDT_CLK_EN **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** This option is available only when WDT_ASYNC_CLK_MODE_ENABLE = 0' **Description**: Configures the peripheral to have an external counter clock enable signal (wdt_clk_en) on the interface that can be used to control the rate at which the counter decrements.
Include pause enable input on I/F?	**Parameter Name:** WDT_PAUSE **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None **Description**: Configures the peripheral to have a pause enable signal (pause) on the interface that can be used to freeze the watchdog counter during pause mode.

Table 4-2 Timeout Configuration Parameters

Field Label	Parameter Definition
Timeout Configuration	
Hard code timeout period range selection?	**Parameter Name:** WDT_HC_TOP **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None **Description**: When set to 1, the selected timeout period(s) is set to be hard coded.

Table 4-2 Timeout Configuration Parameters (Continued)

Field Label	Parameter Definition
Use predefined timeout period ranges?	**Parameter Name:** WDT_USE_FIX_TOP **Legal Values:** True (1) or False (0) **Default Value:** True (1) **Dependencies:** None **Description**: When this setting is 1, timeout period range is fixed. The range increments by the power of 2 from 2^{16} to $2^{WDT\_CNT\_WIDTH-1}$. When this setting is 0, the user must define the timeout period range (2^8 to $2^{WDT\_CNT\_WIDTH}$ – 1) using the WDT_USER_TOP_(i) parameter.
Main Timeout	
Main timeout period range to be selected as default?	**Parameter Name:** WDT_DFLT_TOP **Legal Values:** 0 to 15 **Default Value:** 0 **Dependencies:** None. **Description**: The timeout period that is available directly after reset. It controls the reset value of the register. If WDT_HC_TOP is 1, then the default timeout period is the only possible timeout period. Can choose one of 16 values.
User defined main timeout period for range 0 to 15	**Parameter Name:** WDT_USER_TOP_(i) **Legal Values:** $2^8 - 1$ to $2^{WDT\_CNT\_WIDTH}-1$ **Default Value:** $2^{(16+i)} - 1$ **Dependencies:** This option is relevant under two scenarios: ■ If WDT_USE_FIX_TOP = 0 and WDT_HC_TOP = 0 ■ If WDT_USE_FIX_TOP = 0 and WDT_HC_TOP = 1 and WDT_DFLT_TOP = *i* **Description**: Describes the user-defined timeout period that is selected when WDT_TORR bits[3:0] (TOP) are equal to i.
Initial Timeout	
Include a second timeout period for initialization?	**Parameter Name:** WDT_DUAL_TOP **Legal Values:** True (1) or False (0) **Default Value:** False (0) **Dependencies:** None. **Description**: When set to 1, includes a second timeout period that is used for initialization prior to the first kick.
Initial timeout period range to be selected as default?	**Parameter Name:** WDT_DFLT_TOP_INIT **Legal Values:** 0 to 15 **Default Value:** 0 **Dependencies:** This option is relevant only if WDT_DUAL_TOP is 1. **Description**: Describes the initial timeout period that is available directly after reset. It controls the reset value of the register. If WDT_HC_TOP is 1, then the default initial time period is the only possible period.

Table 4-2 Timeout Configuration Parameters (Continued)

Field Label	Parameter Definition
User defined initial timeout period for range 0 to 15	**Parameter Name:** WDT_USER_TOP_INIT_(i) **Legal Values:** $2^8 – 1$ to $2^{WDT\_CNT\_WIDTH} – 1$ **Default Value:** $2^{(16+i)} – 1$ **Dependencies:** This option is relevant only if WDT_DUAL_TOP = 1, WDT_USE_FIX_TOP = 0, and WDT_HC_TOP = 1. **Description**: Describes the user-defined timeout period that is selected when WDT_TORR bits[7:4] (TOP_INIT) are equal to i.

Table 4-3 includes parameters that are derived from the user-selected parameters in coreConsultant.

Table 4-3 Derived Configuration Parameters

Parameter	Parameter Definition
WDT_CNT_RST	WDT Counter Reset value. If you have a dual timeout period, it is the default initial timeout period. Otherwise, it is the default timeout period. **Range:** If the timeout period is fixed (WDT_USE_FIX_TOP = 1), then 2^{16} to $2^{WDT\_CNT\_WIDTH–1}$, in increments of 2^1. Otherwise (WDT_USE_FIX_TOP = 0), then 2^8 to $2^{WDT\_CNT\_WIDTH}$ –1
WDT_TORR_WIDTH	Width of Timeout Range Register. If the WDT has been configured to have dual timeout periods, then this value is the default timeout period range concatenated with the default initial timeout period range, such as {WDT_DFLT_TOP_INIT, WDT_DFLT_TOP}. Otherwise, it is the default timeout period range. **Range:** 4 or 8 bits
WDT_TOP_RST	WDT Counter Reset value. If the WDT has been configured to have dual timeout periods then this value is the default initial timeout period. Otherwise, it is the default timeout period. **Range:** If WDT_DUAL_TOP {WDT_DFLT_TOP_INIT, WDT_DFLT_TOP} else WDT_DFLT_TOP

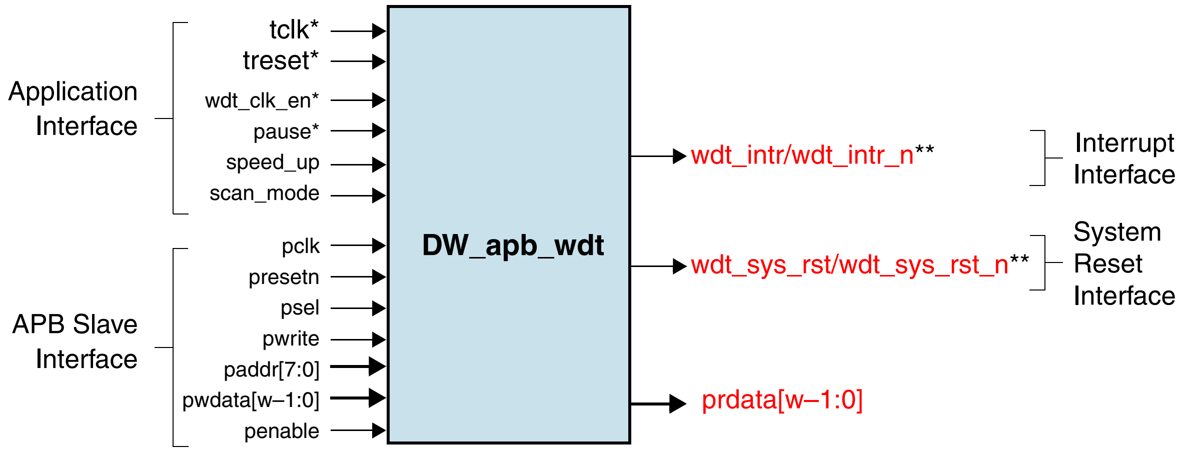
5 Signals

The following subsections describe the DW_apb_wdt I/O signals.

5.1 DW_apb_wdt Interface Diagram

Figure 5-1 shows the I/O signals for DW_apb_wdt (see Table 5-1 on page 42).

Figure 5-1 DW_apb_wdt Interface Diagram



*Optional signals. Refer to Table 5-1 for more information about the signal descriptions.
**Polarity is determined by configuration parameter setting. See Table 4-1 on page 35 for more information.
w = APB_DATA_WIDTH
Signals in red are registered.

Note The speed_up signal must be connected to zero when the DW_apb_wdt is instantiated.

5.2 DW_apb_wdt Signal Descriptions

Table 5-1 lists the signal description for the DW_apb_wdt component.

Note

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

In the **Registered** field, a "Yes" indicates whether an I/O signal is directly connected to an internal register and nothing else. An I/O signal is also considered to be registered if the signal is connected to one or more inverters or buffers between the I/O port and internal register, but not connected to any logic that involves another signal.

The **Input/Output Delay** field provides the percentage of the clock cycle assumed to be used by logic outside this design. The given value is used to automatically define the default synthesis constraints for input/output delay. You can override these default values in the Specify Port Constraints activity in coreConsultant or coreAssembler.

Table 5-1 I/O Signal Description

Name	Width	I/O	Function
APB Interface			
pclk	1 bit	In	APB clock – This clock times all bus transfers. All signal timings are related to the rising edge of pclk. **Active State:** N/A **Registered:** N/A **Synchronous to:** N/A **External Input Delay:** N/A
presetn	1 bit	In	APB Reset Signal – The bus reset signal is used to reset the system and the bus on the DesignWare interface. **Active State:** Low **Registered:** N/A **Synchronous to:** Asynchronous assertion, synchronous de-assertion. The reset must be de-asserted synchronously after the rising edge of pclk. DW_apb_wdt does not contain logic to perform this synchronization, so it must be provided externally. **External Input Delay:** N/A
psel	1 bit	In	APB peripheral select. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%
paddr	8 bits (7 to 0)	In	APB address bus; uses the lower bits of the APB address bus for register decode. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%

Table 5-1 I/O Signal Description (Continued)

Name	Width	I/O	Function
pwdata	See Description	In	APB write data bus. **Width:** *APB_DATA_WIDTH* **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%
pwrite	1 bit	In	APB write control. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%
penable	1 bit	In	APB enable control that indicates the second cycle of the APB frame. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%
prdata	See Description	Out	APB read data. **Width:** *APB_DATA_WIDTH* **Active State:** High **Registered:** Yes **Synchronous to:** pclk **External Output Delay:** 25%
Application Interface			
tclk	1 bit	In	*Optional*. Watchdog timer clock that can be asynchronous to pclk. **Active State:** N/A **Registered:** N/A **Synchronous to:** N/A **External Input Delay:** N/A **Dependencies:** Present only when configuration parameter WDT_ASYNC_CLK_MODE_ENABLE = 1.
treset	1 bit	In	*Optional*. Watchdog timer reset; used to reset the watchdog counter. **Active State:** Low **Registered:** N/A **Synchronous to:** Asynchronous assertion, synchronous de-assertion. The reset must be de-asserted synchronously after the rising edge of tclk. DW_apb_wdt does not contain logic to perform this synchronization, so it must be provided externally. **External Input Delay:** N/A **Dependencies:** Present only when configuration parameter WDT_ASYNC_CLK_MODE_ENABLE = 1.

