DesignWare® DW_apb_wdt

Databook

DW_apb_wdt **–** ***Product Code***

1.12a
December 2020

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 1.03d onward.

Version	Date	Description
1.12a	December 2020	**Added:** ■ "Asynchronous Clock Mode" on page 23 ■ "Synchronization Depth" on page 24 ■ "BCM Library Components" on page 107 **Updated:** ■ Version changed for 2020.12a release ■ Chapter 7, "Verification" ■ "Performance" on page 106 ■ "Parameter Descriptions" on page 31, "Signal Descriptions" on page 41, "Register Descriptions" on page 51, and "Internal Parameter Descriptions" on page 109 auto-extracted with change bars from the RTL **Renamed:** ■ System Resets to "Clocks and Resets" ■ Synchronizer to Appendix A, "Basic Core Module (BCM) Library" **Removed:** ■ Index chapter

Version	Date	Description
1.11a	July 2018	**Updated:** ■ Version changed for 2018.07a release ■ "Performance" on page 106 ■ "Parameter Descriptions" on page 31, "Signal Descriptions" on page 41, "Register Descriptions" on page 51, and "Internal Parameter Descriptions" on page 109 auto-extracted with change bars from the RTL **Added:** ■ Support for configurable synchronization depth through parameter WDT_ASYNC_CLK_SYNC_DEPTH **Removed:** ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.
1.10a	October 2016	■ Version changed to 2016.10a ■ "Parameter Descriptions" on page 31 and "Register Descriptions" on page 51 auto-extracted from the RTL ■ Removed "Running Leda on Generated Code with coreConsultant" and reference to Leda directory in Table 2-1 ■ Removed "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Added "APB Interface" on page 27 ■ Added "System Reset When WDT_NEW_RMOD = 1" on page 25 ■ Added parameter "WDT_NEW_RMOD" in "Parameter Descriptions" on page 31 ■ Included area and power numbers for the APB4 feature "Integration Considerations" on page 87 ■ Added Xprop directory in Table 2-1and Table 2-4 ■ Added "Running VCS XPROP Analyzer" ■ Moved "Internal Parameter Descriptions" to Appendix
1.09a	June 2015	■ Added "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added "Running SpyGlass on Generated Code with coreAssembler" ■ "Signal Descriptions" on page 41 auto-extracted from the RTL ■ Added "Internal Parameter Descriptions" on page 109 ■ Added Appendix A, "Basic Core Module (BCM) Library"
1.08a	June 2014	■ Version change for 2014.06a release. ■ Added "Performance" section in the "Integration Considerations" chapter ■ Corrected External Input/Output Delay in Signals chapter
1.07d	May 2013	■ Made a minor correction in the description of WDT_EN bit of the WDT_CR register. ■ Updated the template.

Version	Date	Description
1.07c	September 2012	Added the product code on the cover and in Table 1-1.
1.07c	March 2012	Version change for 2012.03a release.
1.07b	November 2011	Version change for 2011.11a release.
1.07a	October 2011	Added material for WDT_ASYNC_CLK_MODE_ENABLE parameter.
1.06a	June 2011	Updated system diagram in Figure 1-1; enhanced "Related Documents" section in Preface.
1.06a	September 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	December 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	October 2008	Version change for 2008.10a release.
1.03e	June 2008	Version change for 2008.06a release.
1.03d	Jane 2008	■ Updated to revised installation guide and consolidated release notes ■ Changed references of "Designware AMBA" to simply "DesignWare"
1.03d	June 2007	Version change for 2007.06a release.

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.

DatabookOrganization

The chapters of this databook are organized as follows:

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

Related Documentation

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

- coreAssembler User Guide – Contains information on using coreAssembler
- coreConsultant User Guide – Contains information on using coreConsultant

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

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

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

    File > Build Debug Tar-file

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

  a. https://solvnetplus.synopsys.com

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

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

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

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

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

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

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

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

Product Code

Table 3-1 lists all the components associated with the product code for DesignWare APB Peripherals.

Table 3-1 DesignWare APB Peripherals – Product Code: 3771-0

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

1 Product Overview

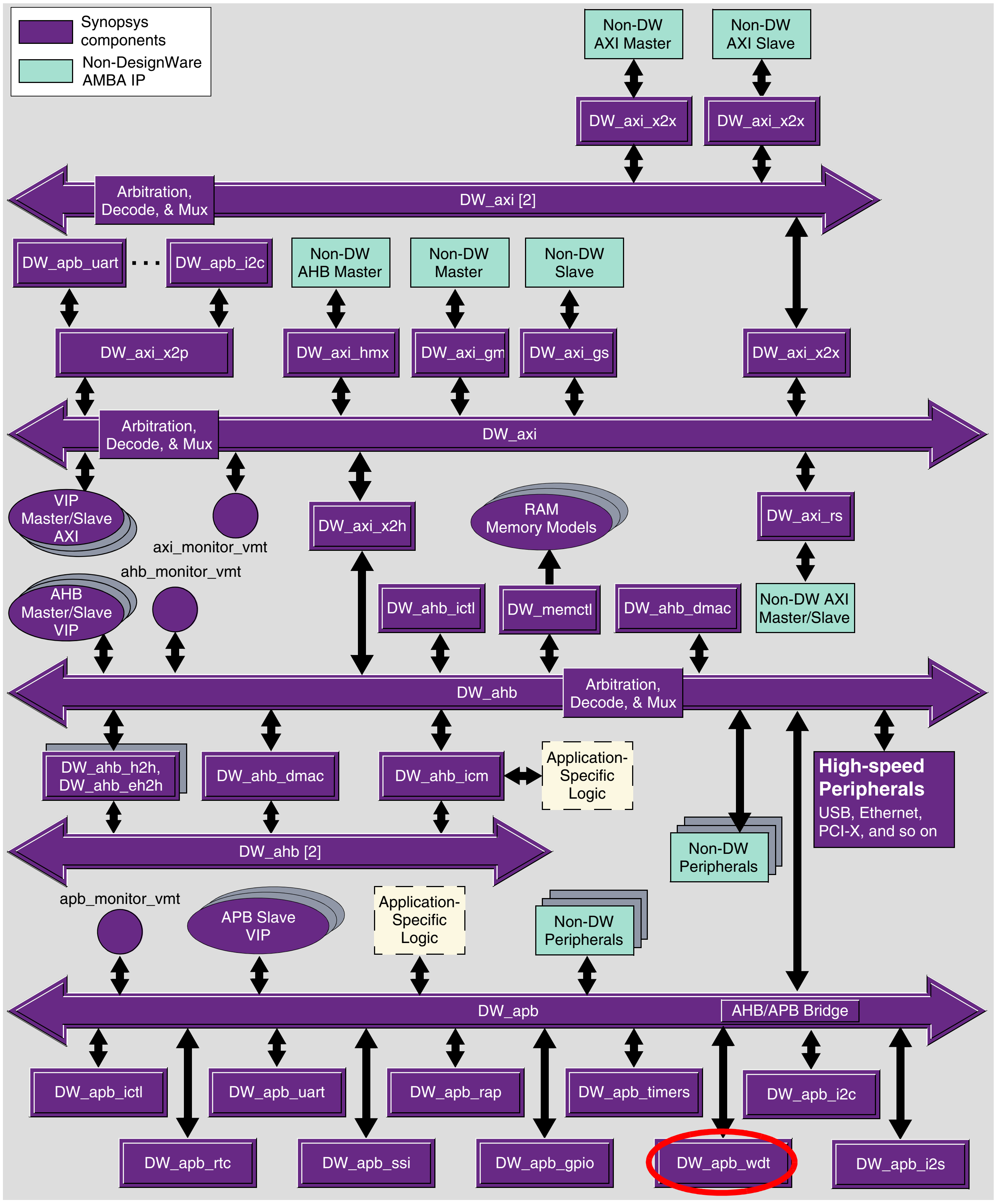
The DW_apb_wdt is a programmable Watchdog Timer (WDT) peripheral. This component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components.

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. To access the product page and documentation for AMBA components, see the DesignWare IP Solutions for AMBA Interconnect page. (SolvNetPlus ID required)

Figure 1-1 Example of DW_apb_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

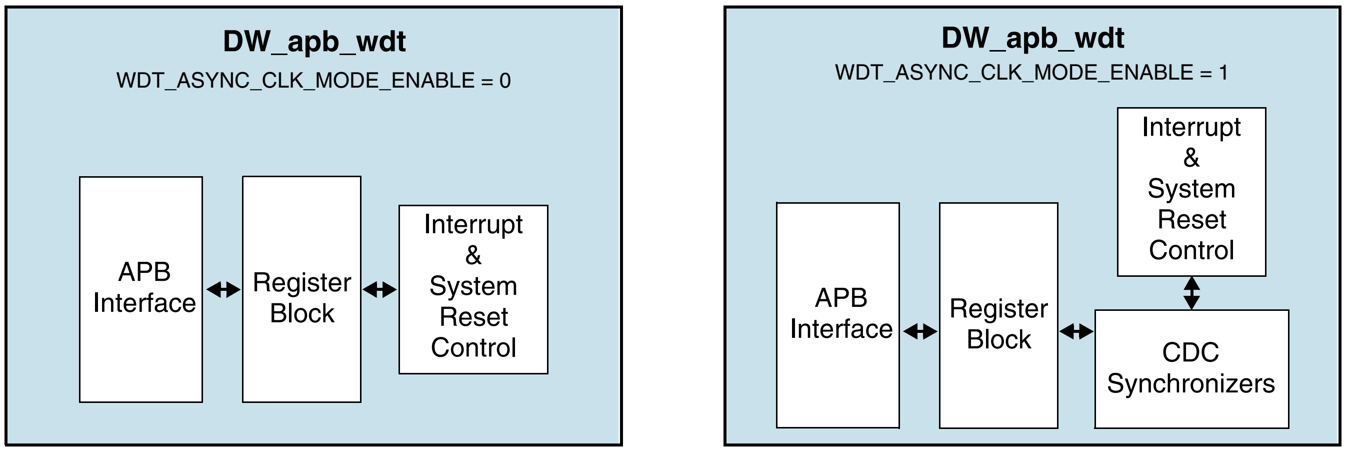
The Synopsys DW_apb_wdt is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the *AMBA Specification, Revision 2.0* from Arm®.

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

Note In Asynchronous mode when WDT_ASYNC_CLK_MODE_ENABLE=1, DW_apb_wdt is designed to generate an interrupt as soon as it is generated in timer clock domain, but de-asserts only after it is cleared in timer clock domain.

1.3 Features

The DW_apb_wdt supports the following features:

- Configurable APB2, APB3, and APB4 interface support, to allow easy integration into DesignWare Synthesizable Components for AMBA 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 even if it is cleared (or not) 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. 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 81 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, see "Licenses" in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_apb_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, see *Guide to coreTools Documentation*.

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

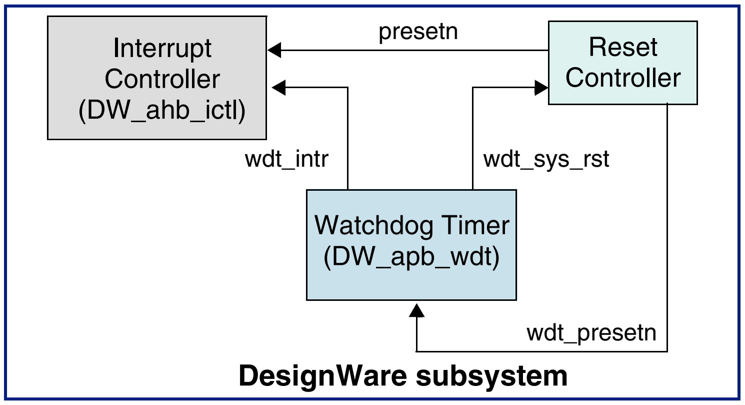
For more information about implementing your DW_apb_wdt component within a DesignWare subsystem using coreAssembler, see "Overview of the coreAssembler Configuration and Integration Process" in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

2 Functional Description

This chapter describes the 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 2-1.

Figure 2-1 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.

2.1 Counter

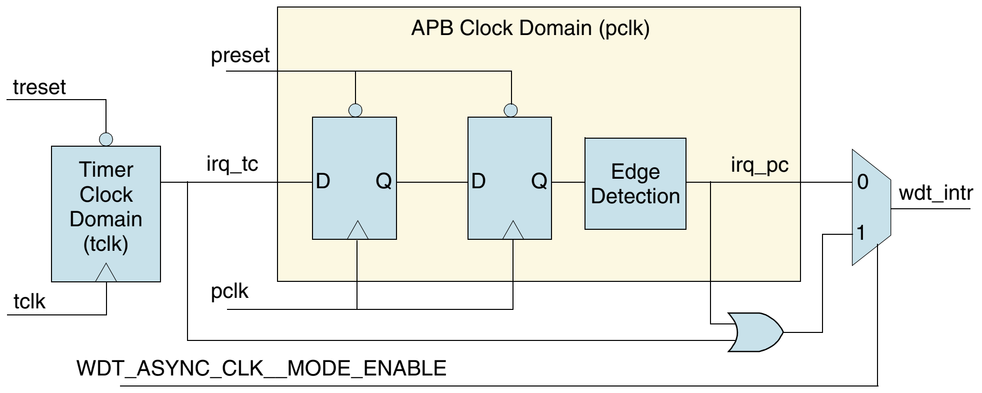
The DW_apb_wdt counts from a preset (timeout) value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. When the counter reaches zero, it wraps to the selected timeout value and continues decrementing. You 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).

2.2 Interrupts

The DW_apb_wdt can be programmed to generate an interrupt (and then a system reset) when a timeout occurs. When 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. When WDT_NEW_RMOD = 0, if the interrupt is not cleared by the time a second timeout occurs, then it generates a system reset. When WDT_NEW_RMOD = 1, even if the interrupt is cleared by the time a second timeout occurs, it generates a system reset.

Figure 2-2 shows 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 2-2 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 2-3 shows the timing diagram of the interrupt being generated and cleared when WDT_ASYNC_CLK_MODE_ENABLE = 0.

Figure 2-3 Interrupt Generation When WDT_ASYNC_CLK_MODE_ENABLE = 0

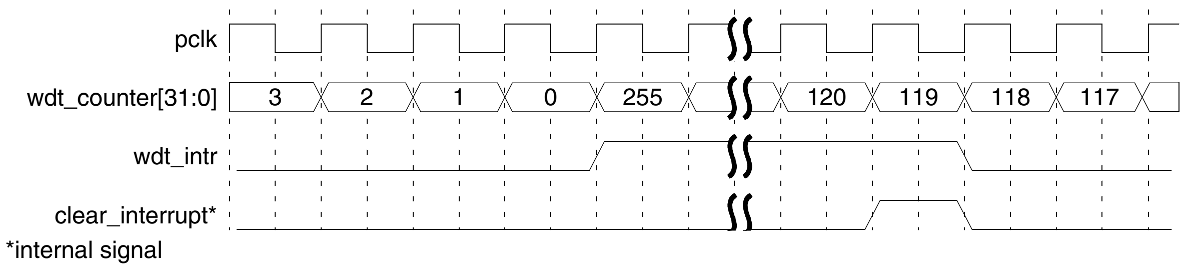
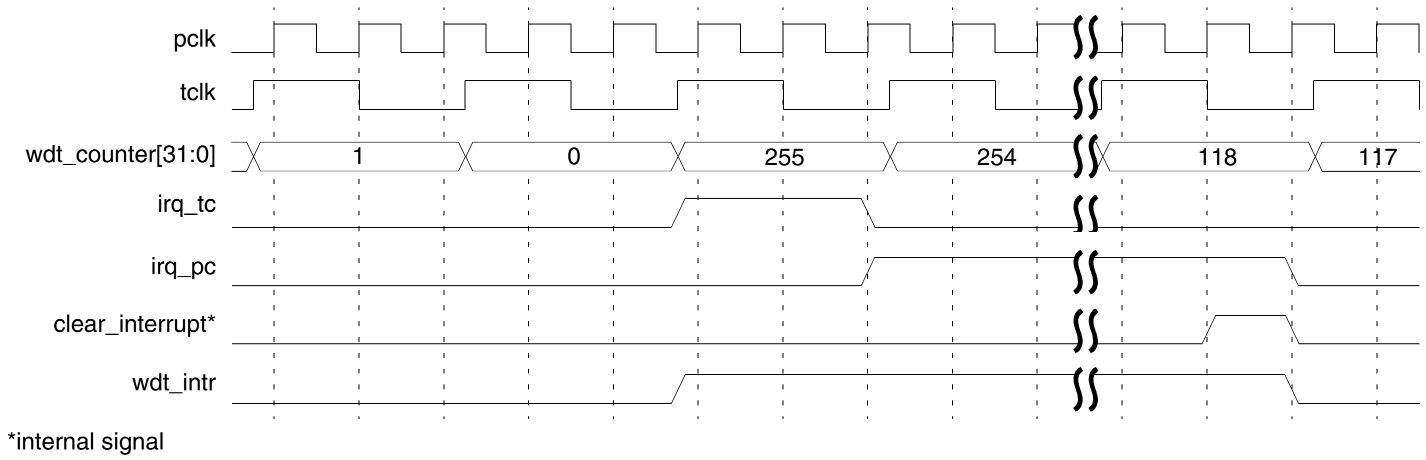


Figure 2-4 shows the timing diagram of the interrupt being generated and cleared when WDT_ASYNC_CLK_MODE_ENABLE = 1.