Table 5-1 I/O Signal Description (Continued)

Name	Width	I/O	Function
wdt_clk_en	1 bit	In	*Optional*. Watchdog counter clock enable used to control the rate at which the counter decrements. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25% **Dependencies:** Present only when the configuration parameter WDT_CLK_EN = 1.
pause	1 bit	In	*Optional*. Pause enable. Used to freeze the watchdog counter during pause mode. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25% **Dependencies:** Present only when the configuration parameter WDT_PAUSE = 1.
speed_up	1 bit	In	Test signal. When asserted, the watchdog counter restart value is 255, regardless of the selected timeout period; used to speed up test times. Users must tie this signal to low when they instantiate DW_apb_wdt. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25% **Dependencies:** This signal must be connected to zero when the DW_apb_wdt is instantiated.
scan_mode	1 bit	In	Scan Mode used to obtain testability on speed_up logic during scan. This signal must be asserted during scan testing to ensure that all flip-flops in the design are controllable and observable during scan testing. At all other times, this signal must be deasserted. **Active State:** High **Registered:** No **Synchronous to:** pclk **External Input Delay:** 25%

Table 5-1 I/O Signal Description (Continued)

Name	Width	I/O	Function
Interrupt Interface			
wdt_intr	1 bit	Out	*Optional*. Active-high WDT interrupt. The following table shows when the wdt_intr signal exists. **Note** No interrupt pin exists if the DW_apb_wdt is configured to have a hardcoded timeout response (WDT_HC_RMOD = 1) and that hardcoded response does not include an interrupt (WDT_DFLT_RMOD = 0). (see table below) **Active State:** High **Registered:** Yes **Synchronous to:** pclk **Dependencies:** Present only when the configuration parameters (WDT_INT_POL==1) && !(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==0). **Note** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

WDT_INT_POL	WDT_HC_RMOD	WDT_DFLT_RMOD	wdt_intr
0	0	X	no
0	1	0	no
0	X	1	no
1	0	X	yes
1	1	0	no
1	X	1	yes

Table 5-1 I/O Signal Description (Continued)

Name	Width	I/O	Function
wdt_intr_n	1 bit	Out	*Optional*. Active-low interrupt. The following table shows when the wdt_intr_n signal exists. **Note** No interrupt pin exists if the DW_apb_wdt is configured to have a hardcoded timeout response (WDT_HC_RMOD = 1) and that hardcoded response does not include an interrupt (WDT_DFLT_RMOD = 0).

WDT_INT_ POL	WDT_HC_R MOD	WDT_DFLT _RMOD	wdt_intr_n
0	0	X	yes
0	1	0	no
0	X	1	yes
1	0	X	no
1	1	0	no
1	X	1	no

Name	Width	I/O	Function
			Active State: Low **Registered:** Yes **Synchronous to:** pclk **Dependencies:** Present only when (WDT_INT_POL==0) && !(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==0). **Note** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.
System Reset Interface			
wdt_sys_rst	1 bit	Out	*Optional*. Active-high system reset flag. **Active State:** High **Registered:** Yes **Synchronous to:** pclk **External Output Delay:** 25% **Dependencies:** Present only when the configuration parameter WDT_RST_POL = 1. **Note** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

Table 5-1 I/O Signal Description (Continued)

Name	Width	I/O	Function
wdt_sys_rst_n	1 bit	Out	*Optional*. Active-low system reset flag. **Active State:** Low **Registered:** Yes **Synchronous to:** pclk **Dependencies:** Present only when the configuration parameter WDT_RST_POL = 0. **Note** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

6 Registers

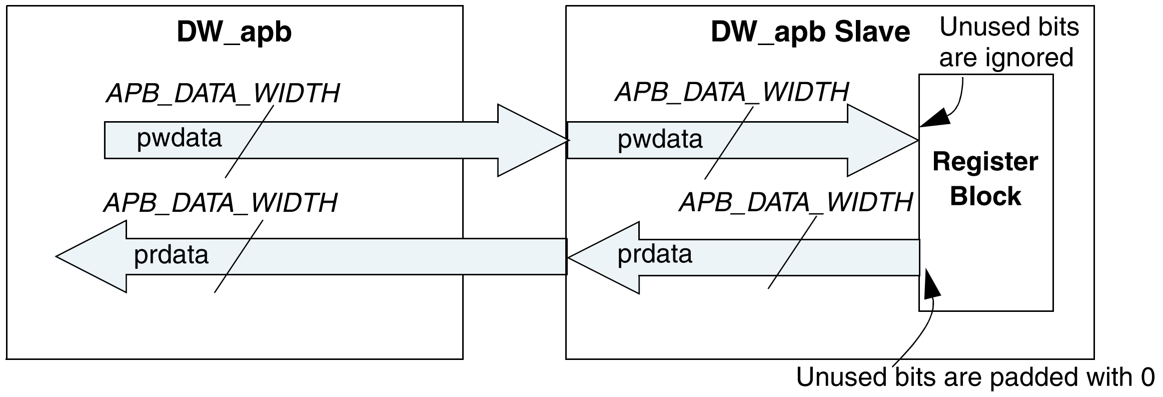
This section describes the programmable registers of the DW_apb_wdt.

6.1 Bus Interface

The DW_apb_wdt peripheral has a standard AMBA 2.0 APB interface for reading and writing the internal registers. This component supports APB data bus widths of 8, 16, or 32 bits, which is set with the APB_DATA_WIDTH parameter.

Figure 6-1 shows the read/write buses between the DW_apb and the APB slave.

Figure 6-1 Relationship Between DW_apb and Slave Data Widths



"Integration Considerations" on page 69 provides the APB interface information that is common to all the DW_apb peripherals.

6.2 Register Memory Map

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

Table 6-1 Memory Map of DW_apb_wdt

Name	Address Offset	Width	Description
WDT_CR	0x00	5 bits	Control register **Reset Value:** Register fields WDT_DFLT_RPL (3 bits), WDT_DFLT_RMOD (1 bit), and WDT_ALWAYS_EN (1 bit)
WDT_TORR	0x04	See Description	Timeout range register **Width:** WDT_TORR_WIDTH (an internal parameter that is either 4- or 8-bits wide, depending on the configuration parameter WDT_DUAL_TOP) **Reset Value:** WDT_TOP_RST
WDT_CCVR	0x08	See Description	Current counter value register **Width:** WDT_CNT_WIDTH **Reset Value:** WDT_CNT_RST
WDT_CRR	0x0c	8 bits	Counter restart register **Reset Value:** 0x0
WDT_STAT	0x10	1 bit	Interrupt status register **Reset Value:** 0x0
WDT_EOI	0x14	1 bit	Interrupt clear register **Reset Value:** 0x0
Reserved	0x18-0xe0		
WDT_COMP_PARAMS_5	0xe4	32 bits	Refer to table on page 57. **Reset Value:** Reset value depends on user configuration, refer to relevant tables for more information.
WDT_COMP_PARAMS_4	0xe8	32 bits	Refer to table on page 58. **Reset Value:** Reset value depends on user configuration, refer to relevant tables for more information.
WDT_COMP_PARAMS_3	0xec	32 bits	Refer to table on page 58. **Reset Value:** Reset value depends on user configuration, refer to relevant tables for more information.
WDT_COMP_PARAMS_2	0xf0	32 bits	Refer to table on page 59. **Reset Value:** 0x0

Table 6-1 Memory Map of DW_apb_wdt (Continued)

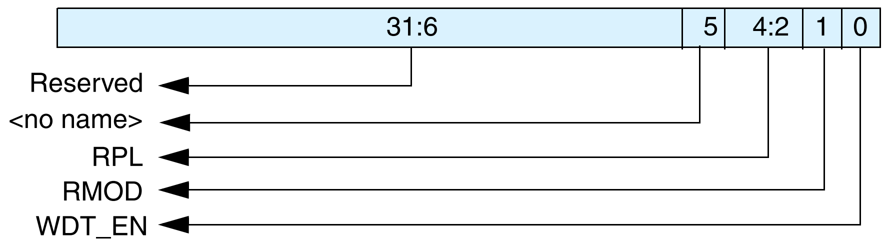
Name	Address Offset	Width	Description
WDT_COMP_PARAMS_1	0xf4	32 bits	Refer to table on page 60. **Reset Value:** 0x0
WDT_COMP_VERSION	0xf8	32 bits	DesignWare Component Version register **Reset Value:** ‘PRIORITY_1
WDT_COMP_TYPE	0xfc	32 bits	DesignWare Component Type register **Reset Value:** ‘PRIORITY_1

6.3 Register and Field Descriptions

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

6.3.1 WDT_CR

- **Name:** Control Register
- **Size:** 5 bits
- **Address Offset:** 0x00
- **Read/write access:** read/write

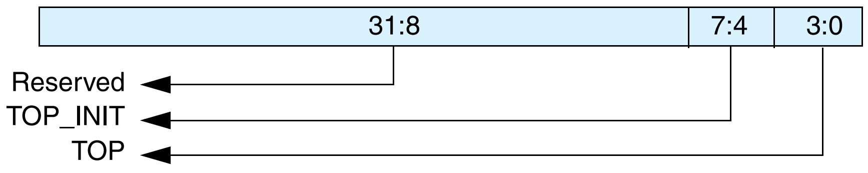


Bits	Name	R/W	Description
31:6	Reserved and read as zero (0).		
5	<no name>	R/W	Redundant R/W bit. Included for ping test purposes, as it is the only R/W register bit that is in every configuration of the DW_apb_wdt.
4:2	RPL	R/W	Reset pulse length. Writes have no effect when the configuration parameter WDT_HC_RPL is 1, making the register bits read-only. This is used to select the number of pclk cycles for which the system reset stays asserted. The range of values available is 2 to 256 pclk cycles. 000 – 2 pclk cycles 001 – 4 pclk cycles 010 – 8 pclk cycles 011 – 16 pclk cycles 100 – 32 pclk cycles 101 – 64 pclk cycles 110 – 128 pclk cycles 111 – 256 pclk cycles **Reset Value:** *WDT_DFLT_RPL* **Note** When WDT_SYNC_CLK_MOPE_ENABLE = 1, the total reset pulse length also includes the reset synchronization delay and the time taken for pclk to be made available. For details, refer to "System Resets" on page 31.

Bits	Name	R/W	Description
1	RMOD	R/W	Response mode. Writes have no effect when the parameter WDT_HC_RMOD = 1, thus this register becomes read-only. Selects the output response generated to a timeout. 0 = Generate a system reset. 1 = First generate an interrupt and if it is not cleared by the time a second timeout occurs then generate a system reset. **Reset Value:** WDT_DFLT_RMOD
0	WDT_EN	R/W	WDT enable. Writable when the configuration parameter WDT_ALWAYS_EN = 0, otherwise, it is readable. This bit is used to enable and disable the DW_apb_wdt. When disabled, the counter does not decrement. Thus, no interrupts or system resets are generated. Once this bit has been enabled, it can be cleared only by a system reset. 0 = WDT disabled. 1 = WDT enabled. **Reset Value:** *WDT_ALWAYS_EN*

6.3.2 WDT_TORR

- **Name:** Timeout Range Register
- **Size:** *WDT_TORR_WIDTH*
- **Address Offset:** 0x04
- **Read/write access:** read/write

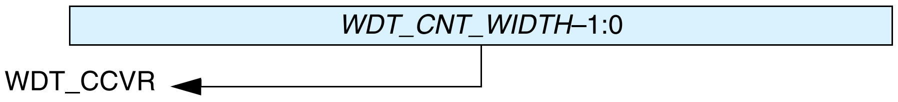


Bits	Name	R/W	Description
31:8	Reserved and read as zero (0).		
7:4	TOP_INIT	R/W	Timeout period for initialization. Writes to these register bits have no effect when the configuration parameter WDT_HC_TOP = 1 or WDT_ALWAYS_EN = 1. Used to select the timeout period that the watchdog counter restarts from for the first counter restart (kick). This register should be written after reset and before the WDT is enabled. A change of the TOP_INIT is seen only once the WDT has been enabled, and any change after the first kick is not seen as subsequent kicks use the period specified by the TOP bits. The range of values is limited by the WDT_CNT_WIDTH. If TOP_INIT is programmed to select a range that is greater than the counter width, the timeout period is truncated to fit to the counter width. This affects only the non-user specified values as users are limited to these boundaries during configuration. The range of values available for a 32-bit watchdog counter are: Where i = TOP_INIT and t = timeout period For i = 0 to 15 if WDT_USE_FIX_TOP==1 t = 2(16 + i) else t = WDT_USER_TOP_INIT_(i) **Reset Value:** Configuration parameter WDT_DFLT_TOP_INIT **NOTE:** These bits exist only when the configuration parameter WDT_DUAL_TOP = 1, otherwise, they are fixed at zero.

Bits	Name	R/W	Description
3:0	TOP	R/W	Timeout period. Writes have no effect when the configuration parameter WDT_HC_TOP = 1, thus making this register read-only. This field is used to select the timeout period from which the watchdog counter restarts. A change of the timeout period takes effect only after the next counter restart (kick). The range of values is limited by the WDT_CNT_WIDTH. If TOP is programmed to select a range that is greater than the counter width, the timeout period is truncated to fit to the counter width. This affects only the non-user specified values as users are limited to these boundaries during configuration. The range of values available for a 32-bit watchdog counter are: Where i = TOP and t = timeout period For i = 0 to 15 if WDT_USE_FIX_TOP==1 t = 2(16 + i) else t = WDT_USER_TOP_(i) **Reset Value:** *WDT_DFLT_TOP*

6.3.3 WDT_CCVR

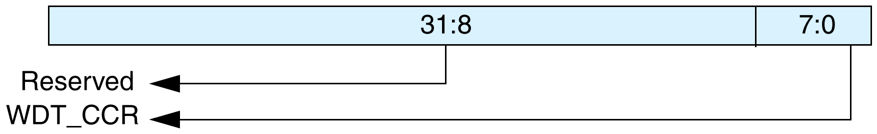
- **Name:** Current Counter Value Register
- **Size:** *WDT_CNT_WIDTH*
- **Address Offset:** 0x08
- **Read/write access:** read



Bits	Name	R/W	Description
WDT_CNT_WIDTH–1:0	Current Counter Value Register	R	This register, when read, is the current value of the internal counter. This value is read coherently when ever it is read, which is relevant when the APB_DATA_WIDTH is less than the counter width. **Reset Value:** *WDT_CNT_RST*

6.3.4 WDT_CRR

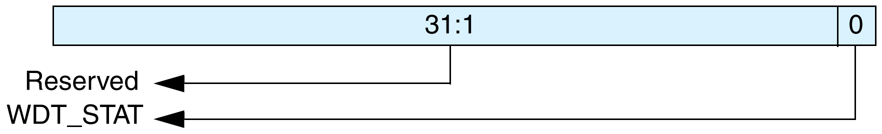
- **Name:** Counter Restart Register
- **Size:** 8 bits
- **Address Offset:** 0x0c
- **Read/write access:** write



Bits	Name	R/W	Description
31:8	Reserved and read as zero.		
7:0	Counter Restart Register	W	This register is used to restart the WDT counter. As a safety feature to prevent accidental restarts, the value 0x76 must be written. A restart also clears the WDT interrupt. Reading this register returns zero. **Reset Value:** 0

6.3.5 WDT_STAT

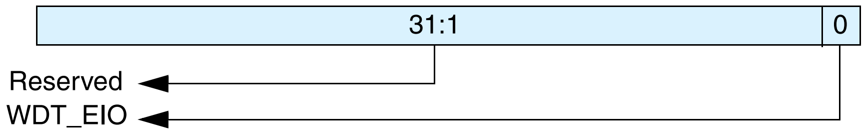
- **Name:** Interrupt Status Register
- **Size:** 1 bit
- **Address Offset:** 0x10
- **Read/write access:** read



Bits	Name	R/W	Description
31:1	Reserved and read as zero.		
0	Interrupt Status Register	R	This register shows the interrupt status of the WDT. 1 = Interrupt is active regardless of polarity. 0 = Interrupt is inactive. **Reset Value:** 0

6.3.6 WDT_EOI

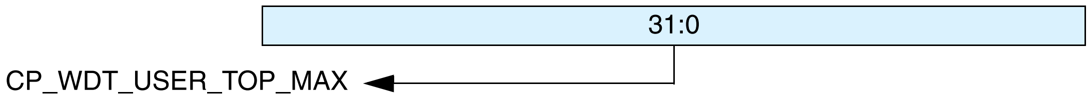
- **Name:** Interrupt Clear Register
- **Size:** 1 bit
- **Address Offset:** 0x14
- **Read/write access:** read



Bits	Name	R/W	Description
31:1	Reserved and read as zero.		
0	Interrupt Clear Register	R	Clears the watchdog interrupt. This can be used to clear the interrupt without restarting the watchdog counter. **Reset Value:** 0

6.3.7 WDT_COMP_PARAMS_5

- **Name:** Component Parameters Register 5
- **Size:** 32 bits
- **Address Offset:** 0xe4
- **Read/write access:** read

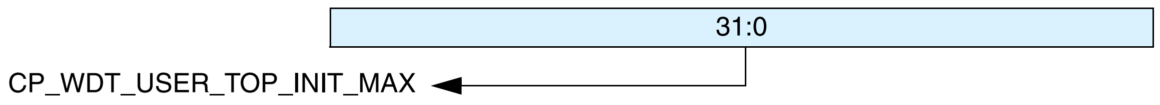


Note This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).

Bits	Name	R/W	Description
31:0	CP_WDT_USER_TOP_MAX	R	Upper limit of Timeout Period parameters. The value of this register is derived from the WDT_USER_TOP_* coreConsultant parameters. See Table 4-2 on page 38 for a description of these parameters.

6.3.8 WDT_COMP_PARAMS_4

- **Name:** Component Parameters Register 4
- **Size:** 32 bits
- **Address Offset:** 0xe8
- **Read/write access:** read

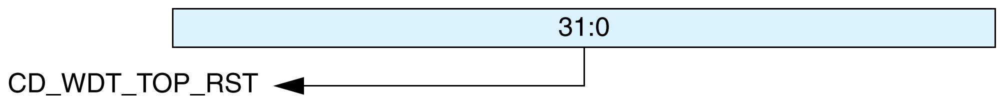


Note This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).

Bits	Name	R/W	Description
31:0	CP_WDT_USER_TOP_INIT_MAX	R	Upper limit of Initial Timeout Period parameters. The value of this register is derived from the WDT_USER_TOP_INIT_* coreConsultant parameters. See Table 4-2 on page 38 for a description of these parameters.

6.3.9 WDT_COMP_PARAMS_3

- **Name:** Component Parameters Register 3
- **Size:** 32 bits
- **Address Offset:** 0xec
- **Read/write access:** read

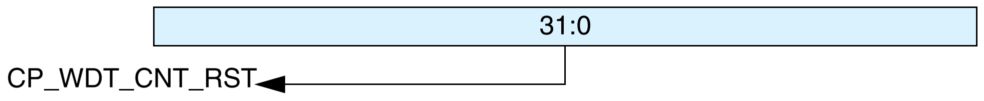


Note This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).

Bits	Name	R/W	Description
31:0	CD_WDT_TOP_RST	R	The value of this register is derived from the WDT_TOP_RST coreConsultant parameter. See Table 4-2 on page 38 for a description of this parameter.

6.3.10 WDT_COMP_PARAMS_2

- **Name:** Component Parameters Register 2
- **Size:** 32 bits
- **Address Offset:** 0xf0
- **Read/write access:** read

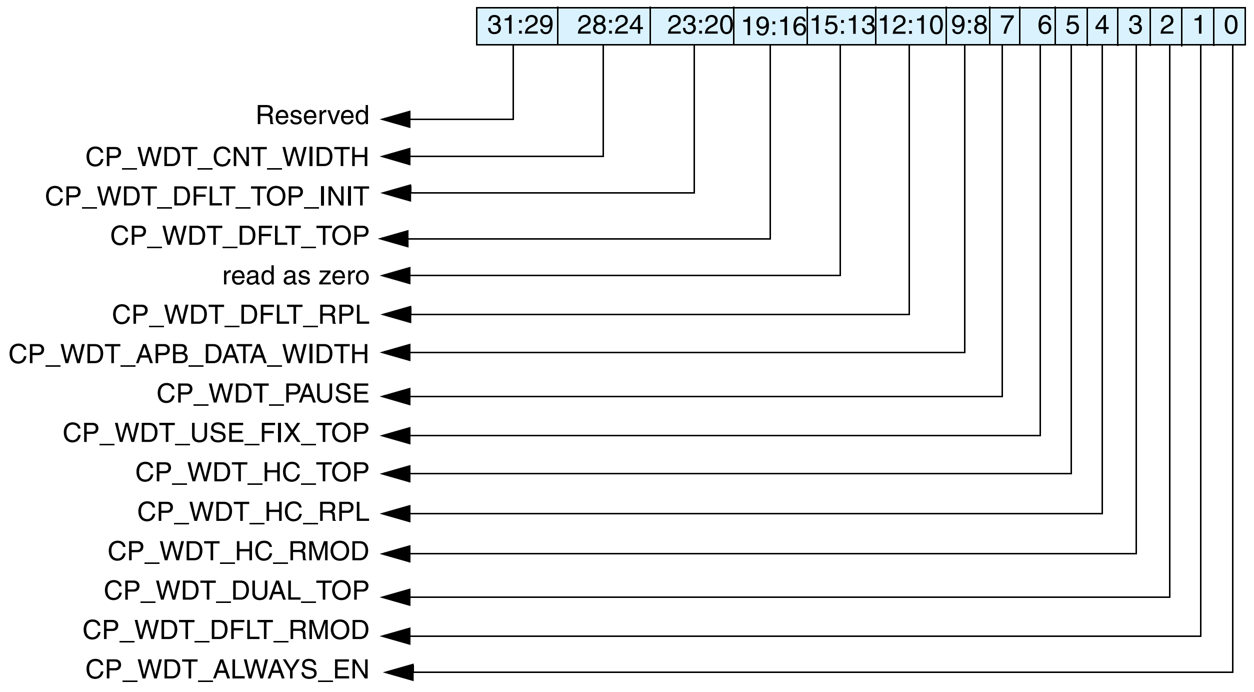


Note This is a constant read-only register that contains encoded information about the component's parameter settings. The reset value depends on coreConsultant parameter(s).