Figure 2-4 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.

2.3 Clocks and Resets

2.3.1 Asynchronous Clock Mode

The DW_apb_wdt uses the APB interface clock and reset signals pclk and present. When DW_apb_wdt is configured with asynchronous clock mode (WDT_ASYNC_CLK_MODE_ENABLE=1), the counter is operated with a clock that is asynchronous from the peripheral clock (pclk). When this parameter is selected, the tclk and tresetn ports are introduced to the top level port list.

2.3.2 Synchronization Depth

The WDT_ASYNC_CLK_SYNC_DEPTH coreConsultant parameter is enabled only when the WDT_ASYNC_CLK_MODE_ENABLE parameter is set to true. In this mode, the user can select the synchronization depth when crossing clock domains between APB and WDT.

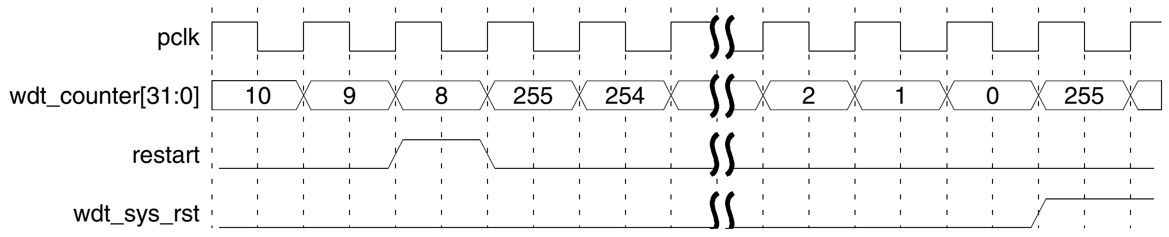
- 2 - 2-stage synchronization with positive-edge capturing at both the stages
- 3 - 3-stage synchronization with positive-edge capturing at all stages
- 4 - 4-stage synchronization with positive-edge capturing at all stages

2.3.3 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 2-5 shows the timing diagram of a counter restart and the generation of a system reset.

Figure 2-5 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:

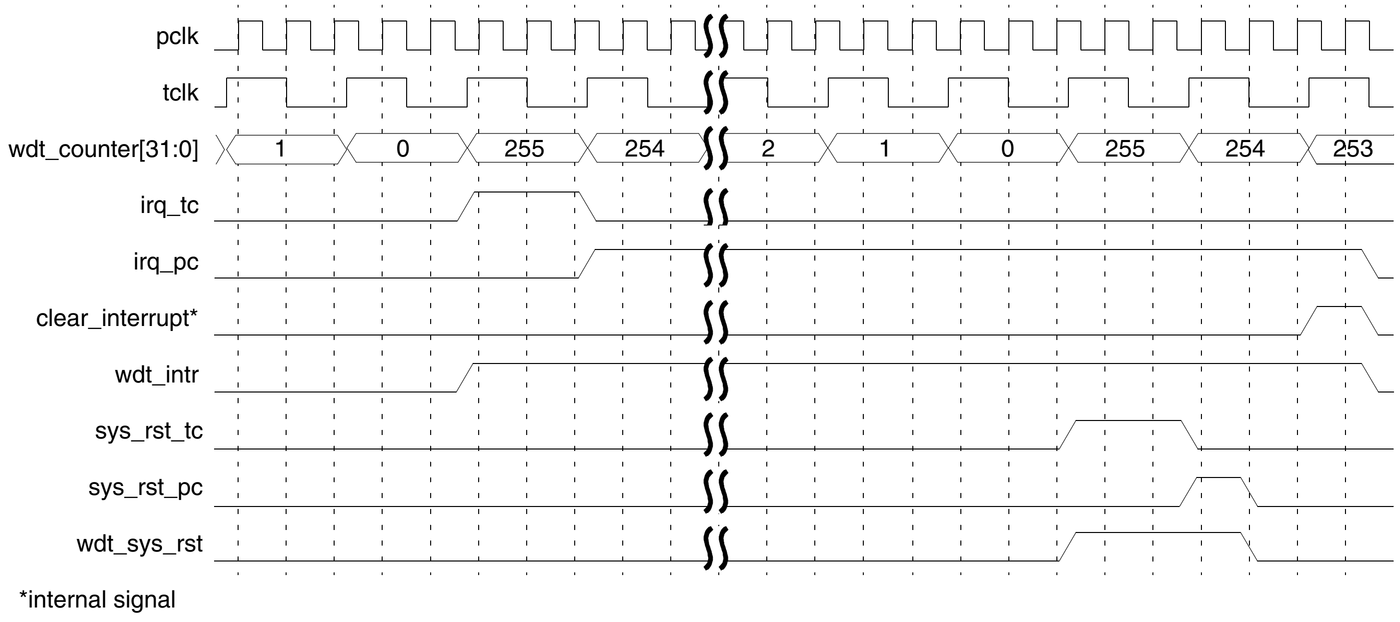
N pclk cycles of *synchronization_delay* + WDT_CR.RPL (programmed Reset Pulse Length) where,

N = WDT_ASYNC_CLK_SYNC_DEPTH.

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 2-6 shows the timing diagram of the generation of a system reset when WDT_ASYNC_CLK_MODE_ENABLE = 1.

Figure 2-6 System Reset Generation



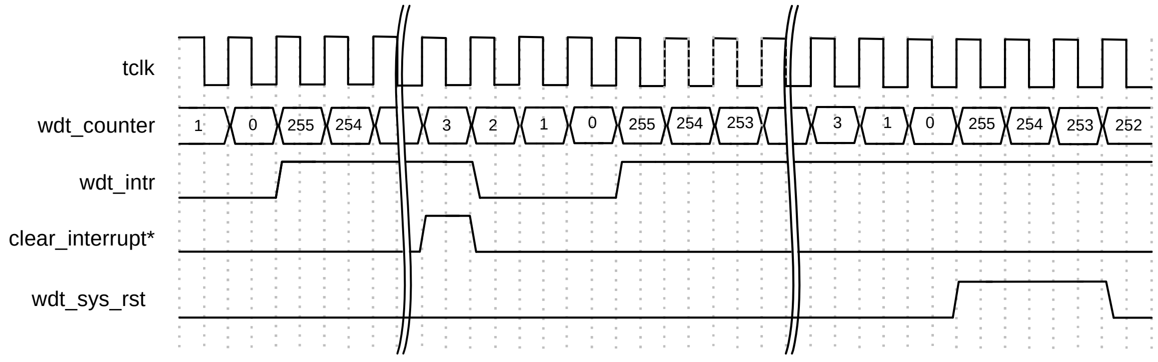
Note The wdt_sys_rst system reset, generated when WDT_ASYNC_CLK_MODE_ENABLE = 1, is asynchronous because it asserts on the rising edge of tclk and de-asserts on the rising edge of pclk.

2.3.4 System Reset When WDT_NEW_RMOD = 1

DW_apb_wdt can be programmed to generate the wdt_intr (and then a system reset) when a timeout occurs, that is when the response mode field of the Watchdog Timer Control Register (WDT_CR.RMOD) is set to 1.

Figure 2-7 shows the timing diagram when WDT_NEW_RMOD is set to 0, which is the default system reset behavior (see "Clocks and Resets" on page 23) in which DW_apb_wdt generates a system reset, if the interrupt is not cleared by the time a second timeout occurs.

Figure 2-7 WDT System Reset Timing Diagram When WDT_NEW_RMOD = 0



When WDT_NEW_RMOD is set to 1, irrespective of whether the wdt_intr is cleared (by reading the WDT_EOI register) and if the second timeout occurs, DW_apb_wdt generates a system reset. For subsequent timeouts, both the interrupt and the system reset are generated. However, this system reset generation can be avoided by performing a restart, that is by writing 0x76 into the WDT_CRR register before the second or subsequent timeout occurs. This behavior of a system reset (when WDT_NEW_RMOD = 1) is enabled only if the Response Mode is set to 1 either through the configuration (that is, if (WDT_HC_RMOD==1 && WDT_DFLT_RMOD==1)) or by programming the WDT_CR.RMOD register bit to 1.

Figure 2-8 and Figure 2-9 shows WDT system reset timing diagrams when the wdt_intr is cleared during the first timeout and not cleared during the first timeout, respectively.

Figure 2-8 WDT System Reset Timing Diagram When wdt_intr is Cleared During First Timeout (WDT_NEW_RMOD=1)

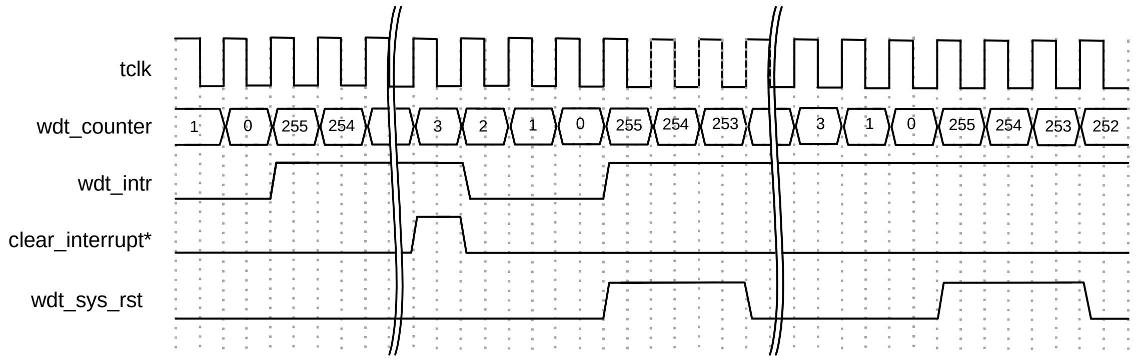
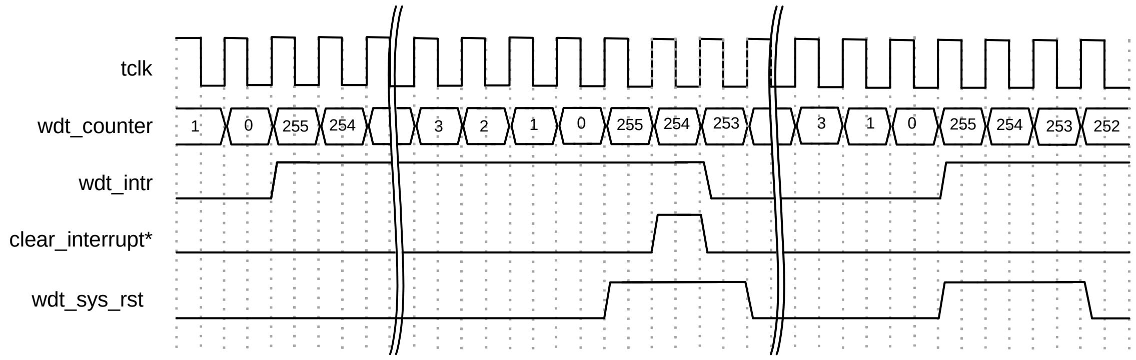


Figure 2-9 WDT System Reset Timing Diagram When wdt_intr is Not Cleared During First Timeout (WDT_NEW_RMOD=1)



2.4 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.

2.5 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.

2.6 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 2-1 on page 21). A counter restart has no effect on the system reset once it has been asserted.

2.7 Timeout Period Values

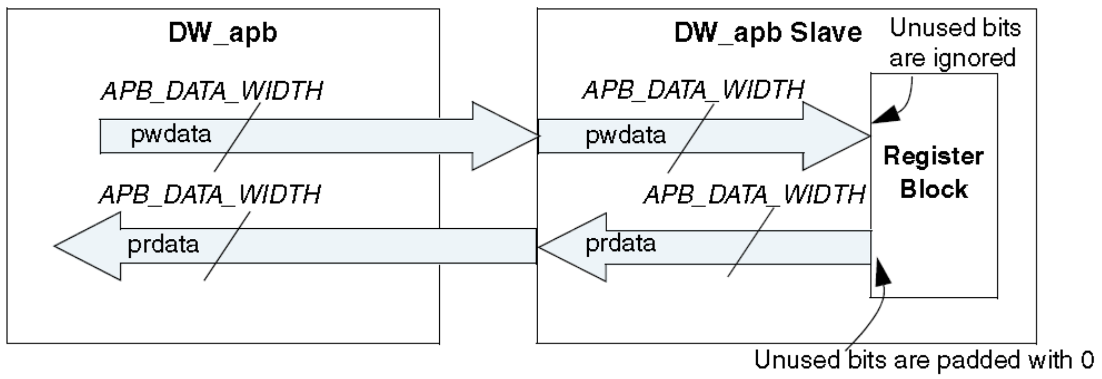
The DW_apb_wdt can be configured to have a fixed or user-defined timeout period range. In both cases, the values that may be selected are limited by the WDT counter width. If the timeout period range that is selected is greater than the counter width, the timeout period is truncated to fit to the counter 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.

2.8 APB Interface

The DW_apb_wdt peripheral has a standard AMBA 2.0 APB interface for reading and writing the internal registers. The host processor accesses internal registers on the DW_apb_wdt peripheral through the AMBA APB 2.0/3.0/4.0 interface. This peripheral supports APB data bus widths of 8, 16, or 32 bits, which is set with the APB_DATA_WIDTH parameter.

Figure 2-10 shows the read/write buses between the DW_apb and the APB slave.

Figure 2-10 Read/Write Buses Between the DW_apb and an APB Slave



The data, control and status registers within the DW_apb_wdt are byte-addressable. The maximum width of the control or status register (except for WDT Component Version register, component parameters registers and component type register) in the DW_apb_wdt is 8 bits. Therefore, if the APB data bus is 8, 16, or 32 bits wide, all read and write operations to the DW_apb_wdt control and status registers require only one APB access.

The WDT_CCVR register width depends on the WDT_CNT_WIDTH parameter, which can vary from 8 to 32. Depending on the width of timer and width of APB data bus (APB_DATA_WIDTH), the APB interface may need to perform single or multiple accesses to the previously mentioned registers.

"Integration Considerations" on page 87 provides more information about reading to and writing from the APB interface.

The APB 3 and APB4 register accesses to the DW_apb_wdt peripheral are discussed in the following sections:

- "APB 3.0 Support" on page 28
- "APB 4.0 Support" on page 29

2.8.1 APB 3.0 Support

The DW_apb_wdt register interface is compliant with the AMBA APB 2.0, APB 3.0 and APB 4.0 specifications. The SLAVE_INTERFACE_TYPE parameter is used to select the APB interface type of the register interface. To comply with the AMBA APB 3.0 specification, DW_apb_wdt supports the following signals:

- PREADY – This signal is always set to its default value that is high for all APB processes.

Note DW_apb_wdt does not use the PREADY signal and it used only for interface consistency.

- PSLVERR – This signal issues an error when protected registers are accessed without relevant authorization levels. The PSLVERR signal is enabled when the SLVERR_RESP_EN parameter is set to 1, so that DW_apb_wdt provides any slave error response from register interface. For more information on this signal, see "APB 4.0 Support" on page 29.

2.8.2 APB 4.0 Support

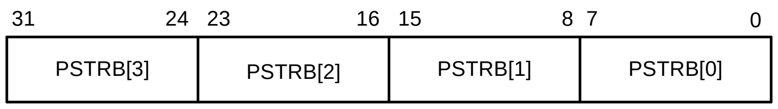
The DW_apb_wdt register interface is compliant with the AMBA APB 2.0, APB 3.0 and APB 4.0 specifications. To comply with the AMBA APB 4.0 specification, DW_apb_wdt supports the following signals:

- PSTRB – This signal specifies the APB4 write strobe bus. In a write transaction to the ABP 4.0 register interface, the PSTRB signal indicates validity of PWDATA bytes. DW_apb_wdt selectively writes to the bytes of the addressed register whose corresponding bit in the PSTRB signal is high. Bytes strobed low by the corresponding PSTRB bits are not modified.