Bits	Name	R/W	Description
31:0	CP_WDT_CNT_RST	R	The value of this register is derived from the WDT_RST_CNT coreConsultant parameter. See Table 4-2 on page 38 for a description of this parameter.

6.3.11 WDT_COMP_PARAMS_1

- **Name:** Component Parameters Register 1
- **Size:** 32 bits
- **Address Offset:** 0xf4
- **Read/write access:** read



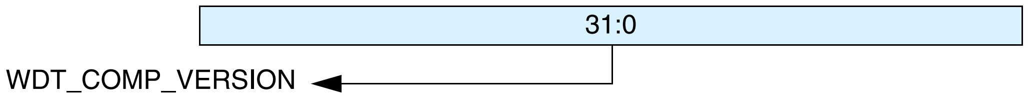
Note This is a constant read-only register that contains encoded information about the component's parameter settings. Some of the reset values depend on coreConsultant parameter(s).

Bits	Name	R/W	Description
31:29	Reserved and read as zero		
28:24	CP_WDT_CNT_WIDTH	R	WDT_CNT_WIDTH - 16
23:20	CP_WDT_DFLT_TOP_INIT	R	WDT_DFLT_TOP_INIT
19:16	CP_WDT_DFLT_TOP	R	WDT_DFLT_TOP
15:13	Reserved and read as zero		
12:10	CP_WDT_DFLT_RPL	R	WDT_DFLT_RPL

Bits	Name	R/W	Description
9:8	CP_WDT_APB_DATA_WIDTH	R	(APB_DATA_WIDTH == 8) = 0 (APB_DATA_WIDTH == 16) = 1 (APB_DATA_WIDTH == 32) = 2 reserved = 3
7	CP_WDT_PAUSE	R	**Reset Value:** 0
6	CP_WDT_USE_FIX_TOP	R	(WDT_USE_FIX_TOP == FALSE) =0 (WDT_USE_FIX_TOP == TRUE) =1
5	CP_WDT_HC_TOP	R	(WDT_HC_TOP == FALSE) =0 (WDT_HC_TOP == TRUE) =1
4	CP_WDT_HC_RPL	R	(WDT_HC_RPL == FALSE) =0 (WDT_HC_RPL == TRUE) =1
3	CP_WDT_HC_RMOD	R	(WDT_HC_RMOD == FALSE) =0 (WDT_HC_RMOD == TRUE) =1
2	CP_WDT_DUAL_TOP	R	(WDT_DUAL_TOP == FALSE) =0 (WDT_DUAL_TOP == TRUE) =1
1	CP_WDT_DFLT_RMOD	R	(WDT_DFLT_RMOD == FALSE) =0 (WDT_DFLT_RMOD == TRUE) =1
0	CP_WDT_ALWAYS_EN	R	(WDT_ALWAYS_EN == FALSE) =0 (WDT_ALWAYS_EN == TRUE) =1

6.3.12 WDT_COMP_VERSION

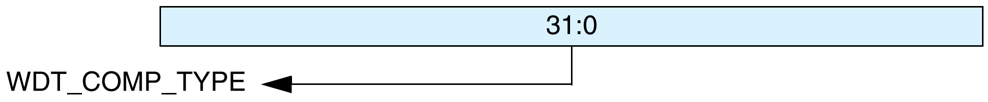
- **Name:** Component Version Register
- **Size:** 32 bits
- **Address Offset:** 0xf8
- **Read/write access:** read



Bits	Name	R/W	Description
31:0	WDT Component 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 AMBA 2 release notes

6.3.13 WDT_COMP_TYPE

- **Name:** Component Type Register
- **Size:** 32 bits
- **Address Offset:** 0xfc
- **Read/write access:** read



Bits	Name	R/W	Description
31:0	Component Type Register	R	Designware Component Type number = 0x44_57_01_20. This assigned unique hex value is constant, and is derived from the two ASCII letters “DW” followed by a 16-bit unsigned number.

7 Programming the DW_apb_wdt

This chapter describes the programmable features of the DW_apb_wdt.

7.1 Programming Considerations

As an APB slave, the DW_apb_wdt component is little-endian. As the largest register width can be 32-bits, all registers are aligned to 32-bit boundaries. Aligning to 32-bit boundaries keeps the same memory map for all bus widths. The APB bus reset, (presetn), resets all registers. The base address of DW_apb_wdt is not fixed and is determined by the DW_apb in the generation of the psel signal for DW_apb_wdt. Offset addresses from the base address are used for each register.

When a register to be read is narrower than the data bus width, there is no need for coherency logic. There are coherency registers when the data bus width is less than the counter width. It is possible for the WDT_CCVR registers to be larger than the data bus width, therefore coherency logic may be required when reading.

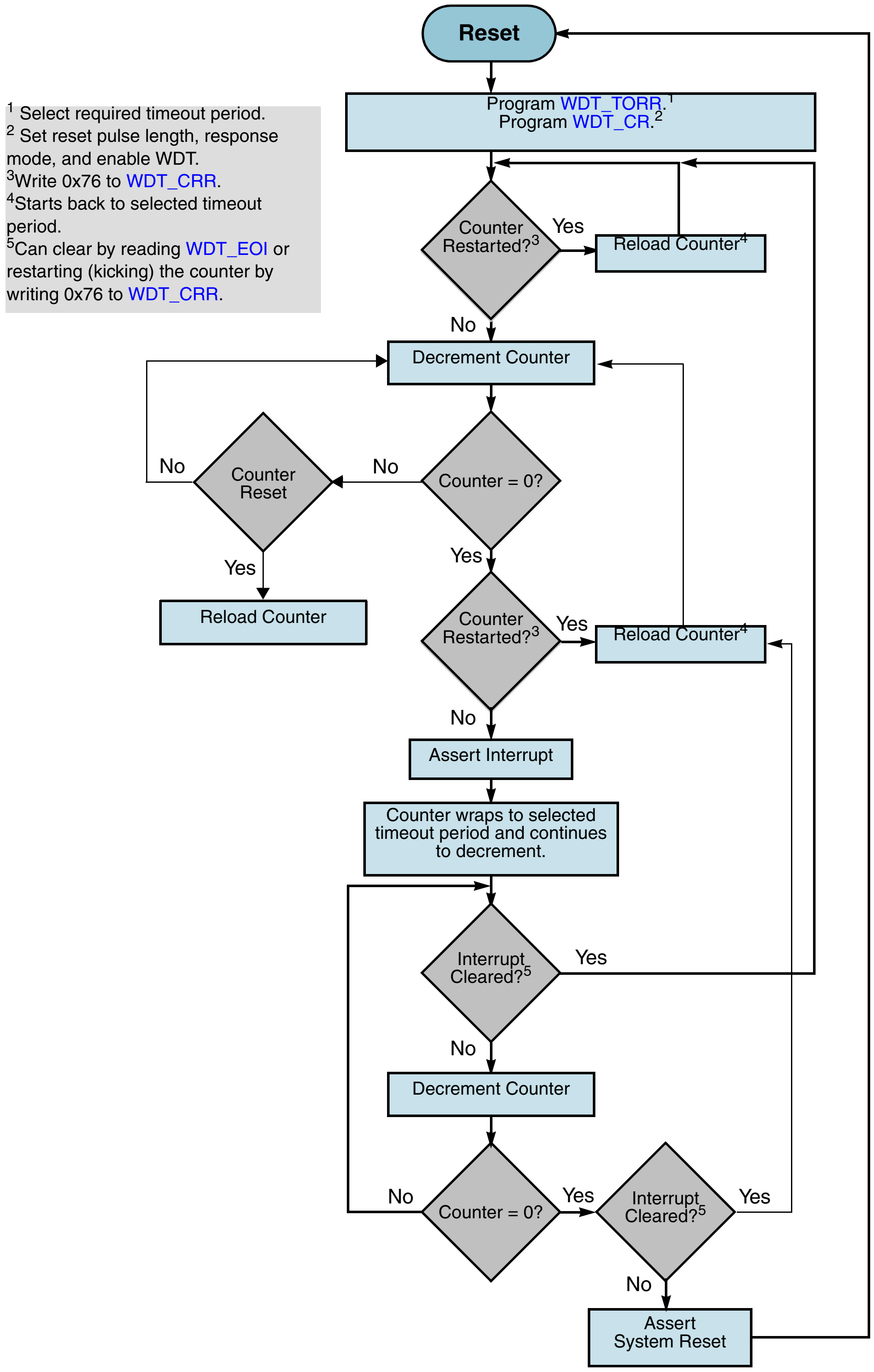
7.2 Example Operational Flow of DW_apb_wdt

Figure 7-1 on page 64 illustrates the operational flow of a DW_apb_wdt component configured with the following configuration parameters:

- Single timeout period
- Response mode not hard coded
- Reset pulse length not hard coded
- WDT not always enabled
- Generates interrupt, and then system reset

For more information about configuration parameters, refer to "Top-Level Parameters" on page 35.

Figure 7-1 Operation Flow of an Example DW_apb_wdt Configuration


Reset
Program WDT_TORR.[1]
Program WDT_CR.[2]
[1] Select required timeout period.
[2] Set reset pulse length, response mode, and enable WDT.
[3]Write 0x76 to WDT_CRR.
[4]Starts back to selected timeout period.
[5]Can clear by reading WDT_EOI or restarting (kicking) the counter by writing 0x76 to WDT_CRR.
Counter Restarted?[3]
Yes
Reload Counter[4]
No
Decrement Counter
No
Counter Reset
No
Counter = 0?
Yes
Reload Counter
Yes
Counter Restarted?[3]
Yes
Reload Counter[4]
No
Assert Interrupt
Counter wraps to selected timeout period and continues to decrement.
Interrupt Cleared?[5]
Yes
No
Decrement Counter
No
Counter = 0?
Yes
Interrupt Cleared?[5]
Yes
No
Assert System Reset

8 Verification

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

Note The DW_apb_wdt verification testbench is built with DesignWare Verification IP (VIP). Please make sure you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, refer to the following web page:
www.synopsys.com/products/designware/docs/doc/amba/latest/dw_amba_install.pdf

8.1 Overview of Vera Tests

The DW_apb_wdt verification testbench performs the following set of tests, which have been written to exhaustively verify the functionality of the component and have also achieved maximum RTL code coverage.

8.1.1 Test_apbif

The DW_apb_wdt consists of a register block, which implements the memory map for the peripheral. Regardless of the contents of any/all of the registers, a reset returns them to their default state. A reset also clears the interrupt and/or the system reset if it is asserted.

This test verifies that all the WDT read/write registers can be written and read. It also verifies that the read-only registers are updated correctly and that these values can be read back. Various register widths are dependent on the configuration settings. Setting upper bits has no effect on operation; they are ignored. Upper bits are read back as zero.

This test also verifies read coherency in the WDT_CCVR register. Additionally, this test verifies compliance with the *AMBA Specification*, version 2.0 from ARM.

8.1.2 Test_intr_n_sys_rst

The DW_apb_wdt consists of an interrupt and system reset control block, which is responsible for the generation of interrupts and system resets. This test verifies that the interrupts and/or system resets are generated correctly and at the correct polarity (active high or active low). It also verifies that the interrupt (if selected) can be cleared by either reading the WDT_EOI register or restarting the WD counter.

8.1.3 Test_counter

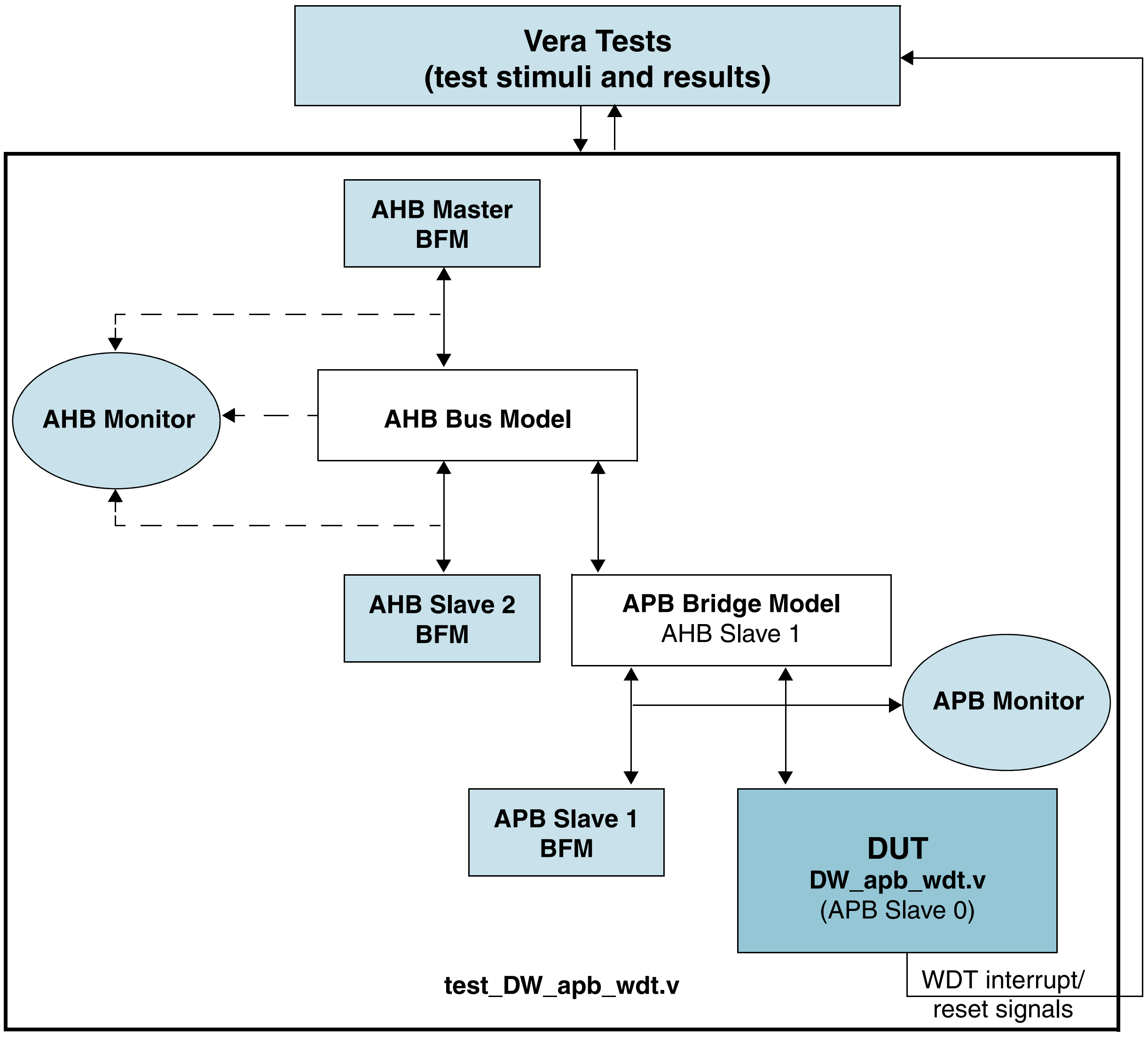
The WDT counter operation is essential to the overall operation of the design. Unless the WDT is enabled from reset, the counter does not start until the peripheral is enabled. If the counter clock enable is present, this test checks that it operates as expected. When the counter reaches zero, it wraps to the restart value and continue decrementing. When the WDT is configured to have a dual timeout period, the counter resets to the first timeout period and all subsequent restarts use the second timeout period. These tests also verify the operation of the component during pause mode, making sure that an interrupt or system reset is not generated during pause mode if the counter is frozen at the zero count. When the count is zero, the test checks to see if the interrupt or system reset becomes asserted on the next rising edge of the clock. If wdt_clk_en is present, the interrupt or reset is generated only when the wdt_clk_en is asserted.

8.2 Overview of DW_apb_wdt Testbench

As illustrated in Figure 8-1 on page 67, the Verilog DW_apb_wdt testbench includes an instantiation of the design under test (DUT), AHB and APB Bridge bus models, and a Vera shell. The Vera shell consists of an AHB master bus functional model (BFM), two AHB Slave BFMs, an AHB monitor, APB slave BFMs, an APB monitor, test stimuli, BFM configuration, and test results. The AHB monitor monitors activity from the AHB master and slave BFMs; the APB monitor oversees activity from the APB Slave BFMs.

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 8-1 DW_apb_wdt Testbench



9 Integration Considerations

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

9.1 Reading and Writing from an APB Slave

When writing to and reading from DesignWare APB slaves, you should consider the following:

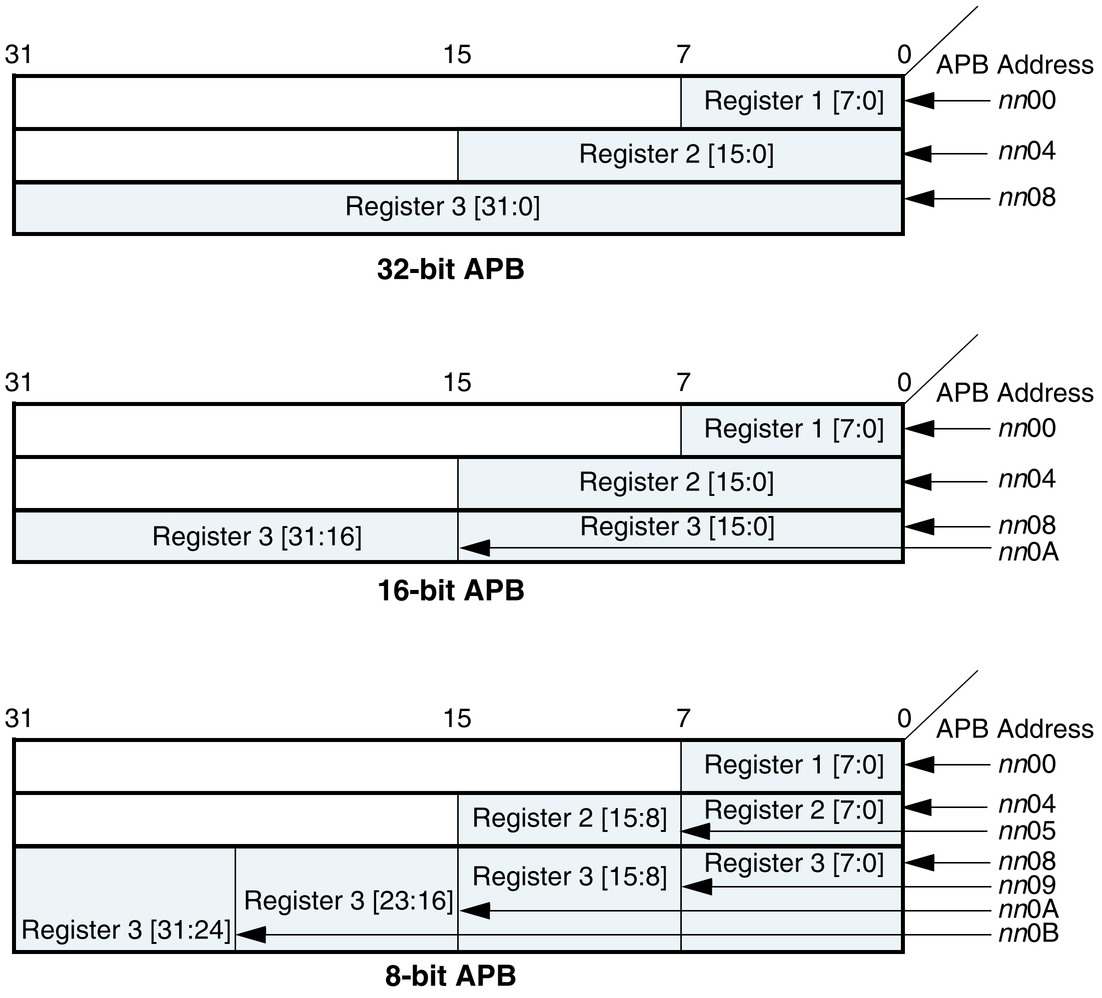
- The size of the APB peripheral should always be set equal to the size of the APB data bus, if possible.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this occupies more than one location in the memory map.
- The DW_apb does not return any ERROR, SPLIT, or RETRY responses; it always returns an OKAY response to the AHB.
- For all bus widths:
  - In the case of a read transaction, registers less than the full bus width returns zeros in the unused upper bits.
  - Writing to bit locations larger than the register width does not have any effect. Only the pertinent bits are written to the register.
- The APB slaves do not need the full 32-bit address bus, paddr. The slaves include the lower bits even though they are not actually used in a 32- or 16-bit system.