  The incoming strobe bits for a read transaction is always zero as per protocol.

  Figure 2-11 shows the byte lane mapping of the PSTRB signal.

Figure 2-11 Byte Lane Mapping of the PSTRB Signal



- PPROT – This signal supports the protection feature of the APB4 protocol. The APB4 protection feature is supported only on the WDT_TORR register. The protection level register (WDT_PROT_LEVEL) defines the APB4 protection level, that is the protected register (WDT_TORR) is updated only if the PPROT privilege is more than the protection privilege programmed in the protection level register (see Table 2-1). Otherwise, PSLVERR is asserted and the protected register is not updated, provided that PSLVERR_RESP_EN is set as high. If the PSLVERR_RESP_EN is low, then protection feature and PSLVERR generation logic is not implemented. A PSLVERR is asserted for the PPROT and WDT_PROT_LEVEL combinations as shown in Table 2-1. For all other combinations, PSLVERR is driven LOW.

Table 2-1 PPROT Level, Protection Level Programmed in WDT_PROT_LEVEL, and Slave Error Response

PPROT			WDT_PROT_LEVEL			
[2]	[1]	[0]	[2]	[1]	[0]	PSLVERR
X	X	0	X	X	1	HIGH
X	1	X	X	0	X	HIGH
0	X	X	1	X	X	HIGH

3 Parameter Descriptions

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

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

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

- Top Level Parameters on page 32
- Timeout Configuration on page 38

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
System Configuration	
APB data bus width	Width of the APB Data Bus to which this component is attached. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Register Interface Type	Select Register Interface type as APB2, APB3 or APB4. By default, DW_apb_wdt supports APB2 interface. **Values:** ■ APB2 (0) ■ APB3 (1) ■ APB4 (2) **Default Value:** APB2 **Enabled:** Always **Parameter Name:** SLAVE_INTERFACE_TYPE
Slave Error Response Enable	Enable Slave Error response signaling. The component will refrain From signaling an error response if this parameter is disabled. This will result in disabling all features that require PSLVERR functionality to be implemented. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** SLAVE_INTERFACE_TYPE>1 **Parameter Name:** SLVERR_RESP_EN
WDT Protection Level	Reset Value of WDT_PROT_LEVEL register. A high on any bit of WDT protection level requires a high on the corresponding pprot input bit to gain access to the load-count registers. Else, SLVERR response is triggered. A zero on the protection bit will provide access to the register if other protection levels are satisfied. **Values:** 0x0, ..., 0x7 **Default Value:** 0x2 **Enabled:** SLAVE_INTERFACE_TYPE>1 && SLVERR_RESP_EN==1 **Parameter Name:** PROT_LEVEL_RST

Table 3-1 Top Level Parameters (Continued)

Label	Description
Hard-Code Protection Level?	Checking this parameter makes WDT_PROT_LEVEL a read-only register. The register can be programmed at run-time by a user if this hard-code option is set to zero. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** SLAVE_INTERFACE_TYPE>1 && SLVERR_RESP_EN==1 **Parameter Name:** HC_PROT_LEVEL
WDT Counter	
WDT counter width	The Watchdog Timer counter width. **Values:** 16, ..., 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** WDT_CNT_WIDTH
Enable WDT from reset?	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. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_ALWAYS_EN
Interrupt and System Restart	
Active high interrupt polarity?	This parameter 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. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** !(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==0) **Parameter Name:** WDT_INT_POL

Table 3-1 Top Level Parameters (Continued)

Label	Description
Active high reset polarity?	This parameter 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. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** WDT_RST_POL
Hard code reset pulse length?	Configures the reset pulse length to be hard coded. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_HC_RPL

Table 3-1 Top Level Parameters (Continued)

Label	Description
Default reset pulse length	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_MODE_ENABLE~~WDT_ASYNC_CLK_MODE_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" section of DW_apb_wdt Databook. **Values:** ~~0x0, ..., 0x7~~ ■ 2 pclk cycles (0x0) ■ 4 pclk cycles (0x1) ■ 8 pclk cycles (0x2) ■ 16 pclk cycles (0x3) ■ 32 pclk cycles (0x4) ■ 64 pclk cycles (0x5) ■ 128 pclk cycles (0x6) ■ 256 pclk cycles (0x7) **Default Value:** ~~0x0~~2 pclk cycles **Enabled:** Always **Parameter Name:** WDT_DFLT_RPL
Hard code output response mode?	Configures the output response mode to be hard coded. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_HC_RMOD

Table 3-1 Top Level Parameters (Continued)

Label	Description
Output response mode default value?	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 equals 0 and an interrupt followed by a system reset, if equals 1. If WDT_HC_RMOD is 1, then default response mode is the only possible output response mode. **Values:** ■ System reset only (0x0) ■ Interrupt and system reset (0x1) **Default Value:** System reset only **Enabled:** Always **Parameter Name:** WDT_DFLT_RMOD
Enable new response mode feature?	If this parameter is enabled, generate an interrupt when first timeout occurs. Irrespective of interrupt got cleared or not by the time a second timeout occurs, generate the system reset. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** (WDT_HC_RMOD==0)||(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==1) **Parameter Name:** WDT_NEW_RMOD
External Configuration	
Include WDT counter clock enable input on I/F?	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. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** WDT_ASYNC_CLK_MODE_ENABLE==0 **Parameter Name:** WDT_CLK_EN
Are the Counter clock and the Peripheral clock asynchronous?	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. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_ASYNC_CLK_MODE_ENABLE

Table 3-1 Top Level Parameters (Continued)

Label	Description
Clock Domain Crossing Synchronization Depth?	Sets the number of synchronization stages to be placed on clock domain crossing signals. ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages **Values:** 2, 3, 4 **Default Value:** 2 **Enabled:** WDT_ASYNC_CLK_MODE_ENABLE==1 **Parameter Name:** WDT_ASYNC_CLK_SYNC_DEPTH
Include pause enable input on I/F?	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. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_PAUSE

3.2 Timeout Configuration Parameters

Table 3-2 Timeout Configuration Parameters

Label	Description
Timeout Configuration	
Hard code timeout period range selection?	When set to 1, the selected timeout period(s) is set to be hard coded. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_HC_TOP
Use pre-defined timeout period ranges?	When this parameter is set to 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 parameter is set to 0, the user must define the timeout period range (2^8 to 2^(WDT_CNT_WIDTH)-1) using the WDT_USER_TOP_(i) parameter. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** WDT_USE_FIX_TOP
Main Timeout	
Main timeout period range to be selected as default	Selects the timeout period that is available directly after reset. It controls the reset value of the register. If WDT_HC_TOP is set to 1, then the default timeout period is the only possible timeout period. Can choose one of 16 values. **Values:** 0, ..., 15 **Default Value:** 0 **Enabled:** Always **Parameter Name:** WDT_DFLT_TOP
User defined main timeout period for range 0 to 15 (for i = 0; i <= 15)	Describes the user-defined main timeout period for range i that is selected when WDT_TORR bits [3:0] (TOP) are equal to i. **Values:** 0xff, ..., WDT_USER_TOP_MAXVALUE **Default Value:** ~~0xffff~~For i=0: 0x0000ffff; for i=1: 0x0001ffff; for i=2: 0x0003ffff; for i=3: 0x0007ffff; for i=4: 0x000fffff; for i=5: 0x001fffff; for i=6: 0x003fffff; for i=7: 0x007fffff; for i=8: 0x00ffffff; for i=9: 0x01ffffff; for i=10: 0x03ffffff; for i=11: 0x07ffffff; for i=12: 0x0fffffff; for i=13: 0x1fffffff; for i=14: 0x3fffffff; for i=15: 0x7fffffff **Enabled:** ((WDT_USE_FIX_TOP == 0) AND !((WDT_HC_TOP == 1) AND (WDT_DFLT_TOP!=i))) **Parameter Name:** WDT_USER_TOP_(i)

Table 3-2 Timeout Configuration Parameters (Continued)

Label	Description
	Initial Timeout
Include a second timeout period for initialization?	When set to 1, includes a second timeout period that is used for initialization prior to the first kick. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** WDT_DUAL_TOP
Initial timeout period range to be selected as default	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. **Values:** 0, ..., 15 **Default Value:** 0 **Enabled:** WDT_DUAL_TOP==1 **Parameter Name:** WDT_DFLT_TOP_INIT
User defined initial timeout period for range 0 to 15 (for i = 0; i <= 15)	Describes the user-defined initial timeout period for range i that is selected when WDT_TORR bits [7:4] (TOP_INIT) are equal to i. **Values:** 0xff, ..., WDT_USER_TOP_MAXVALUE **Default Value:** ~~0xffff~~For i=0: 0x0000ffff; for i=1: 0x0001ffff; for i=2: 0x0003ffff; for i=3: 0x0007ffff; for i=4: 0x000fffff; for i=5: 0x001fffff; for i=6: 0x003fffff; for i=7: 0x007fffff; for i=8: 0x00ffffff; for i=9: 0x01ffffff; for i=10: 0x03ffffff; for i=11: 0x07ffffff; for i=12: 0x0fffffff; for i=13: 0x1fffffff; for i=14: 0x3fffffff; for i=15: 0x7fffffff **Enabled:** (WDT_DUAL_TOP == 1) AND (WDT_USE_FIX_TOP == 0) AND !((WDT_HC_TOP==1) AND (WDT_DFLT_TOP_INIT!=i)) AND !((WDT_ALWAYS_EN==1) AND (WDT_DFLT_TOP_INIT!=i)) **Parameter Name:** WDT_USER_TOP_INIT_(i)

4 Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- APB Interface on page 43
- Application Interface on page 46
- Interrupt Interface on page 48
- System Reset Interface on page 49

4.1 APB Interface Signals

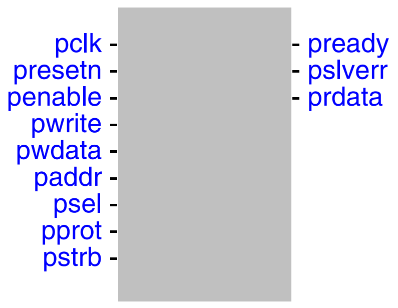


Table 4-1 APB Interface Signals

Port Name	I/O	Description
pclk	I	APB clock - This clock times all bus transfers. All signal timings are related to the rising edge of pclk. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
presetn	I	APB Reset Signal - The bus reset signal is used to reset the system and the bus on the DesignWare interface. Asynchronous 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. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
penable	I	APB enable control that indicates the second cycle of the APB frame. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 APB Interface Signals (Continued)

Port Name	I/O	Description
pwrite	I	APB write control. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
paddr[(WDT_ADDR_SLICE_LHS-1):0]	I	APB address bus; uses the lower bits of the APB address bus for register decode. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
psel	I	APB peripheral select. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pprot[2:0]	I	APB4 Protection type. The input bits should match the corresponding protection activated level bit of the accessed register to gain access to the WDT load-count registers. Else the DW_apb_wdt generates an error. If protection level is turned off, any value on the corresponding bit is acceptable. This Signal is ignored if SLVERR_RESP_EN==0. **Exists:** SLAVE_INTERFACE_TYPE>1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 APB Interface Signals (Continued)

Port Name	I/O	Description
pstrb[((APB_DATA_WIDTH/8)-1):0]	I	APB4 Write strobe bus. A high on individual bits in the pstrb bus indicate that the corresponding incoming write data byte on APB bus is to be updated in the addressed register. **Exists:** SLAVE_INTERFACE_TYPE>1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pready	O	This APB3 protocol signal indicates the end of a transaction when high in the access phase of a transaction. PREADY never goes low in DW_apb_wdt and is tied to one. **Exists:** SLAVE_INTERFACE_TYPE>0 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pslverr	O	APB3 slave error response signal. The signal issues an error when some error condition occurs, as specified in Slave error response section. **Exists:** SLAVE_INTERFACE_TYPE>0 **Synchronous To:** pclk **Registered:** (SLAVE_INTERFACE_TYPE>1 && SLVERR_RESP_EN==1) ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prdata[(APB_DATA_WIDTH-1):0]	O	APB read data. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Application Interface Signals

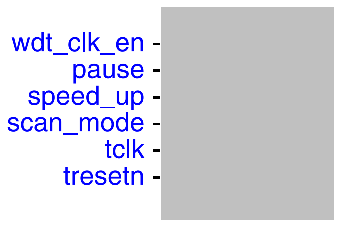


Table 4-2 Application Interface Signals

Port Name	I/O	Description
wdt_clk_en	I	Optional. Watchdog counter clock enable. Used to control the rate at which the counter decrements. **Exists:** WDT_CLK_EN==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pause	I	Optional. Pause enable. Used to freeze the watchdog counter during pause mode. **Exists:** WDT_PAUSE==1 **Synchronous To:** pclk **Registered:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
speed_up	I	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. **Exists:** Always **Synchronous To:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? "tclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-2 Application Interface Signals (Continued)

Port Name	I/O	Description
scan_mode	I	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. **Exists:** Always **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
tclk	I	Optional. Watchdog timer clock that can be asynchronous to pclk. **Exists:** WDT_ASYNC_CLK_MODE_ENABLE==1 **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
tresetn	I	Optional. Watchdog timer reset; used to reset the watchdog counter. 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. reuse-pragma attr PowerDomain SINGLE_DOMAIN **Exists:** WDT_ASYNC_CLK_MODE_ENABLE==1 **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.3 Interrupt Interface Signals

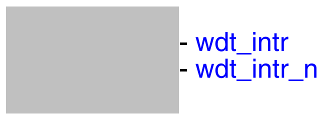


Table 4-3 Interrupt Interface Signals

Port Name	I/O	Description
wdt_intr	O	Optional. Active-High WDT interrupt. **Note:** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and deasserts on the rising edge of pclk. **Exists:** WDT_INT_POL==1 && !(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==0) **Synchronous To:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? "None" : "pclk" **Registered:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdt_intr_n	O	Optional. Active-Low WDT interrupt. **Note:** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and deasserts on the rising edge of pclk. **Exists:** WDT_INT_POL==0 && !(WDT_HC_RMOD==1 && WDT_DFLT_RMOD==0) **Synchronous To:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? "None" : "pclk" **Registered:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.4 System Reset Interface Signals

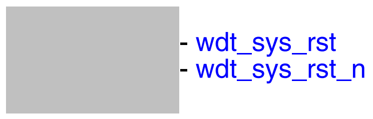


Table 4-4 System Reset Interface Signals

Port Name	I/O	Description
wdt_sys_rst	O	Optional. Active-high system reset flag. **Note:** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and deasserts on the rising edge of pclk. **Exists:** WDT_RST_POL==1 **Synchronous To:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? "None" : "pclk" **Registered:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdt_sys_rst_n	O	Optional. Active-Low system reset flag. **Note:** When WDT_ASYNC_CLK_MODE_ENABLE = 1, this output is asynchronous because it asserts on the rising edge of tclk and deasserts on the rising edge of pclk. **Exists:** WDT_RST_POL==0 **Synchronous To:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? "None" : "pclk" **Registered:** (WDT_ASYNC_CLK_MODE_ENABLE==1) ? No : Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low

5 Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
wdt_address_block on page 54	DW_apb_wdt register address block **Exists:** Always

5.1 wdt_memory_map/wdt_address_block Registers

~~wdt_address_block~~DW_apb_wdt register address block. Follow the link for the register to see a detailed description of the register.

Table 5-5 Registers for Address Block: wdt_memory_map/wdt_address_block

Register	Offset	Description
WDT_CR on page 55	0x0	Control Register
WDT_TORR on page 58	0x4	Timeout Range Register
WDT_CCVR on page 63	0x8	Current Counter Value Register~~.~~
WDT_CRR on page 64	0xc	Counter Restart Register
WDT_STAT on page 65	0x10	Interrupt Status Register
WDT_EOI on page 66	0x14	Interrupt Clear Register
WDT_PROT_LEVEL on page 67	0x1c	WDT Protection level ~~register~~
WDT_COMP_PARAM_5 on page 68	0xe4	Component Parameters Register 5 ~~Note: This is a constant read-only register that contains encoded...~~
WDT_COMP_PARAM_4 on page 69	0xe8	Component Parameters Register 4 ~~Note: This is a constant read-only register that contains encoded...~~
WDT_COMP_PARAM_3 on page 70	0xec	Component Parameters Register 3 ~~Note: This is a constant read-only register that contains encoded...~~
WDT_COMP_PARAM_2 on page 71	0xf0	Component Parameters Register 2 ~~Note: This is a constant read-only register that contains encoded...~~
WDT_COMP_PARAM_1 on page 72	0xf4	Component Parameters Register 1 ~~Note:This is a constant read-only register that contains encoded...~~
WDT_COMP_VERSION on page 76	0xf8	Component Version Register
WDT_COMP_TYPE on page 77	0xfc	Component Type Register

5.1.1 WDT_CR

- **Name:** Control Register
- **Description:** Control Register
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

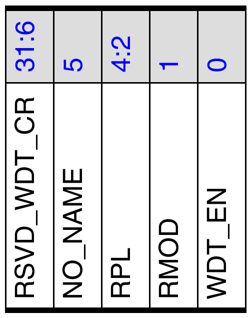


Table 5-6 Fields for Register: WDT_CR

Bits	Name	Memory Access	Description
31:6	RSVD_WDT_CR	R	WDT_CR[31:6] Reserved bits and read as zero (0). **Value After Reset:** 0x0 **Exists:** Always
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. **Value After Reset:** 0x0 **Exists:** Always

Table 5-6 Fields for Register: WDT_CR (Continued)

Bits	Name	Memory Access	Description
4:2	RPL	* Varies	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. **Note:** When WDT_ASYNC_CLK_MODE_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" section of DW_apb_wdt Databook. **Values:** ■ 0x0 (PCLK_CYCLES_2): 2 pclk cycles ■ 0x1 (PCLK_CYCLES_4): 4 pclk cycles ■ 0x2 (PCLK_CYCLES_8): 8 pclk cycles ■ 0x3 (PCLK_CYCLES_16): 16 pclk cycles ■ 0x4 (PCLK_CYCLES_32): 32 pclk cycles ■ 0x5 (PCLK_CYCLES_64): 64 pclk cycles ■ 0x6 (PCLK_CYCLES_128): 128 pclk cycles ■ 0x7 (PCLK_CYCLES_256): 256 pclk cycles **Value After Reset:** WDT_DFLT_RPL **Exists:** Always **Memory Access:** {(WDT_HC_RPL == 1) ? "read-only" : "read-write"}
1	RMOD	* Varies	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. **Values:** ■ 0x0 (RESET): Generate a system reset ■ 0x1 (INTERRUPT): First generate an interrupt and even if it is cleared by the time a second timeout occurs then generate a system reset **Value After Reset:** WDT_DFLT_RMOD **Exists:** Always **Memory Access:** {(WDT_HC_RMOD == 1) ? "read-only" : "read-write"}

Table 5-6 Fields for Register: WDT_CR (Continued)

Bits	Name	Memory Access	Description
0	WDT_EN	* Varies	WDT enable. When the configuration parameter WDT_ALWAYS_EN = 0, this bit can be set; otherwise, it is read-only. 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. The DW_apb_wdt is used to prevent system lock-up. To prevent a software bug from disabling the DW_apb_wdt, once this bit has been enabled, it can be cleared only by a system reset. **Values:** ■ 0x0 (DISABLED): Watchdog timer disabled ■ 0x1 (ENABLED): Watchdog timer enabled **Value After Reset:** WDT_ALWAYS_EN **Exists:** Always **Memory Access:** {(WDT_ALWAYS_EN == 1) ? "read-only" : "read-write"}

5.1.2 WDT_TORR

- **Name:** Timeout Range Register
- **Description:** Timeout Range Register
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** Always

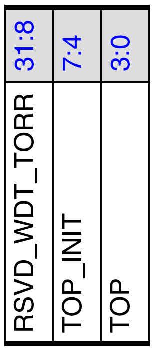


Table 5-7 Fields for Register: WDT_TORR

Bits	Name	Memory Access	Description
31:8	RSVD_WDT_TORR	R	WDT_TORR[31:8] Reserved and read as zero (0). **Value After Reset:** 0x0 **Exists:** Always

Table 5-7 Fields for Register: WDT_TORR (Continued)

Bits	Name	Memory Access	Description
7:4	TOP_INIT	* Varies	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) **Note:** These bits exist only when the configuration parameter WDT_DUAL_TOP = 1, otherwise, they are fixed at zero. **Values:** ■ 0x0 (USER0_OR_64K): Time out of WDT_USER_TOP_INIT_0 or 64K Clocks ■ 0x1 (USER1_OR_128K): Time out of WDT_USER_TOP_INIT_1 or 128K Clocks ■ 0x2 (USER2_OR_256K): Time out of WDT_USER_TOP_INIT_2 or 256K Clocks ■ 0x3 (USER3_OR_512K): Time out of WDT_USER_TOP_INIT_3 or 512K Clocks ■ 0x4 (USER4_OR_1M): Time out of WDT_USER_TOP_INIT_4 or 1M Clocks ■ 0x5 (USER5_OR_2M): Time out of WDT_USER_TOP_INIT_5 or 2M Clocks

Table 5-7 Fields for Register: WDT_TORR (Continued)

Bits	Name	Memory Access	Description
			■ 0x6 (USER6_OR_4M): Time out of WDT_USER_TOP_INIT_6 or 4M Clocks ■ 0x7 (USER7_OR_8M): Time out of WDT_USER_TOP_INIT_7 or 8M Clocks ■ 0x8 (USER8_OR_16M): Time out of WDT_USER_TOP_INIT_8 or 16M Clocks ■ 0x9 (USER9_OR_32M): Time out of WDT_USER_TOP_INIT_9 or 32M Clocks ■ 0xa (USER10_OR_64M): Time out of WDT_USER_TOP_INIT_10 or 64M Clocks ■ 0xb (USER11_OR_128M): Time out of WDT_USER_TOP_INIT_11 or 128M Clocks ■ 0xc (USER12_OR_256M): Time out of WDT_USER_TOP_INIT_12 or 256M Clocks ■ 0xd (USER13_OR_512M): Time out of WDT_USER_TOP_INIT_13 or 512M Clocks ■ 0xe (USER14_OR_1G): Time out of WDT_USER_TOP_INIT_14 or 1G Clocks ■ 0xf (USER15_OR_2G): Time out of WDT_USER_TOP_INIT_15 or 2G Clocks **Value After Reset:** "(WDT_DUAL_TOP==1) ? WDT_DFLT_TOP_INIT : 0x0" **Exists:** WDT_DUAL_TOP == 1 **Memory Access:** {(WDT_DUAL_TOP==0 || WDT_HC_TOP==1 || WDT_ALWAYS_EN==1) ? "read-only" : "read-write"}

Table 5-7 Fields for Register: WDT_TORR (Continued)

Bits	Name	Memory Access	Description
3:0	TOP	* Varies	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) **Values:** ■ 0x0 (USER0_OR_64K): Time out of WDT_USER_TOP_0 or 64K Clocks ■ 0x1 (USER1_OR_128K): Time out of WDT_USER_TOP_1 or 128K Clocks ■ 0x2 (USER2_OR_256K): Time out of WDT_USER_TOP_2 or 256K Clocks ■ 0x3 (USER3_OR_512K): Time out of WDT_USER_TOP_3 or 512K Clocks ■ 0x4 (USER4_OR_1M): Time out of WDT_USER_TOP_4 or 1M Clocks ■ 0x5 (USER5_OR_2M): Time out of WDT_USER_TOP_5 or 2M Clocks ■ 0x6 (USER6_OR_4M): Time out of WDT_USER_TOP_6 or 4M Clocks ■ 0x7 (USER7_OR_8M): Time out of WDT_USER_TOP_7 or 8M Clocks ■ 0x8 (USER8_OR_16M): Time out of WDT_USER_TOP_8 or 16M Clocks ■ 0x9 (USER9_OR_32M): Time out of WDT_USER_TOP_9 or 32M Clocks ■ 0xa (USER10_OR_64M): Time out of WDT_USER_TOP_10 or 64M Clocks

Table 5-7 Fields for Register: WDT_TORR (Continued)

Bits	Name	Memory Access	Description
			■ 0xb (USER11_OR_128M): Time out of WDT_USER_TOP_11 or 128M Clocks ■ 0xc (USER12_OR_256M): Time out of WDT_USER_TOP_12 or 256M Clocks ■ 0xd (USER13_OR_512M): Time out of WDT_USER_TOP_13 or 512M Clocks ■ 0xe (USER14_OR_1G): Time out of WDT_USER_TOP_14 or 1G Clocks ■ 0xf (USER15_OR_2G): Time out of WDT_USER_TOP_15 or 2G Clocks **Value After Reset:** WDT_DFLT_TOP **Exists:** Always **Memory Access:** {(WDT_HC_TOP == 1) ? "read-only" : "read-write"}

5.1.3 WDT_CCVR

- **Name:** Current Counter Value Register
- **Description:** Current Counter Value Register.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

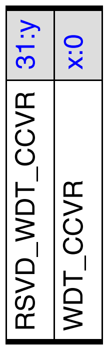


Table 5-8 Fields for Register: WDT_CCVR

Bits	Name	Memory Access	Description
31:y	RSVD_WDT_CCVR	R	WDT_CCVR 31 to WDT_CNT_WIDTH. Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** WDT_CNT_WIDTH
x:0	WDT_CCVR	R	WDT Current Counter Value Register. 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. **Value After Reset:** WDT_CNT_RST **Exists:** Always **Volatile:** true **Range Variable[x]:** WDT_CNT_WIDTH - 1

5.1.4 WDT_CRR

- **Name:** Counter Restart Register
- **Description:** Counter Restart Register
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** Always

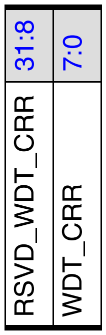


Table 5-9 Fields for Register: WDT_CRR

Bits	Name	Memory Access	Description
31:8	RSVD_WDT_CRR	W	WDT_CRR[31:24] Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always
7:0	WDT_CRR	W	Counter Restart Register. 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. **Values:** ■ 0x76 (RESTART): Watchdog timer restart command **Value After Reset:** 0x0 **Exists:** Always

5.1.5 WDT_STAT

- **Name:** Interrupt Status Register
- **Description:** Interrupt Status Register
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always

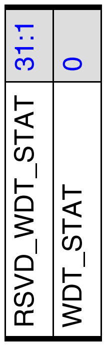


Table 5-10 Fields for Register: WDT_STAT

Bits	Name	Memory Access	Description
31:1	RSVD_WDT_STAT	R	WDT_STAT[31:1] Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	WDT_STAT	R	Interrupt status register This register shows the interrupt status of the WDT. **Values:** ■ 0x0 (INACTIVE): Interrupt is inactive ■ 0x1 (ACTIVE): Interrupt is active regardless of polarity **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.6 WDT_EOI

- **Name:** Interrupt Clear Register
- **Description:** Interrupt Clear Register
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** Always

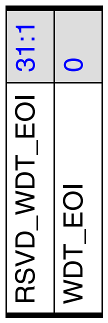


Table 5-11 Fields for Register: WDT_EOI

Bits	Name	Memory Access	Description
31:1	RSVD_WDT_EOI	R	RSVD_WDT_EOI[31:1] Reserved bits and read as zero (0). **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	WDT_EOI	R	Interrupt Clear Register. Clears the watchdog interrupt. This can be used to clear the interrupt without restarting the watchdog counter. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.7 WDT_PROT_LEVEL

- **Name:** WDT Protection level
- **Description:** WDT Protection level register
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** (SLAVE_INTERFACE_TYPE > 1 && SLVERR_RESP_EN==1 && HC_PROT_LEVEL==0) ? 1 : 0

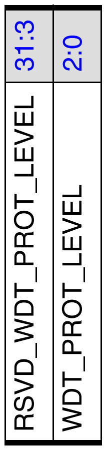


Table 5-12 Fields for Register: WDT_PROT_LEVEL

Bits	Name	Memory Access	Description
31:3	RSVD_WDT_PROT_LEVEL	R	WDT_PROT_LEVEL[31:3] Reserved field- read-only **Value After Reset:** 0x0 **Exists:** Always
2:0	WDT_PROT_LEVEL	* Varies	Protection level register. Enabling protection on any of its three bits would require a match on the input PPROT signal to gain access to protected registers of the WDT. **Value After Reset:** PROT_LEVEL_RST **Exists:** Always **Memory Access:** {(HC_PROT_LEVEL == 1) ? "read-only" : "read-write"}

5.1.8 WDT_COMP_PARAM_5

- **Name:** Component Parameters Register 5
- **Description:** Component Parameters Register 5

  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).
- **Size:** 32 bits
- **Offset:** 0xe4
- **Exists:** Always

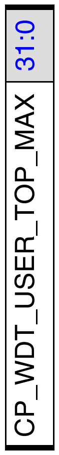


Table 5-13 Fields for Register: WDT_COMP_PARAM_5

Bits	Name	Memory Access	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. **Value After Reset:** WDT_USER_TOP_MAX **Exists:** Always

5.1.9 WDT_COMP_PARAM_4

- **Name:** Component Parameters Register 4
- **Description:** Component Parameters Register 4

  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).
- **Size:** 32 bits
- **Offset:** 0xe8
- **Exists:** Always

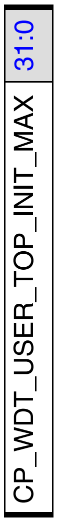


Table 5-14 Fields for Register: WDT_COMP_PARAM_4

Bits	Name	Memory Access	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. **Value After Reset:** WDT_USER_TOP_INIT_MAX **Exists:** Always

5.1.10 WDT_COMP_PARAM_3

- **Name:** Component Parameters Register 3
- **Description:** Component Parameters Register 3

  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).
- **Size:** 32 bits
- **Offset:** 0xec
- **Exists:** Always

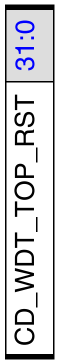


Table 5-15 Fields for Register: WDT_COMP_PARAM_3

Bits	Name	Memory Access	Description
31:0	CD_WDT_TOP_RST	R	The value of this register is derived from the WDT_TOP_RST coreConsultant parameter. **Value After Reset:** WDT_TOP_RST **Exists:** Always

5.1.11 WDT_COMP_PARAM_2

- **Name:** Component Parameters Register 2
- **Description:** Component Parameters Register 2

  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).
- **Size:** 32 bits
- **Offset:** 0xf0
- **Exists:** Always

31:0
CP_WDT_CNT_RST

Table 5-16 Fields for Register: WDT_COMP_PARAM_2

Bits	Name	Memory Access	Description
31:0	CP_WDT_CNT_RST	R	The value of this register is derived from the WDT_RST_CNT coreConsultant parameter. **Value After Reset:** WDT_CNT_RST **Exists:** Always

5.1.12 WDT_COMP_PARAM_1

- **Name:** Component Parameters Register 1
- **Description:** Component Parameters Register 1

  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).
- **Size:** 32 bits
- **Offset:** 0xf4
- **Exists:** Always

31:29	28:24	23:20	19:16	15:13	12:10	9:8	7	6	5	4	3	2	1	0
RSVD_31_29	WDT_CNT_WIDTH	WDT_DFLT_TOP_INIT	WDT_DFLT_TOP	RSVD_15_13	WDT_DFLT_RPL	APB_DATA_WIDTH	WDT_PAUSE	WDT_USE_FIX_TOP	WDT_HC_TOP	WDT_HC_RPL	WDT_HC_RMOD	WDT_DUAL_TOP	WDT_DFLT_RMOD	WDT_ALWAYS_EN

Table 5-17 Fields for Register: WDT_COMP_PARAM_1

Bits	Name	Memory Access	Description
31:29	RSVD_31_29	R	WDT_COMP_PARAM_1[31:29]Reserved bits and read as zero (0). **Value After Reset:** 0x0 **Exists:** Always
28:24	WDT_CNT_WIDTH	R	The Watchdog Timer counter width. **Value After Reset:** "(WDT_CNT_WIDTH-16)" **Exists:** Always
23:20	WDT_DFLT_TOP_INIT	R	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. **Value After Reset:** WDT_DFLT_TOP_INIT **Exists:** Always