9.1.1 Reading From Unused Locations

Reading from an unused location or unused bits in a particular register always returns zeros. Unlike an AHB slave interface, which would return an error, there is no error mechanism in an APB slave and, therefore, in the DW_apb.

The following sections show the relationship between the register map and the read/write operations for the three possible APB_DATA_WIDTH values: 8-, 16-, and 32-bit APB buses.

Figure 9-1 Read/Write Locations for Different APB Bus Data Widths



9.1.2 32-bit Bus System

For 32-bit bus systems, all programming registers can be read or written with one operation, as illustrated in the previous figure.

Because all registers are on 32-bit boundaries, paddr[1:0] is not actually needed in the 32-bit bus case. But these bits still exist in the configured code for usability purposes.

Note If you write to an address location not on a 32-bit boundary, the bottom bits are ignored/not used.

9.1.3 16-bit Bus System

For 16-bit bus systems, two scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 16 bits

   In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 16 bits wide returns zeros in the un-used bits. Writing to bit locations larger than the register width causes nothing to happen, i.e. only the pertinent bits are written to the register.

2. The register to be written to or read from is >16 and <= 32 bits

   In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower two bytes (half-word) and the second transaction the upper half-word.

Because the bus is reading a half-word at a time, paddr[0] is not actually needed in the 16-bit bus case. But these bits still exist in the configured code for connectivity purposes.

Note If you write to an address location not on a 16-bit boundary, the bottom bits are ignored/not used.

9.1.4 8-bit Bus System

For 8-bit bus systems, three scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 8 bits

   In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 8 bits wide returns zeros in the unused bits. Writing to bit locations larger than the register width causes nothing to happen, that is, only the pertinent bits are written to the register.

2. The register to be written to or read from is >8 and <=16 bits

   In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the upper byte.

3. The register to be written to or read from is >16 and <=32 bits

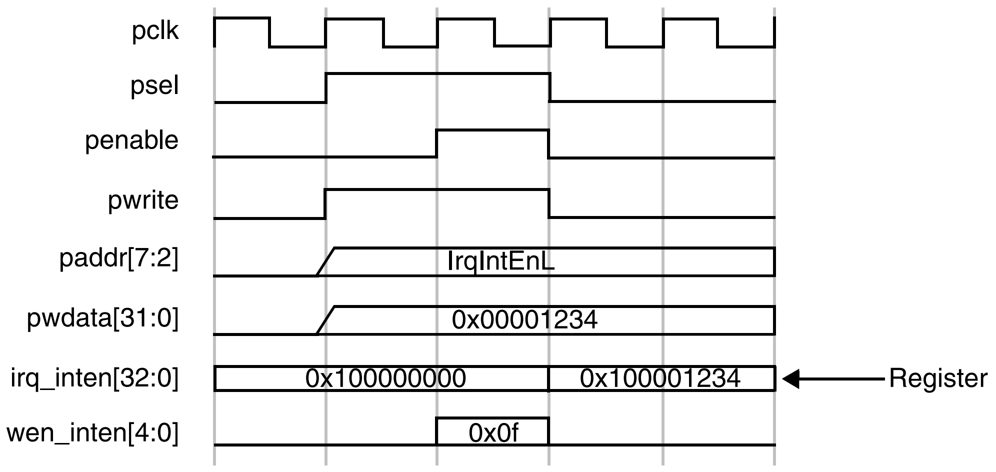
   In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

Because the bus is reading a byte at a time, all lower bits of paddr are decoded in the 8-bit bus case.

9.2 Write Timing Operation

A timing diagram of an APB write transaction for an APB peripheral register (an earlier version of the DW_apb_ictl) is shown in the following figure. Data, address, and control signals are aligned. The APB frame lasts for two cycles when psel is high.

Figure 9-2 APB Write Transaction



A write can occur after the first phase with penable low, or after the second phase when penable is high. The second phase is preferred and is used in all APB slave components. The timing diagram is shown with the write occurring after the second phase. Whenever the address on paddr matches a corresponding address from the memory map and provided psel, pwrite, and penable are high, then the corresponding register write enable is generated.

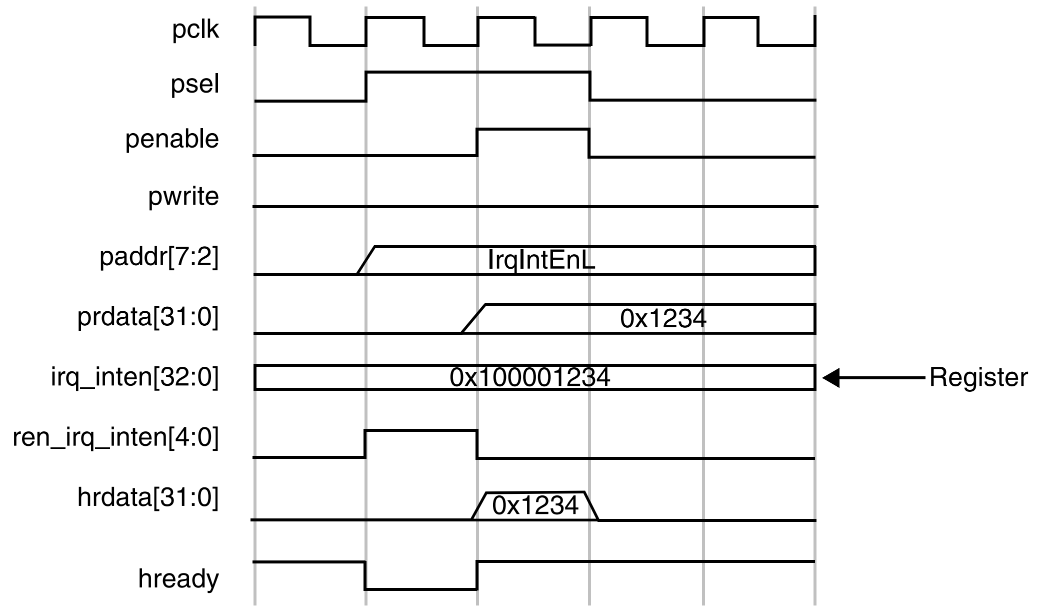
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

The timing example is a 33-bit register and a 32-bit APB data bus. To write this, 5 byte enables would be generated internally. The example shows writing to the first 32 bits with one write transaction.

9.3 Read Timing Operation

A timing diagram of an APB read transaction for an APB peripheral (an earlier version of the DW_apb_ictl) is shown in the following figure. The APB frame lasts for two cycles, when psel is high.

Figure 9-3 APB Read Transaction



Whenever the address on paddr matches the corresponding address from the memory map—psel is high, pwrite and penable are low—then the corresponding read enable is generated. The read data is registered within the peripheral before passing back to the master through the DW_apb and DW_ahb.

The qualification of the read-back data with hready from the bridge is shown in the timing diagram, but this does not form part of the APB interface. The read happens in the first APB cycle and is passed straight back to the AHB master in the same cycles as it passes through the bridge. By returning the data immediately to the AHB bus, the bridge can release control of the AHB data bus faster. This is important for systems where the APB clock is slower than the AHB clock.

Once a read transaction is started, it is completed and the AHB bus is held until the data is returned from the slave

Note If a read enable is not active, then the previously read data is maintained on the read-back data bus.

9.4 Accessing Top-level Constraints

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

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

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

   ```
   autocomplete_activity Synthesize
   ```

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

   ```
   write_sdc <filename>
   ```

9.5

9.6 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.

9.6.1 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 9-1 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

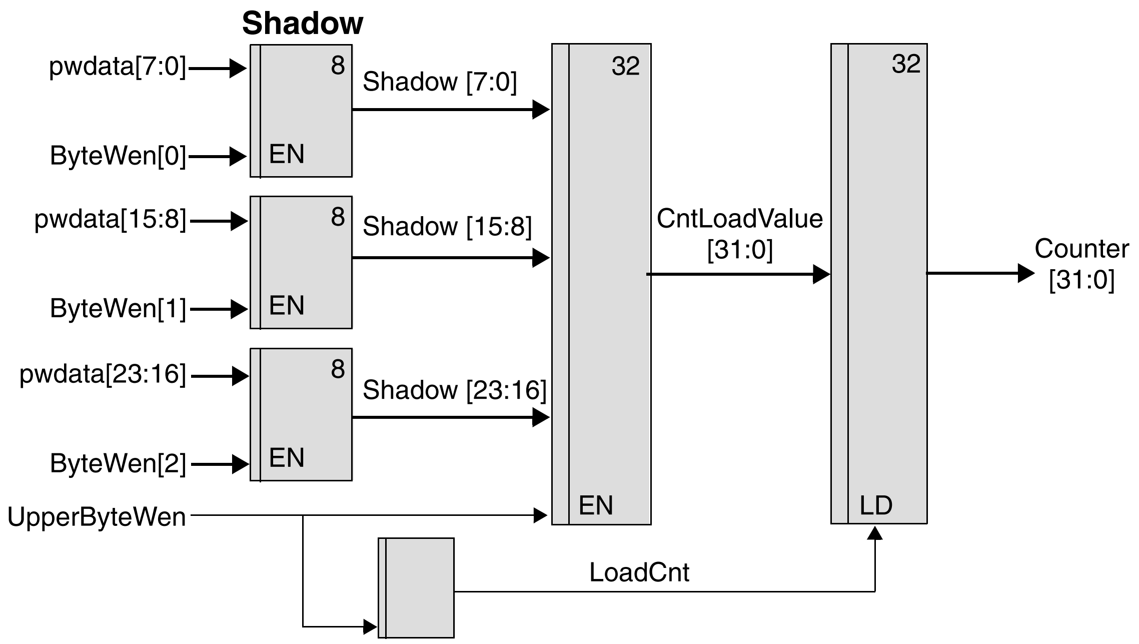
There are three relationship cases to be considered for the processor and peripheral clocks:

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

9.6.1.1 Identical Clocks

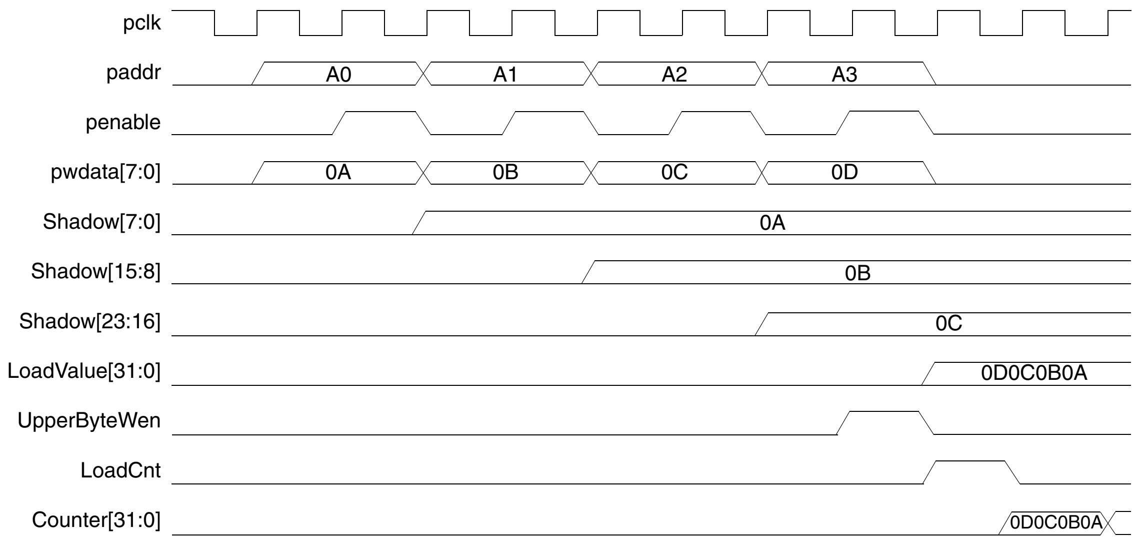
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 9-4 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 9-5 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.

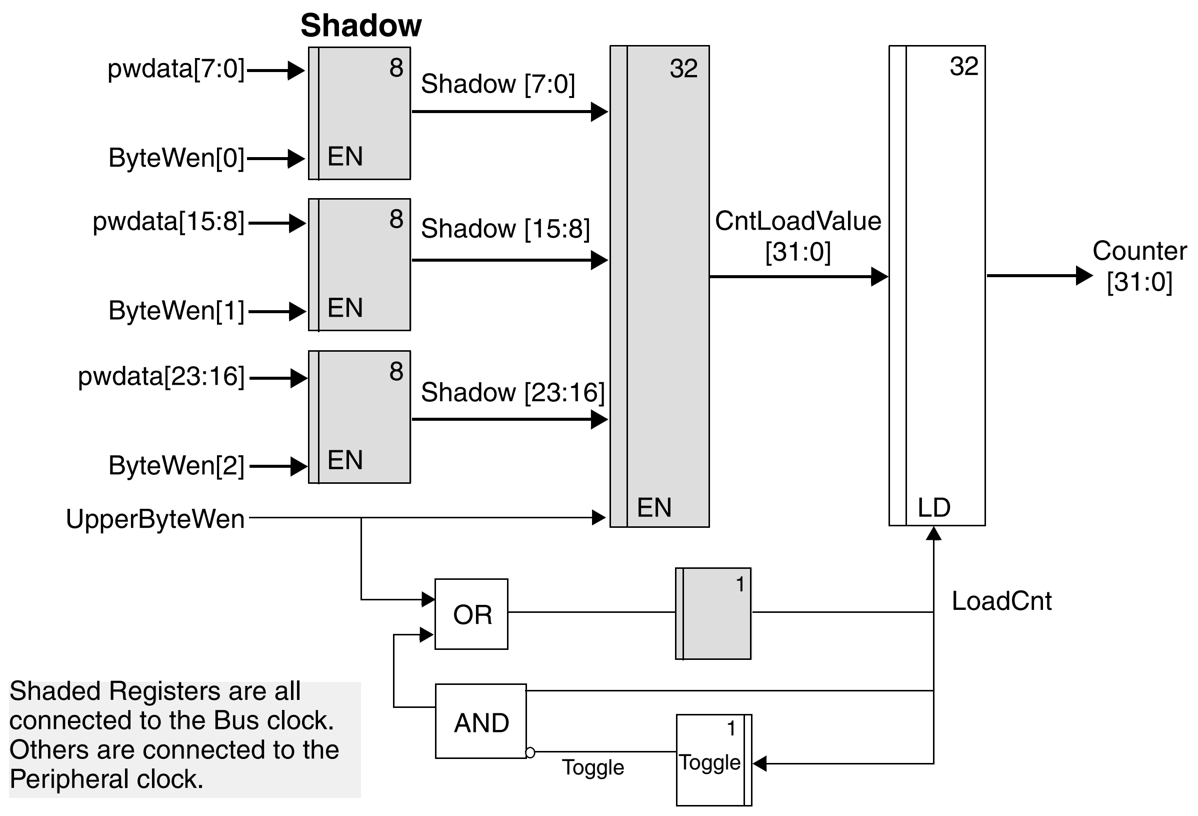
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this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

9.6.1.2 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.

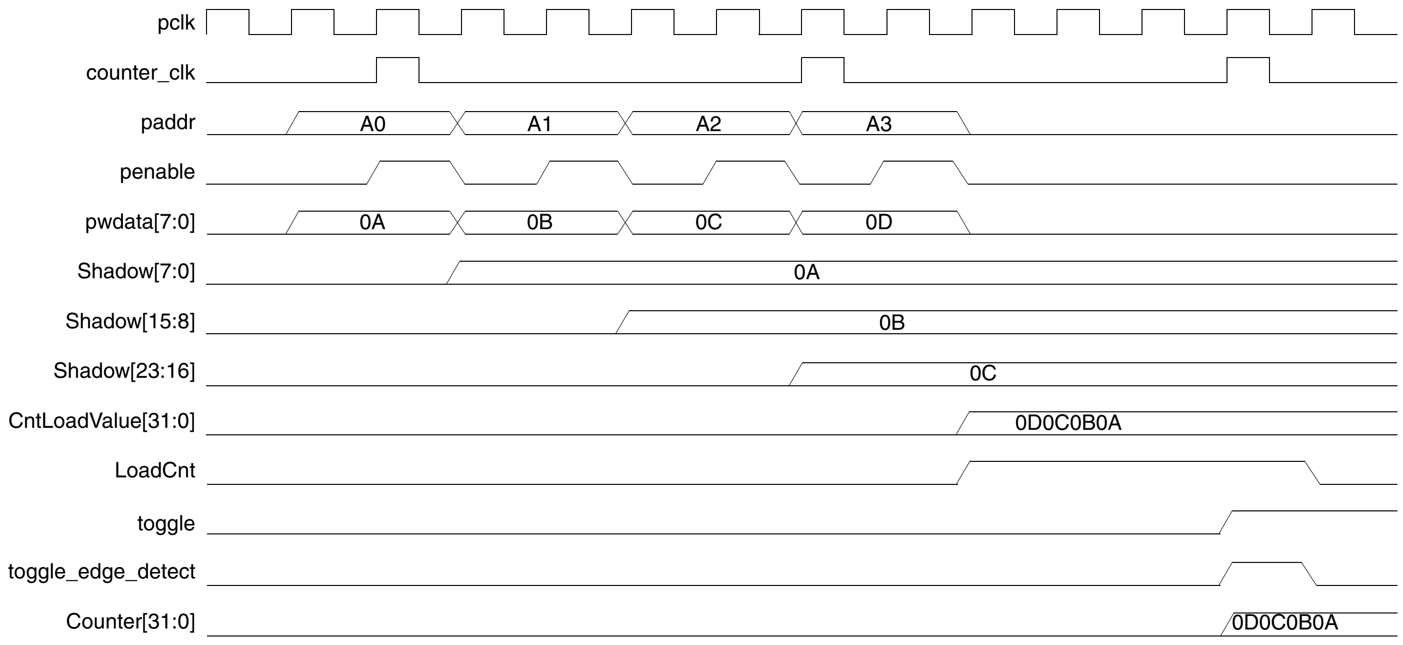
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 9-6 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.

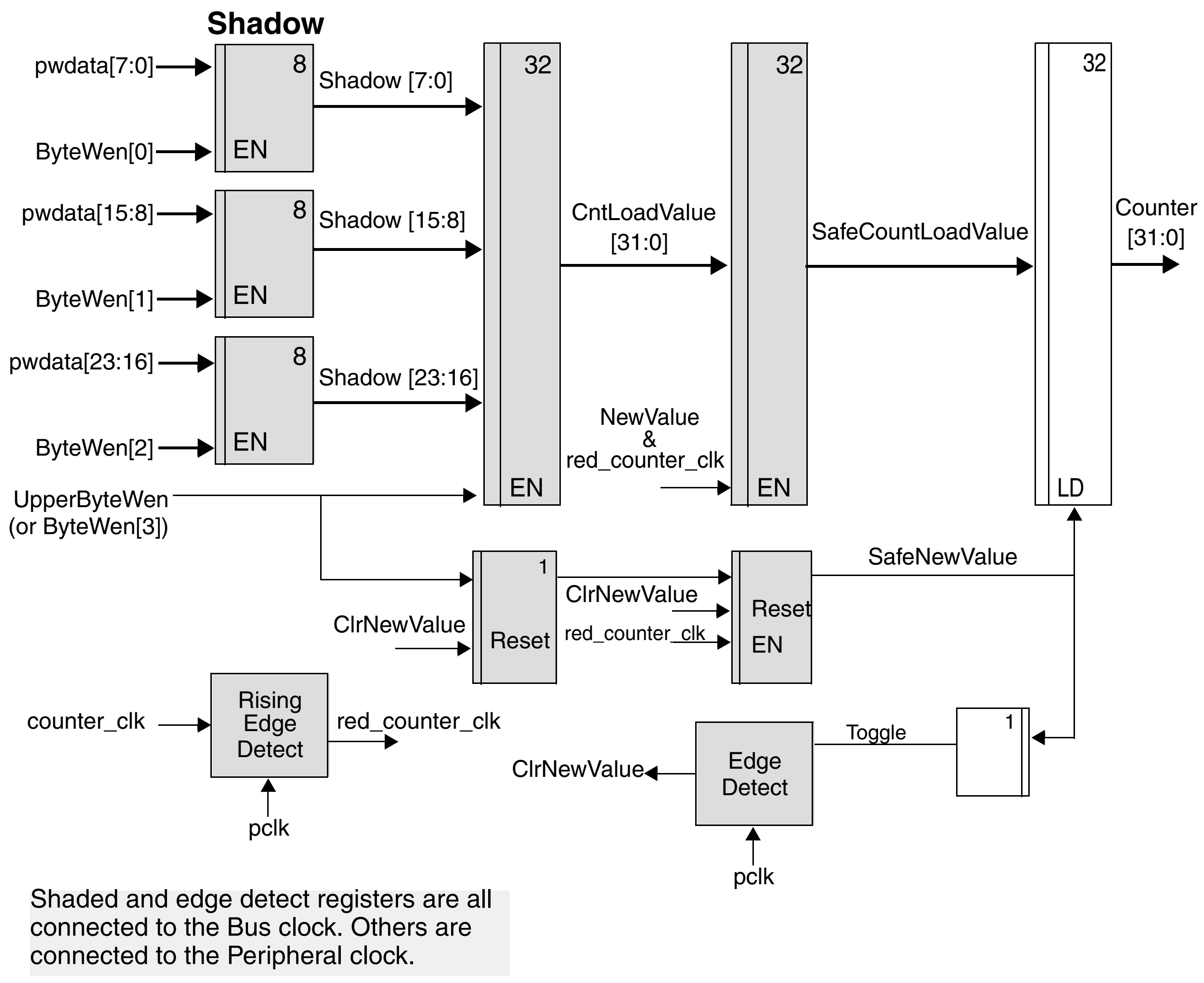
Figure 9-7 Coherent Loading – Synchronous Clocks