Table 5-17 Fields for Register: WDT_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
19:16	WDT_DFLT_TOP	R	Selects the timeout period that is available directly after reset. It controls the reset value of the register. If WDT_HC_TOP is set to 1, then the default timeout period is the only possible timeout period. Can choose one of 16 values. **Value After Reset:** WDT_DFLT_TOP **Exists:** Always
15:13	RSVD_15_13	R	WDT_COMP_PARAM_1[15:13] Reserved bits and read as zero (0). **Value After Reset:** 0x0 **Exists:** Always
12:10	WDT_DFLT_RPL	R	The reset pulse length that is available directly after reset. **Value After Reset:** WDT_DFLT_RPL **Exists:** Always
9:8	APB_DATA_WIDTH	R	Width of the APB Data Bus to which this component is attached. **Values:** ■ 0x0 (APB_8BITS): APB data width is 8 bits ■ 0x1 (APB_16BITS): APB data width is 16 bits ■ 0x2 (APB_32BITS): APB data width is 32 bits **Value After Reset:** "((APB_DATA_WIDTH==8) ? 0 : ((APB_DATA_WIDTH==16) ? 1 :2))" **Exists:** Always
7	WDT_PAUSE	R	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. **Values:** ■ 0x0 (DISABLED): Pause enable signal is non existent ■ 0x1 (ENABLED): Pause enable signal is included **Value After Reset:** WDT_PAUSE **Exists:** Always

Table 5-17 Fields for Register: WDT_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
6	WDT_USE_FIX_TOP	R	When this parameter is set to 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 parameter is set to 0, the user must define the timeout period range (2^8 to 2^(WDT_CNT_WIDTH)-1) using the WDT_USER_TOP_(i) parameter. **Values:** ■ 0x0 (USERDEFINED): User must define timeout values ■ 0x1 (PREDEFINED): Use predefined timeout values **Value After Reset:** WDT_USE_FIX_TOP **Exists:** Always
5	WDT_HC_TOP	R	When set to 1, the selected timeout period(s) is set to be hard coded. **Values:** ■ 0x0 (PROGRAMMABLE): Timeout period is programmable ■ 0x1 (HARDCODED): Timeout period is hard coded **Value After Reset:** WDT_HC_TOP **Exists:** Always
4	WDT_HC_RPL	R	Configures the reset pulse length to be hard coded. **Values:** ■ 0x0 (PROGRAMMABLE): Reset pulse length is programmable ■ 0x1 (HARDCODED): Reset pulse length is hardcoded **Value After Reset:** WDT_HC_RPL **Exists:** Always
3	WDT_HC_RMOD	R	Configures the output response mode to be hard coded. **Values:** ■ 0x0 (PROGRAMMABLE): Output response mode is programmable ■ 0x1 (HARDCODED): Output response mode is hard coded **Value After Reset:** WDT_HC_RMOD **Exists:** Always

Table 5-17 Fields for Register: WDT_COMP_PARAM_1 (Continued)

Bits	Name	Memory Access	Description
2	WDT_DUAL_TOP	R	When set to 1, includes a second timeout period that is used for initialization prior to the first kick. **Values:** ■ 0x0 (DISABLED): Second timeout period is not present ■ 0x1 (ENABLED): Second timeout period is present **Value After Reset:** WDT_DUAL_TOP **Exists:** Always
1	WDT_DFLT_RMOD	R	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 equals 0 and an interrupt followed by a system reset, if equals 1. If WDT_HC_RMOD is 1, then default response mode is the only possible output response mode. **Values:** ■ 0x0 (DISABLED): System reset only ■ 0x1 (ENABLED): Interrupt and system reset **Value After Reset:** WDT_DFLT_RMOD **Exists:** Always
0	WDT_ALWAYS_EN	R	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. **Values:** ■ 0x0 (DISABLED): Watchdog timer disabled on reset ■ 0x1 (ENABLED): Watchdog timer enabled on reset **Value After Reset:** WDT_ALWAYS_EN **Exists:** Always

5.1.13 WDT_COMP_VERSION

- **Name:** Component Version Register
- **Description:** Component Version Register
- **Size:** 32 bits
- **Offset:** 0xf8
- **Exists:** Always

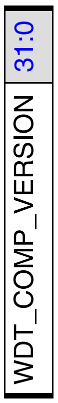


Table 5-18 Fields for Register: WDT_COMP_VERSION

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

5.1.14 WDT_COMP_TYPE

- **Name:** Component Type Register
- **Description:** Component Type Register
- **Size:** 32 bits
- **Offset:** 0xfc
- **Exists:** Always

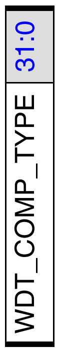


Table 5-19 Fields for Register: WDT_COMP_TYPE

Bits	Name	Memory Access	Description
31:0	WDT_COMP_TYPE	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. **Value After Reset:** WDT_PERIPHERAL_ID **Exists:** Always

6 Programming the DW_apb_wdt

This chapter describes the programmable features of the DW_apb_wdt.

6.1 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.

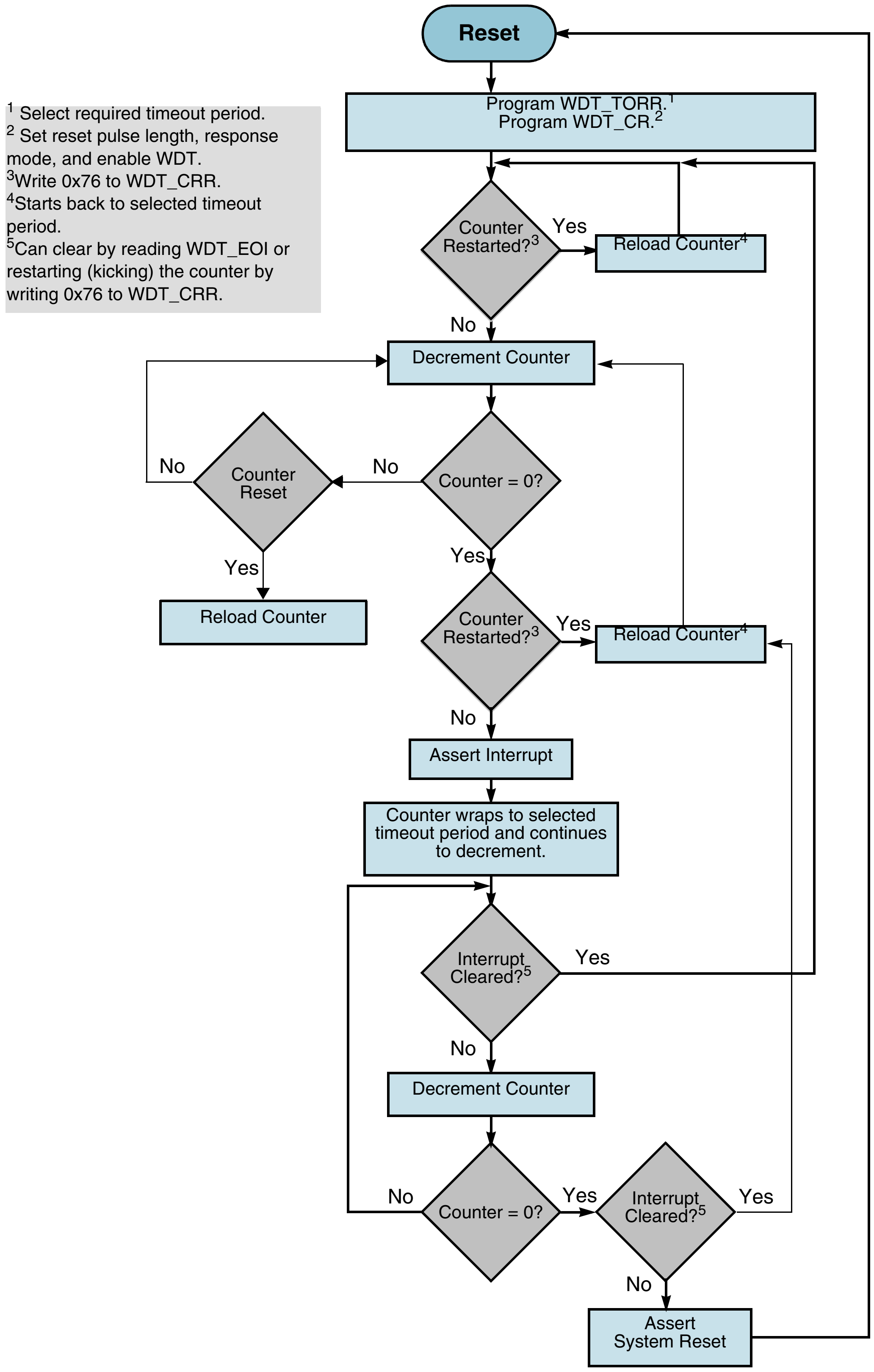
6.2 Example Operational Flow of DW_apb_wdt

Figure 6-1 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 “Parameter Descriptions” on page 31.

Figure 6-1 Operation Flow of an Example DW_apb_wdt Configuration



[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.

7 Verification

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

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

Note The packaged test benches are only for validating the IP configuration in coreConsultant GUI. It is not for system level validation.

IPs that have the Vera test bench packaged, these test benches are encrypted.

This chapter discusses the following sections:

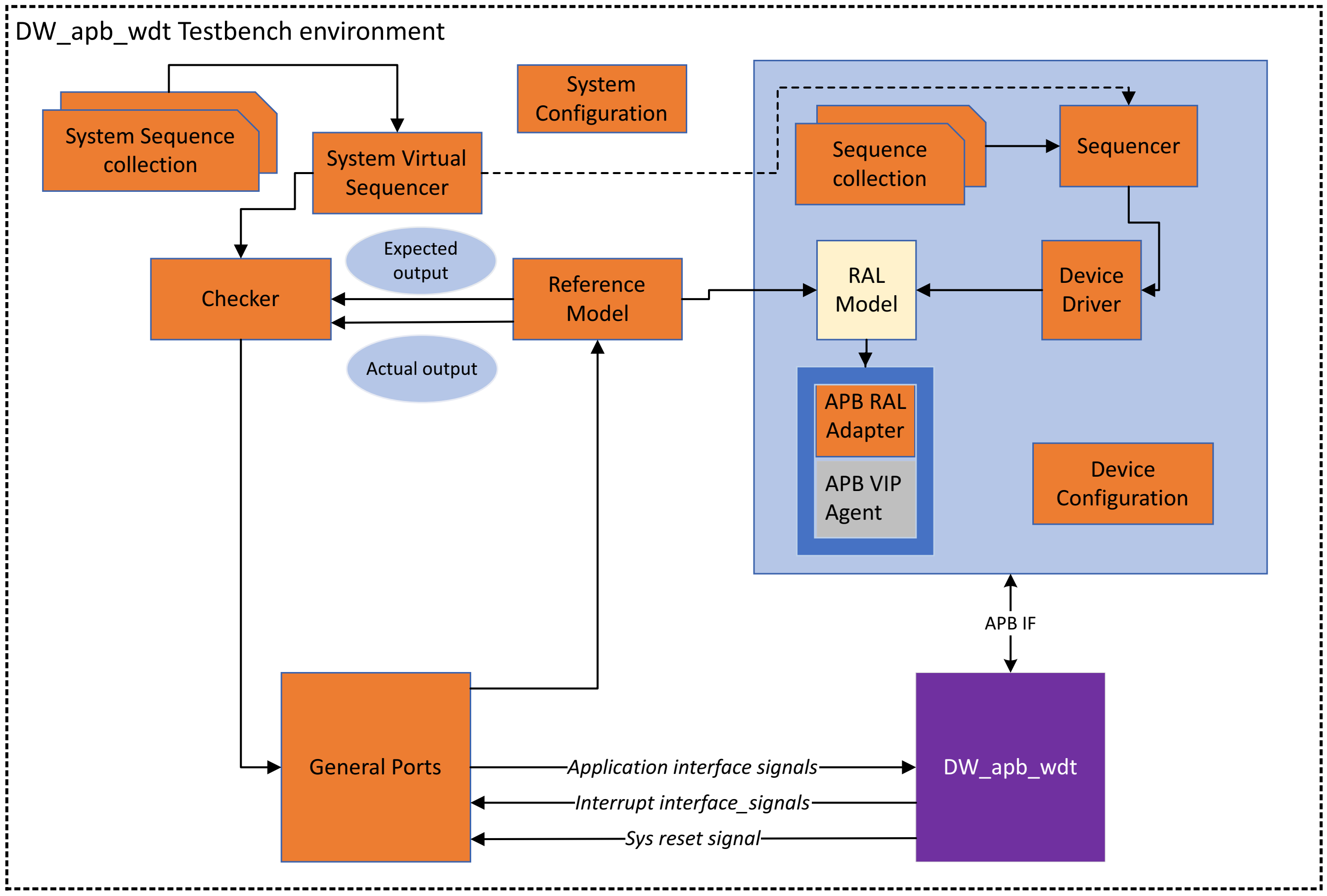
- "Verification Environment" on page 82
- "Testbench Directories and Files" on page 84
- "Packaged Testcases" on page 84

7.1 Verification Environment

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

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

Figure 7-1 DW_apb_wdt UVM Verification Environment



The testbench consists of the following elements:

- Testbench uses the standard SVT VIP for the protocol interfaces:
  - APB SVT VIP Q-2020.03
  - APB VIP Interface for connecting master and slave ports of DUT.
  - APB VIP system environment (svt_apb_system_env):

    APB master agent and APB slave agent for providing the VIP supported randomized transactions with bus instantiation on either side of the DUT.
- Testbench uses the custom components:

- Device Agent

  This component is responsible for creating traffic on the application bus interface (APB). This agent follows the standard structure of a UVM agent and has the following sub-blocks.

  - Device Sequencer

    This component is a standard UVM sequencer, which fetches the top level sequence items from the scenario sequences and feeds the device driver. There is no additional logic present in the device sequencer.
  - Device Driver

    The core logic of device agent sits in the device driver. This block is responsible for the APB side read and write. The bus protocol driver along with the RAL forms the lower layer, which directly interacts with the design bus interface.

    On the Write and Read paths, the device driver fetches a protocol transaction from the scenario sequences (via device sequencer) and initiates the respective APB register writes or reads.

    The device driver is planned to be independent of the application bus interface. This is accomplished by using RAL. In case of a change of bus protocol only change is going to be in the RAL adapter logic.
- General Ports

  The General Ports module is responsible to Read and Write the DUTs signals as follows:

  - Application Interface Signals:

    Write to wdt_clk_en, pause, speed_up, scan_mode, tclk, tresetn inputs
  - Interrupt Interface Signals:

    Read from wdt_intr, wdt_intr_n output
  - System Reset Interface Signals:

    Read from wdt_sys_rst, wdt_sys_rst_n output

  The DUT has two possible modes that are being set using the WDT_ASYNC_CLK_MODE_ENABLE parameter. When this parameter is true, the DUT operates the counter with a clock that is asynchronous from the peripheral clock.

  - Mode 1: One clock domain PCLK, register access is performed using PCLK and also the counter is decremented on PCLK. Interrupt and reset signal are also set and cleared on PCLK.
  - Mode 2: Two clock domains PCLK and TCLK, register access is performed using PCLK, and the counter is decremented on TCLK. Interrupt signal and reset signal are set on PCLK, and cleared on TCLK.
- Reference Model / Checker

  The Reference Model is responsible for updating the RAL content for the activity done outside of main register interface.

  Based on the transactions received from the agent monitors, the Reference Model also calculates the expected values of DUT output signals. The actual DUT outputs, as well as the expected values are then sent to the Checker. In the Checker, SystemVerilog assertions are developed, comparing the expected values of DUT output signals against the values actually driven by DUT.

7.2 Testbench Directories and Files

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

Table 7-2 shows the various directories and associated files:

Table 7-1 DW_apb_wdt Testbench Directory Structure

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

7.3 Packaged Testcases

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

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

Table 7-2 DW_apb_wdt Test Description

Test Name	Test Description
test_100_reg_reset_bit_bash	This test builds the sequence `dw_apb_wdt_reg_reset_bit_bash_virtual_sequence`, which in turn initiates `dw_apb_wdt_reg_reset_bit_bash_sequence`. This test performs: ■ Validation of register values after reset. ■ Bit bash for the bits of all the registers.

Table 7-2 DW_apb_wdt Test Description (Continued)

Test Name	Test Description
test_101_random_wdt	This test initiates the random testing of all the other test sequences. It performs the following validations: 1. This test builds `dw_apb_wdt_enable_virtual_sequence`. - This functionality is checked only when WDT_ALWAYS_EN = 0, else the WDT is always enabled. - This test manually enables the WDT and checks the counter value by reading WDT_CCVR. 2. This test builds the `dw_apb_wdt_pause_mode_virtual_sequence`. - When WDT_USE_PAUSE is enabled, the WDT VIP transactions are made with enabling and disabling the pause mode simultaneously the DUT functionality is validated. 3. This test builds the `dw_apb_wdt_speed_up_counter_virtual_sequence`. - Testing the Watchdog speed_up operation. - WDT transactions by enabling and disabling the speed_up variable are constructed. - Intermediately the WDT_CRR register is set and the functionality of speed up operation is validated.
test_103_restart_wdt_counter_while_zero	This test builds the sequence `dw_apb_wdt_restart_counter_while_zero_virtual_sequence`. ■ Testing the watchdog restart operation while decremented to zero. ■ IF WDT_HC_RMOD = 0, then set the RMOD = 1. ■ If not a free running counter then enable it. ■ While monitoring the counter, when it reaches zero, check the interrupt from the WDT_STAT register.

8 Integration Considerations

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

8.1 Reading and Writing from an APB Slave

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

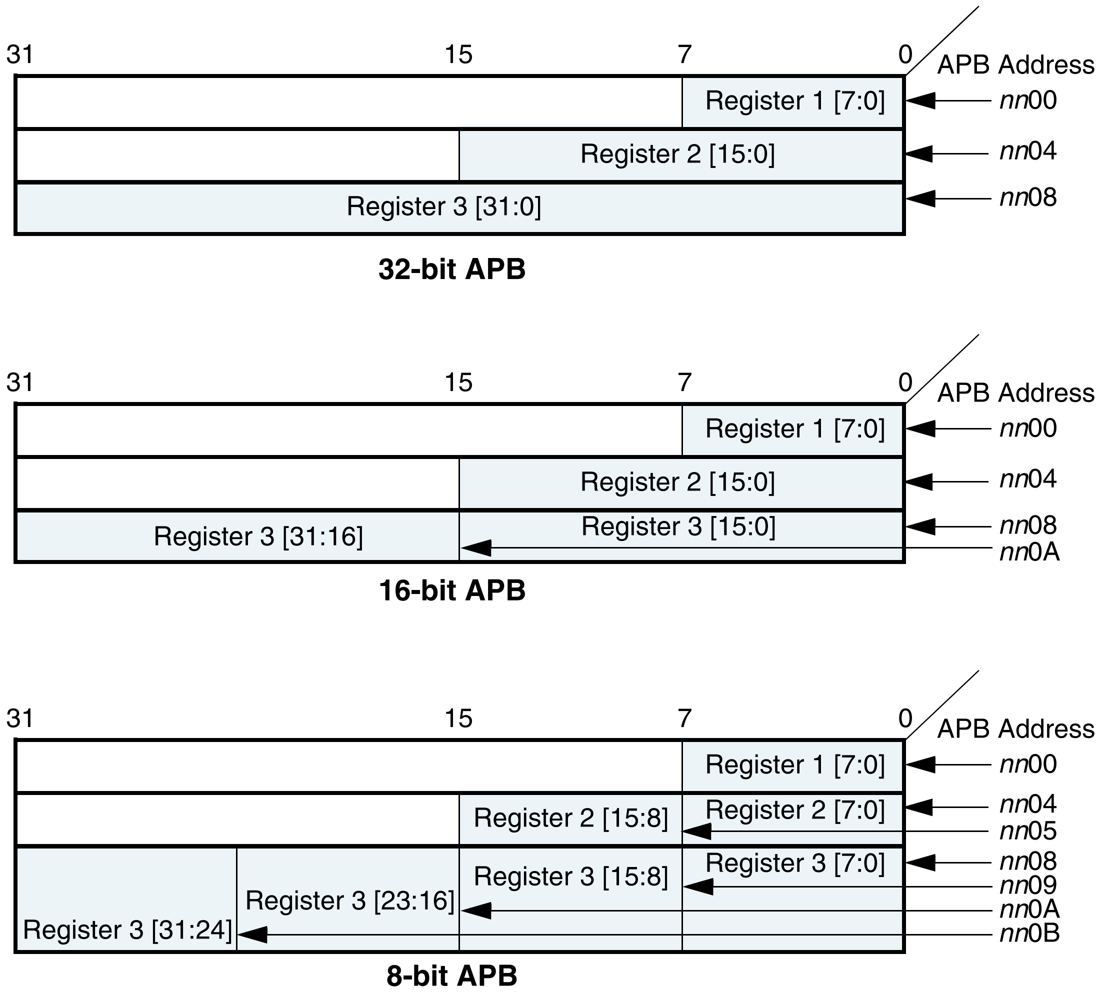
- The size of the APB peripheral should always be set equal to the size of the APB data bus, if possible.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian 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.

8.1.1 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 8-1 Read/Write Locations for Different APB Bus Data Widths



8.1.2 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.

8.1.3 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.

8.1.4 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

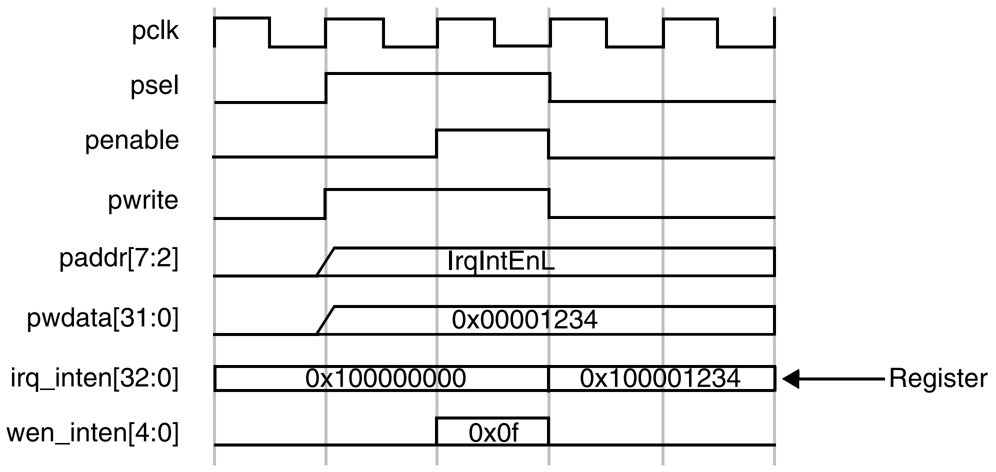
   In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

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

8.2 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 8-2 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.

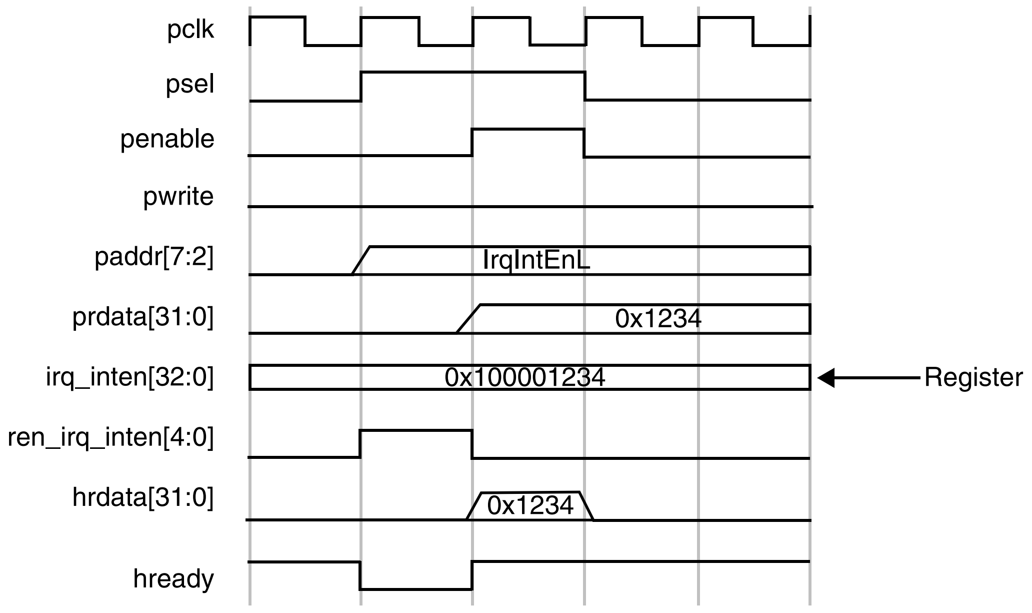
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

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.

8.3 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 8-3 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.

8.4 Addressing

There is mixture of 32 and 64-bit registers in the AHB slave interface. All the registers are 64 bits wide, except the fiq registers, which are 32 bits wide. A register may not be accessed with a hsize greater than its width. Notwithstanding this hsize restriction, all other types of hsize accesses are supported as per the AMBA specification.

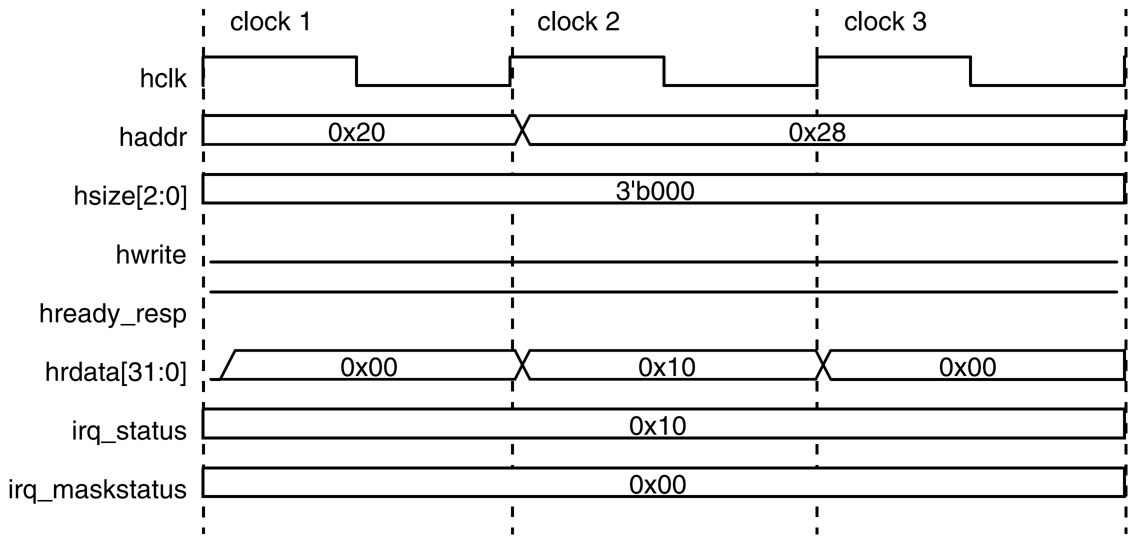
When performing a transfer on the AHB bus, the address must be aligned with the value of hsize. For example, if hsize = 32, then the address must be aligned to 32 bits, such as the two least significant bits = 'b0.

This condition is not checked in the slave interface but is a requirement of the AMBA specification. Also, hsize must not specify a transfer width greater than the slave's AHB data width; this is also a protocol violation.

8.5 Read Accesses

For reads, registers less than the full access width return zeros in the unused upper bits. An AHB read takes two hclk cycles. The two cycles can be thought of as a control and data cycle, respectively. As shown in the following figure, the address and control is driven from clock 1 (control cycle); the read data for this access is driven by the slave interface onto the bus from clock 2 (data cycle) and is sampled by the master on clock 3. The operation of the AHB bus is pipelined, so while the read data from the first access is present on the bus for the master to sample, the control for the next access is present on the bus for the slave to sample.

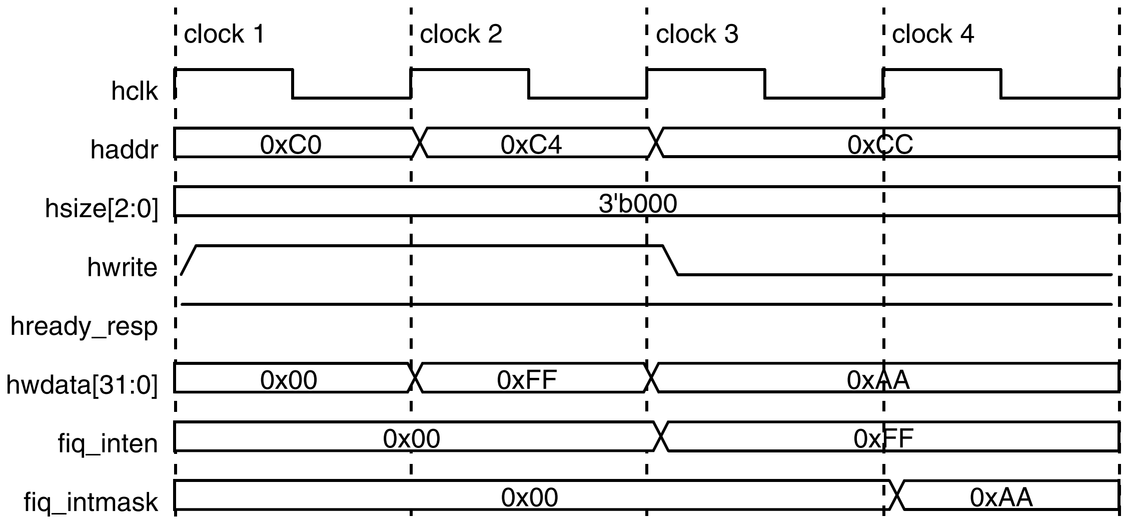
Figure 8-4 AHB Read



8.6 Write Accesses

When writing to a register, bit locations larger than the register width or allocation are ignored. Only pertinent bits are written to the register. Similar to the read case, a write access may be thought of as comprising a control and data cycle. As illustrated in the following figure, the address and control is driven from clock1 (control cycle), and the write data is driven by the bus from clock 2 (data cycle) and sampled by the destination register on clock 3.

Figure 8-5 AHB Write

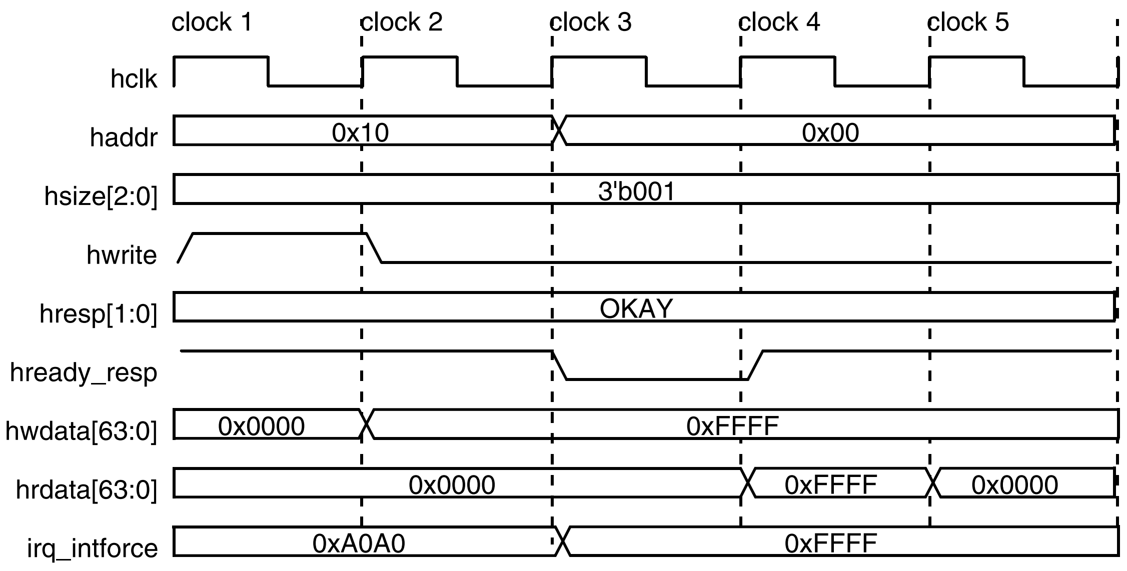


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

8.7 Consecutive Write-Read

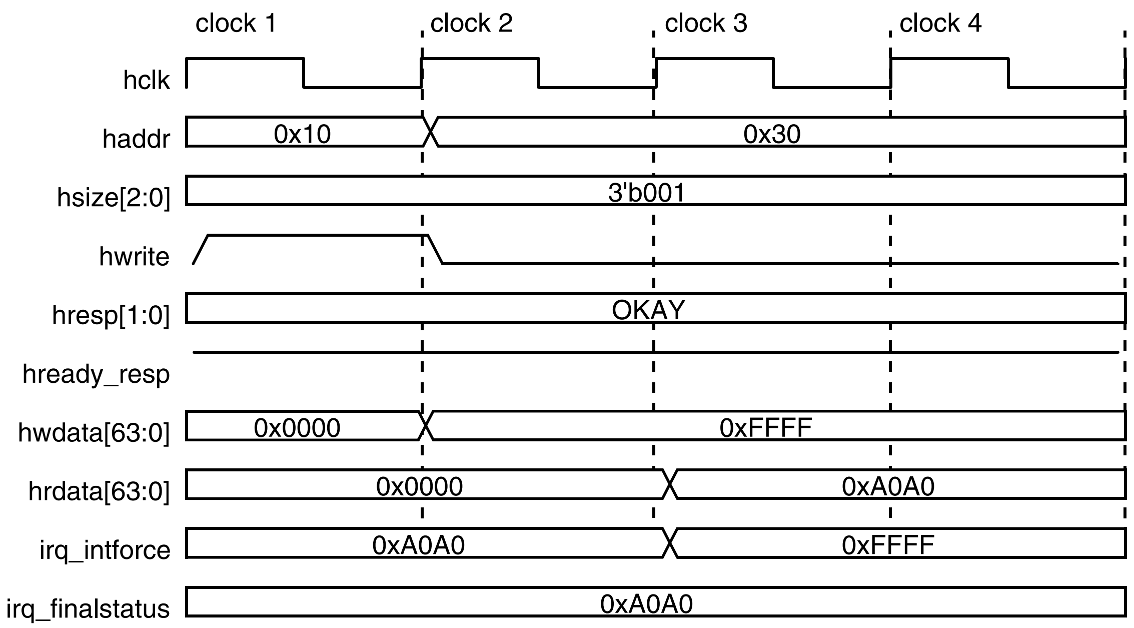
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_ictl inserts a "wait state" when it detects a read immediately after a write to the same address. As shown in the following figure, the control for a write is driven on clock 1, followed by the write data and the control for a read from the same address on clock 2.