9.6.1.3 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 9-8 Coherent Loading – Asynchronous Clocks

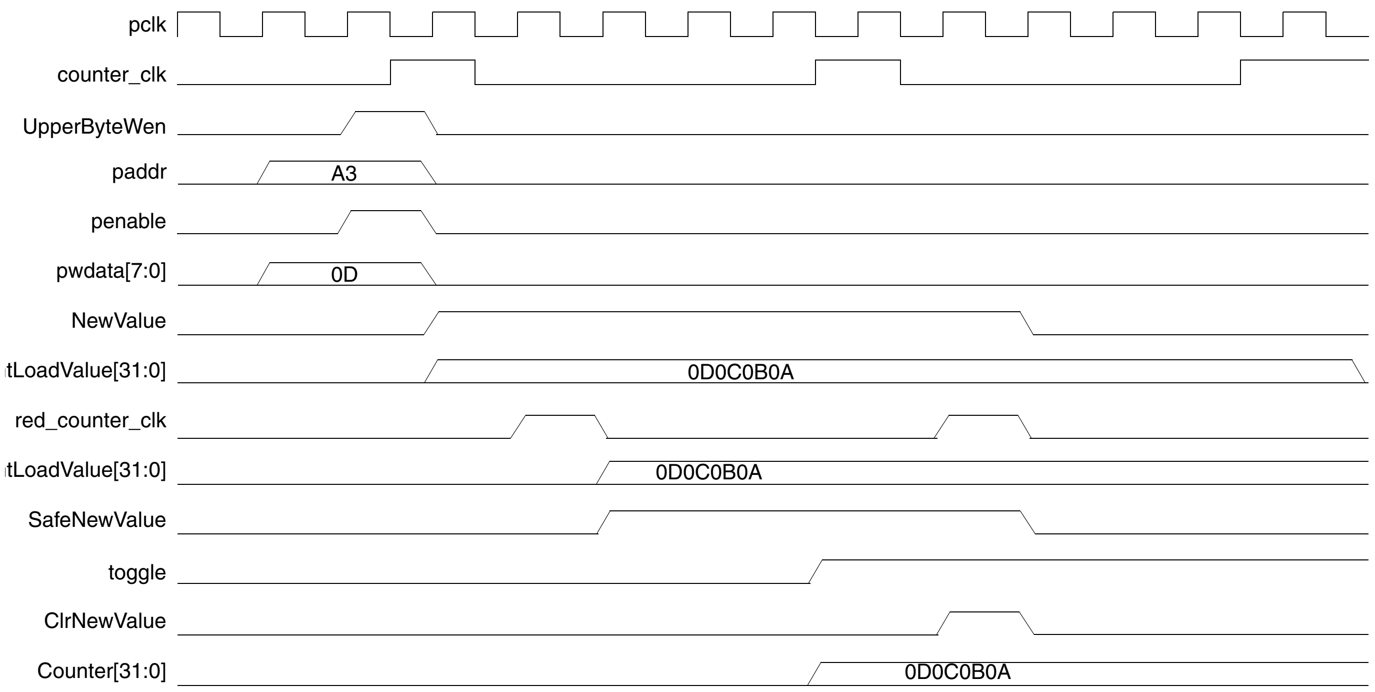


When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock 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 9-9 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.

9.6.2 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 9-2 Lower Byte Generation

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

Table 9-2 Lower Byte Generation

	Lower Byte Bus Width		
17 - 24	0	0	NCR
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

9.6.2.1 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 9-10 Coherent Registering – Synchronous Clocks

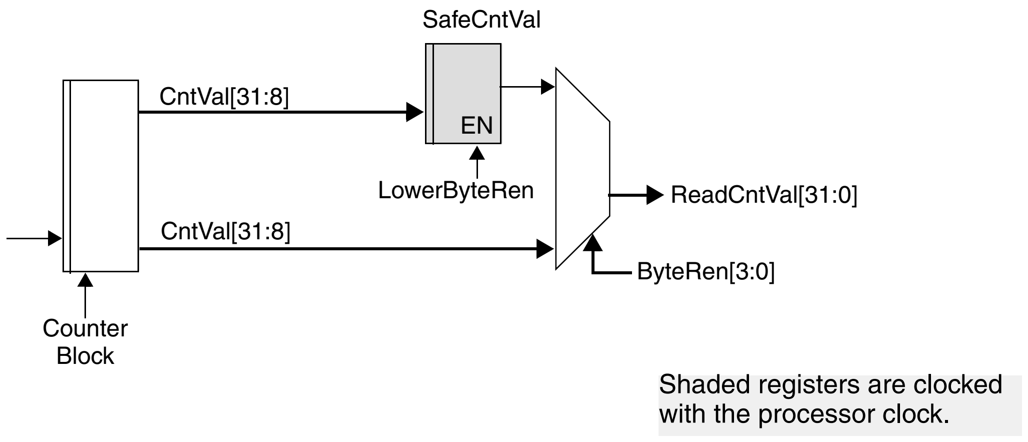
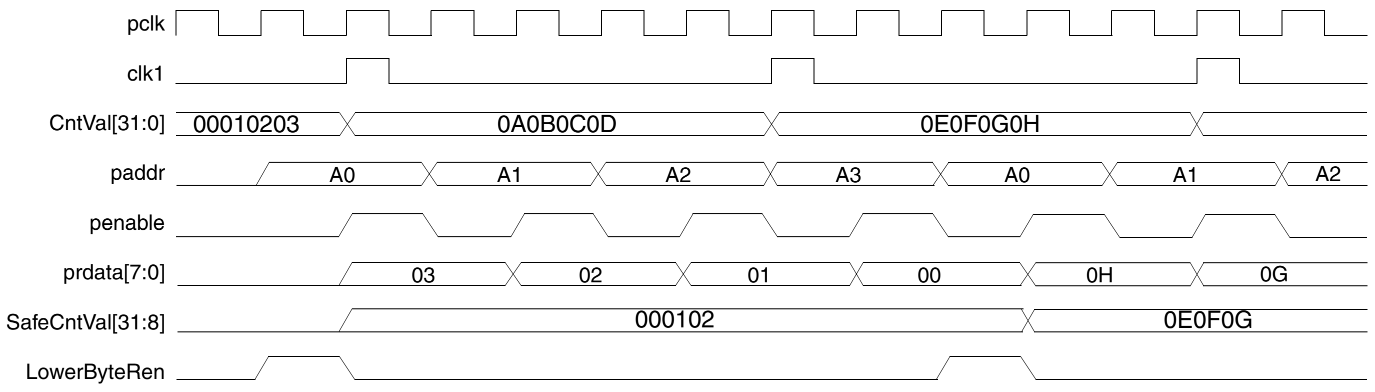


Figure 9-11 Coherent Registering – Synchronous Clocks



9.6.2.2 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.

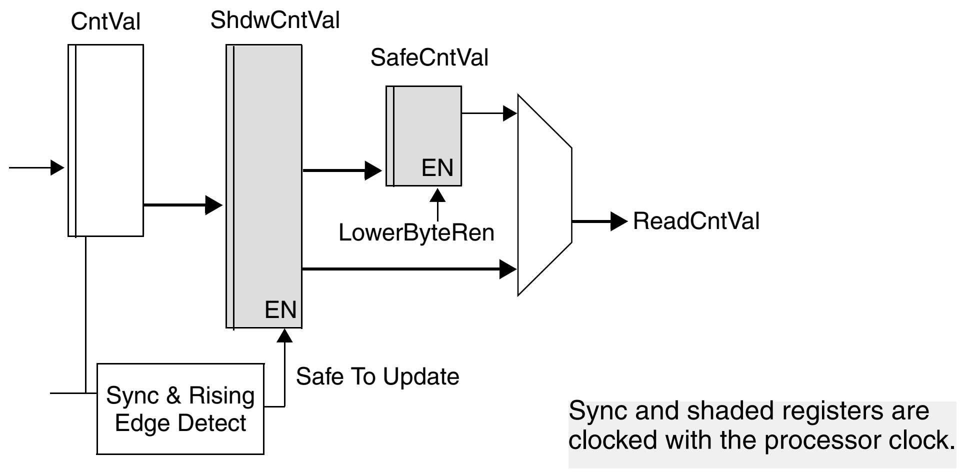
Note You must read LSB to MSB when the bus width is narrower than the counter width.

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 9-12 Coherency and Shadow Registering – Asynchronous Clocks


CntVal
ShdwCntVal
SafeCntVal
EN
LowerByteRen
ReadCntVal
EN
Safe To Update
Sync & Rising Edge Detect
Sync and shaded registers are clocked with the processor clock.

A Glossary

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

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

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

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

Index