Figure 8-6 AHB Wait State Read/Write



Sensing the read after a write to the same address, the slave interface drives hready_resp low from clock 3; hready_resp is driven high on clock 4 when the new write data can be read; and the bus samples hready_resp high on clock 5 and reads the newly written data. The following figure shows a normal consecutive write-read access.

Figure 8-7 AHB Consecutive Read/Write



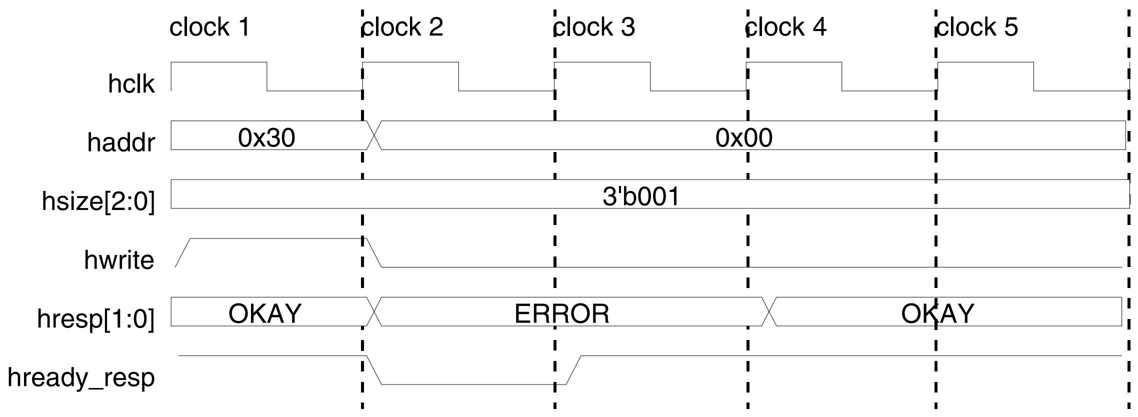
8.8 Error Responses

The following are the cases where the slave interface of the DW_ahb_ictl issues an ERROR response.

- Accessing the slave with an address that does not decode to a register in the design. (The DW_ahb_ictl slave interface decodes for a 1 KB address space [bits 0 to 9 of haddr]. If an address inside this boundary does not refer to a register in the design, an ERROR is given.)
- Attempting to write to a read only register location.
- Accesses to a 64 bit register location with hsize greater than 64.
- Accesses to a 32 bit register location with hsize greater than 32.
- Accesses to the irq_vector register with hsize < 32. We impose this restriction to remove the need for shadow registers to guarantee coherency of irq_vector for reads with a hsize of less than 32 bits.

The following figure shows an error response. The control for the errant access is driven from clock 1. In this case, you are attempting to write to the irq_finalstatus register, which is read only. From clock 2, hready_resp is driven low, and hresp is driven to ERROR. From clock 3, hready_resp is driven to high and hresp remains at ERROR. From clock 4, hresp is driven to OKAY again and the ERROR response is complete.

Figure 8-8 AHB Error Response



8.9 Accessing Top-level Constraints

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

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

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

```
autocomplete_activity Synthesize
```

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

```
write_sdc <filename>
```

8.10 Coherency

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

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

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

Throughout this appendix the following terms are used:

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

8.10.1 Writing Coherently

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

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

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

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

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

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

Table 8-1 Upper Byte Generation

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

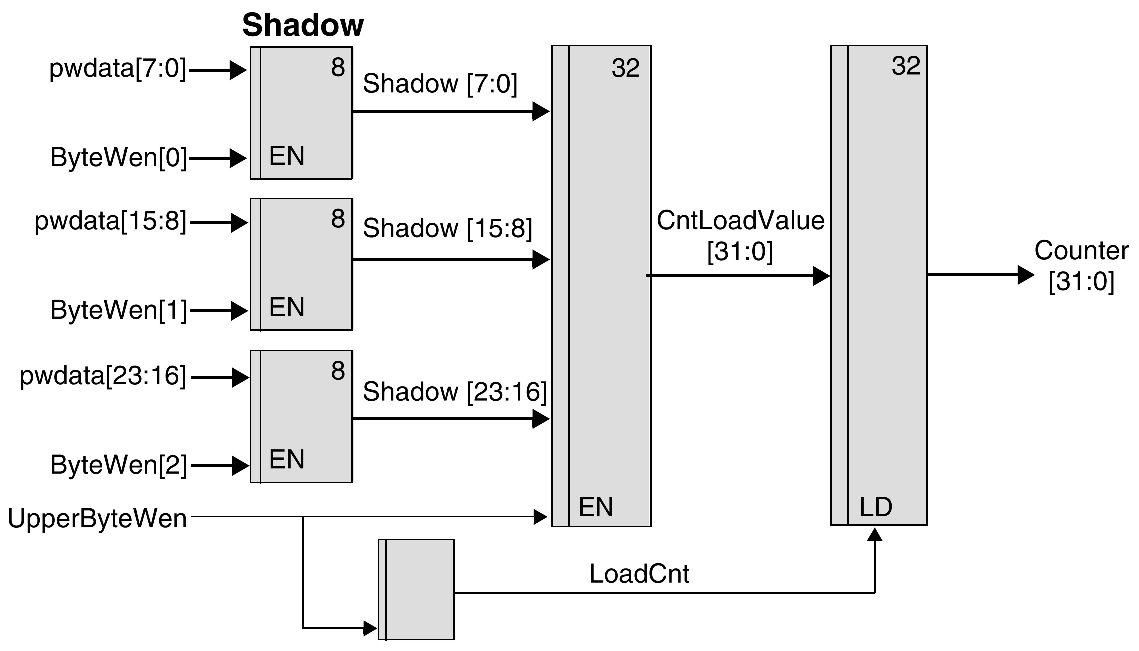
There are three relationship cases to be considered for the processor and peripheral clocks:

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

8.10.1.1 Identical Clocks

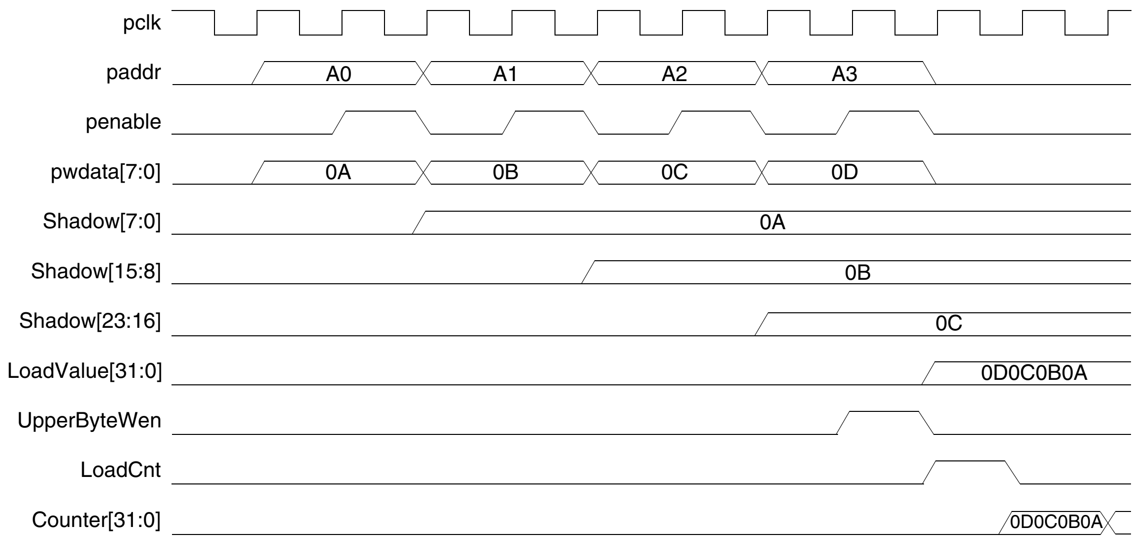
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 8-9 Coherent Loading – Identical Synchronous Clocks



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

Figure 8-10 Coherent Loading – Identical Synchronous Clocks



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

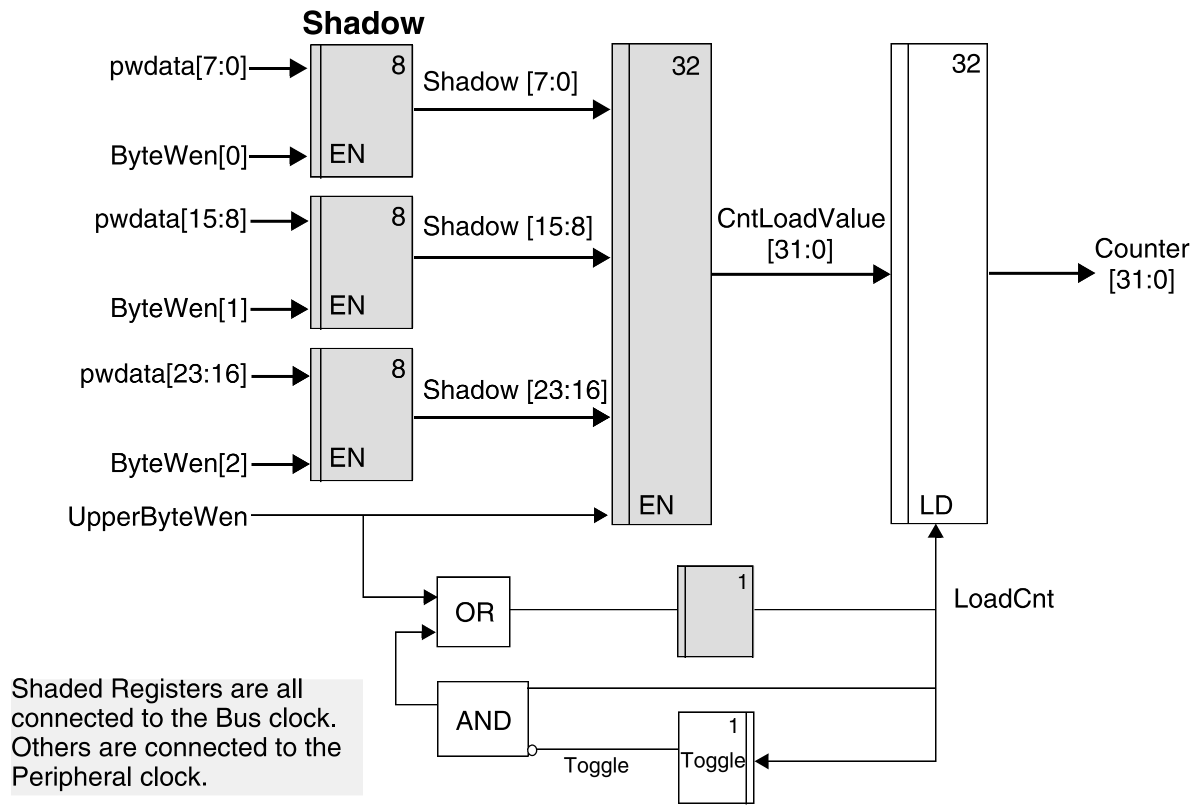
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this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

8.10.1.2 Synchronous Clocks

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

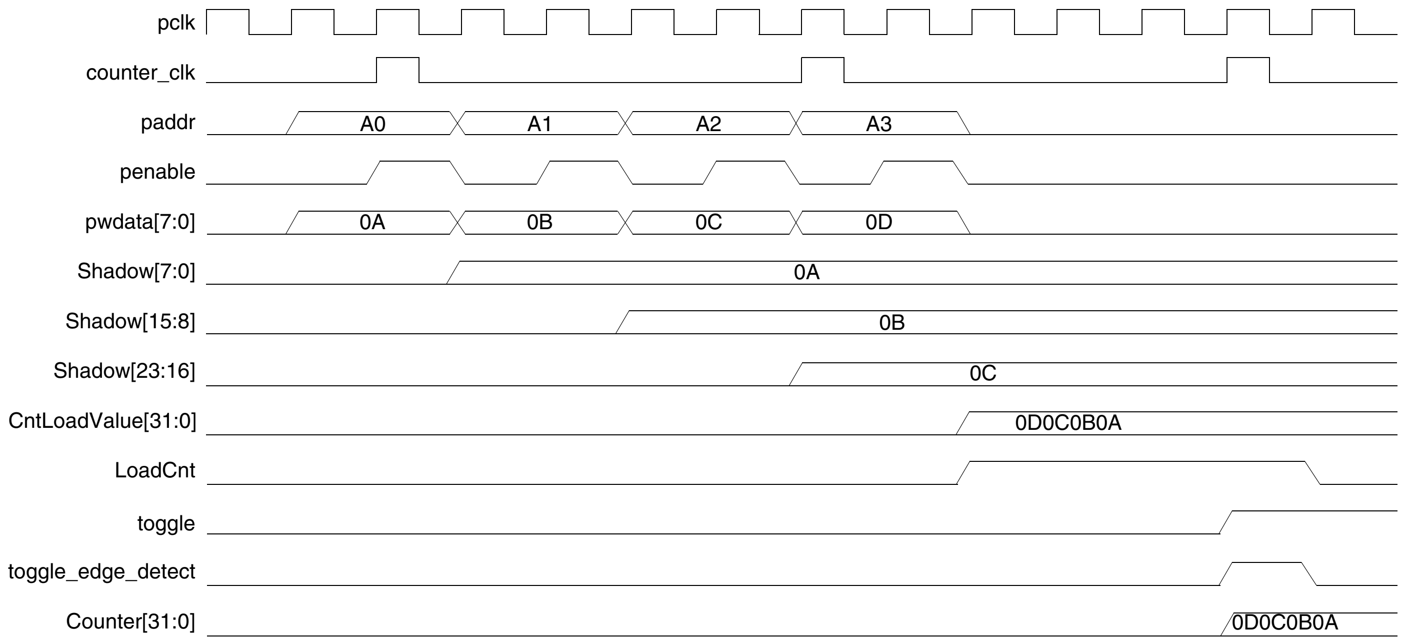
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 8-11 Coherent Loading – Synchronous Clocks



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

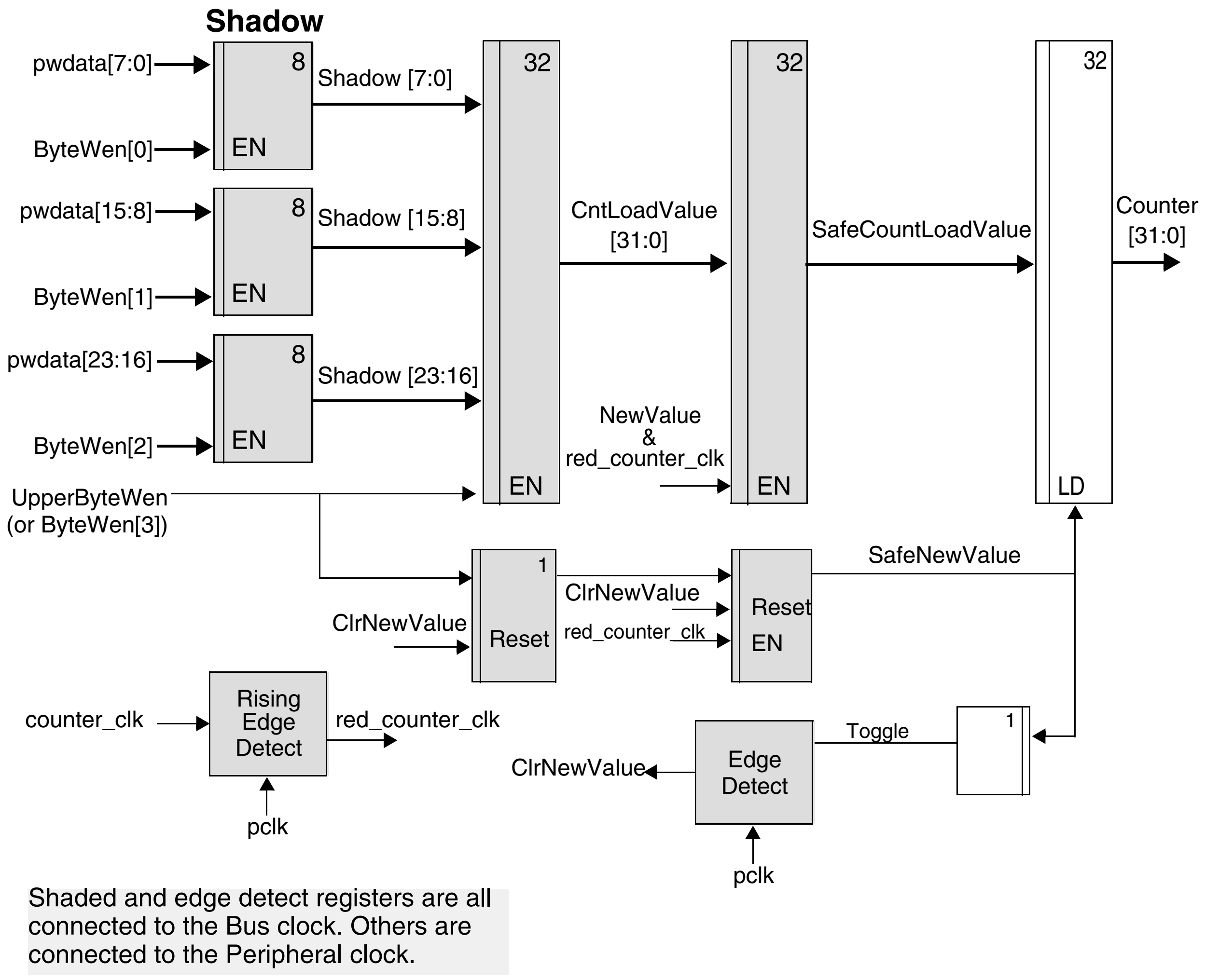
Figure 8-12 Coherent Loading – Synchronous Clocks



8.10.1.3 Asynchronous Clocks

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

Figure 8-13 Coherent Loading – Asynchronous Clocks

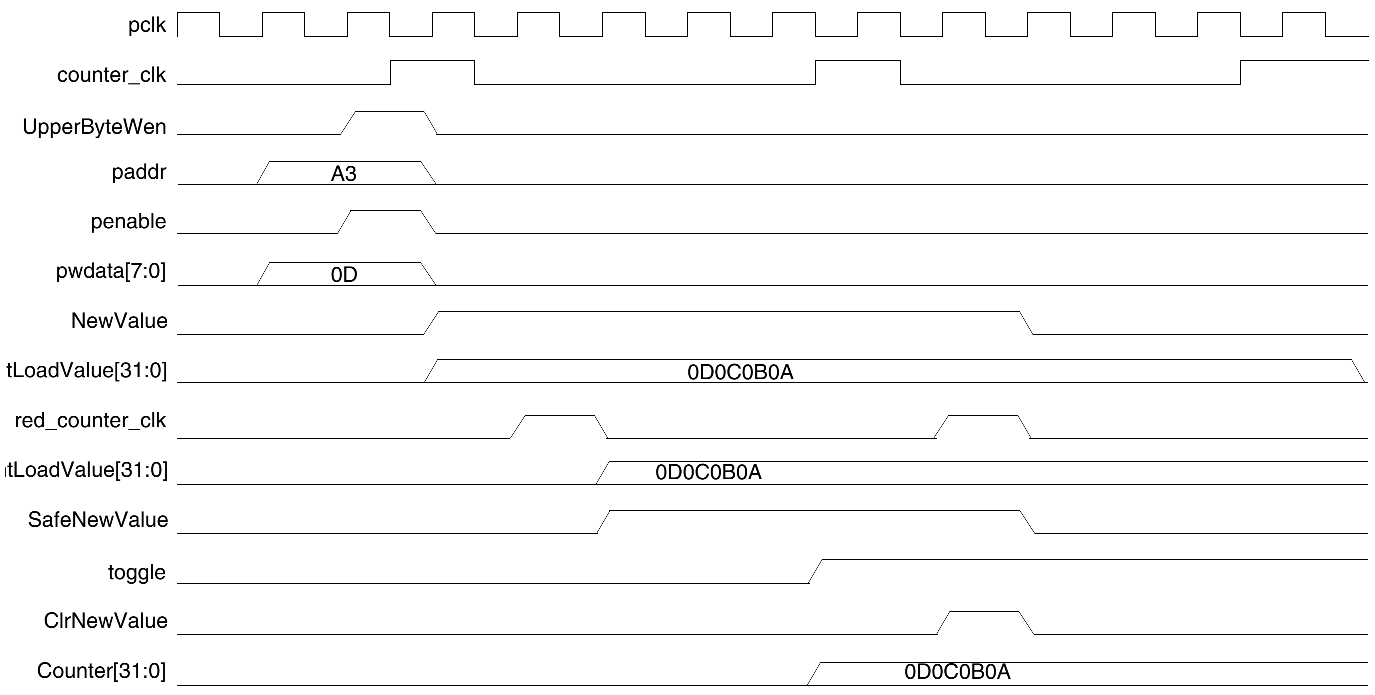


When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock 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 is also a single bit transferred that is used to qualify the loading of the value into the counter.

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

Figure 8-14 Coherent Loading – Asynchronous Clocks



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

8.10.2 Reading Coherently

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

Table 8-2 Lower Byte Generation

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

Table 8-2 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

8.10.2.1 Synchronous Clocks

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

Figure 8-15 Coherent Registering – Synchronous Clocks

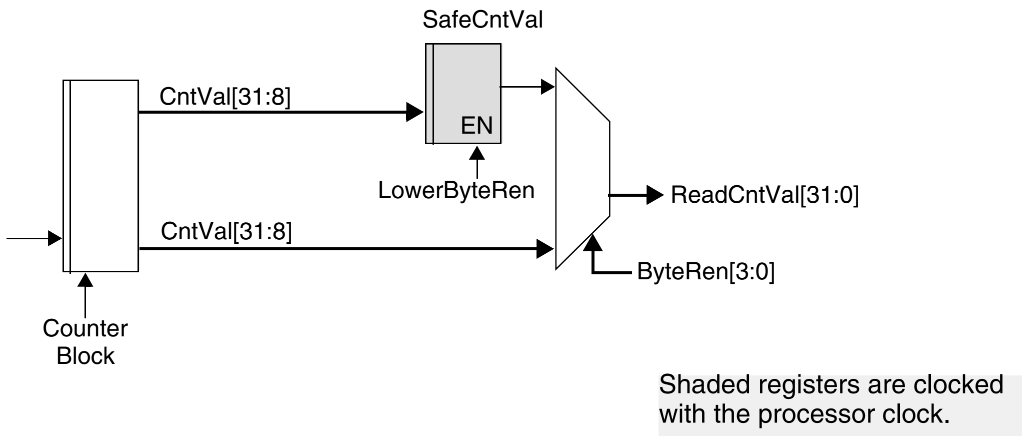
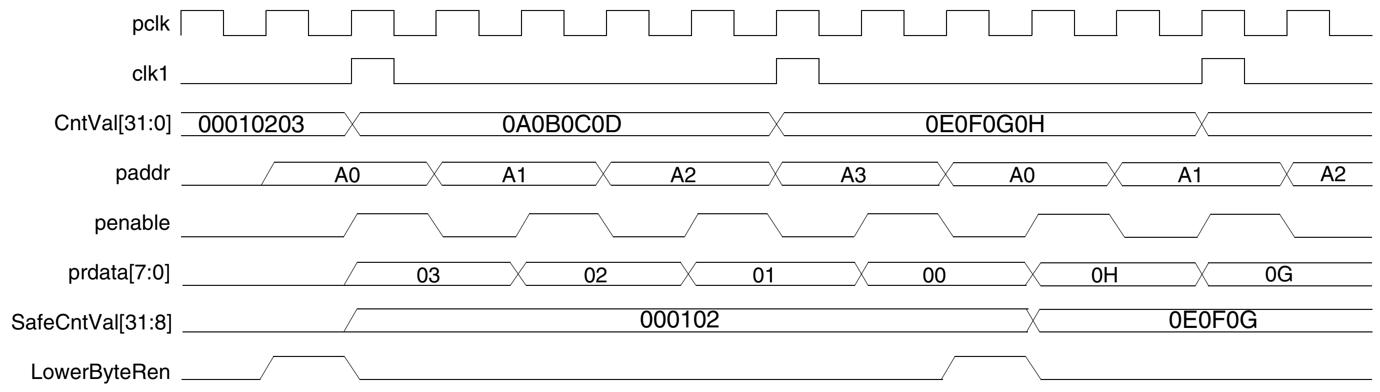


Figure 8-16 Coherent Registering – Synchronous Clocks



8.10.2.2 Asynchronous Clocks

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

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

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

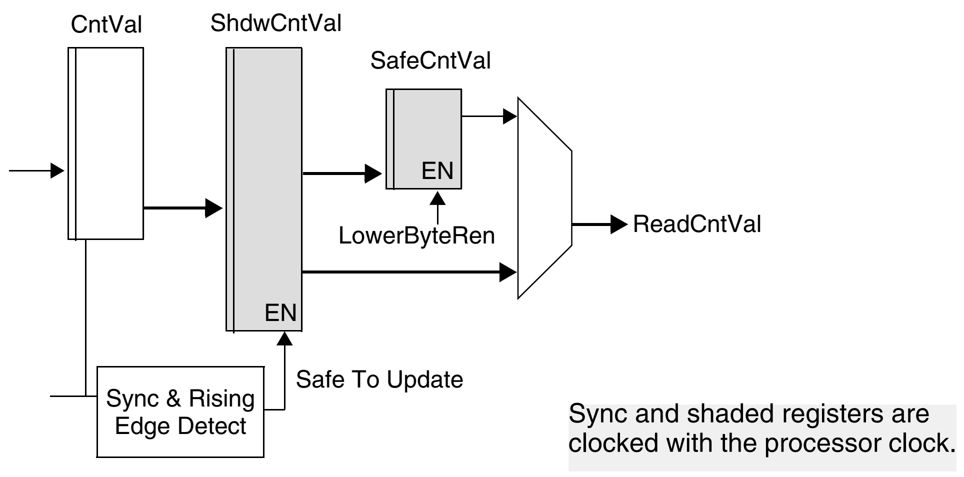
Note You must read LSB to MSB when the bus width is narrower than the counter width.

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

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

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

Figure 8-17 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.

8.11 Timing Exceptions

- For details on multi cycle paths, see the DW_apb_wdt.sdc file generated by the Design Compiler.
- For details on quasi-static signals on the design, refer to manual.sgdc report generated by the SpyGlass tool.

8.12 Performance

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

8.12.1 Power Consumption, Frequency, Area and DFT Coverage

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

Table 8-3 Synthesis Results for DW_apb_wdt

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		Spyglass StuckAtCov (%)
			Static	Dynamic	StuckAtTest	Transition	
Default Configuration	pclk=100 MHz	1045	3 nW	0.005 mW	99.6	98.13	99.9
Typical Configuration - 1 WDT_HC_TOP = 1 WDT_HC_RPL = 1 WDT_HC_RMOD = 1 WDT_ALWAYS_EN = 1	pclk=100 MHz	830	2 nW	0.005 mW	99.6	98.5	99.2
Typical Configuration - 2 APB_DATA_WIDTH = 8 WDT_CNT_WIDTH = 32 WDT_DUAL_TOP = 1	pclk=100 MHz	1158	3 nW	0.006 mW	99.74	98.7	99.9

A
Basic Core Module (BCM) Library

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

This appendix contains the following sections:

- "BCM Library Components" on page 107
- "Synchronizer Methods" on page 107

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_apb_wdt.

Table A-1 BCM Library Components

BCM Module Name	BCM Description	DWBB Equivalent
DW_apb_wdt_bcm21	Single clock data bus synchronizer	DW_sync
DW_apb_wdt_bcm36_nhs	Bus Delay Component	--

A.2 Synchronizer Methods

This section describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_apb_wdt to cross clock boundaries.

This section contains the following sub-sections:

- "Synchronizers Used in DW_apb_wdt" on page 108
- "Synchronizer 1: Simple Double Register Synchronizer (DW_apb_wdt)" on page 108

Note The DesignWare Building Blocks (DWBB) contains several synchronizer components with functionality similar to methods documented in this appendix. For more information about the DWBB synchronizer components go to:
https://www.synopsys.com/dw/buildingblock.php

A.2.1 Synchronizers Used in DW_apb_wdt

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_apb_wdt are listed and cross referenced to the synchronizer type in Table A-2. Note that certain BCM modules are contained in other BCM modules, as they are used as building blocks.

Table A-2 Synchronizers used in DW_apb_wdt

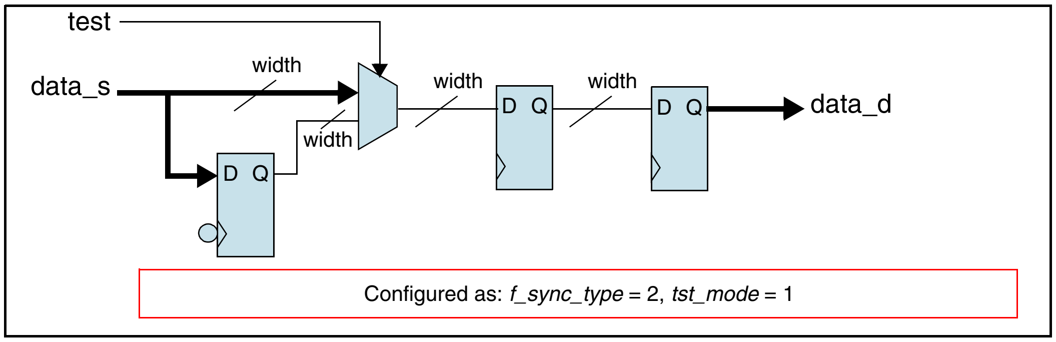
Synchronizer Module File	Synchronizer Type and Number
DW_apb_wdt_bcm21.v	Synchronizer 1: Simple Multiple Register Synchronizer

Note The BCM21 is a basic multiple register based synchronizer module used in the design. It can be replaced with equivalent technology specific synchronizer cell.

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_wdt)

This is a single clock data bus synchronizer for synchronizing control signals that crosses asynchronous clock boundaries. The synchronization scheme uses two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block diagram of Synchronizer 1 with two stage synchronization (both positive edge)



B
Internal Parameter Descriptions

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

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

Table B-1 Internal Parameters

Parameter Name	Equals To
WDT_ADDR_SLICE_LHS	8
WDT_CNT_RST	{[~~function_of:~~::DW_apb_wdt::default_wdt_cnt_rst WDT_DUAL_TOP]}
WDT_PERIPHERAL_ID	32'h44570120
WDT_TOP_RST	=(WDT_DUAL_TOP == 1 ? ((WDT_DFLT_TOP_INIT * 16) + WDT_DFLT_TOP): WDT_DFLT_TOP)
WDT_USER_TOP_INIT_MAX	{[~~function_of:~~::DW_apb_wdt::calc_max_values WDT_USER_TOP_INIT 15]}
WDT_USER_TOP_MAX	{[~~function_of:~~::DW_apb_wdt::calc_max_values WDT_USER_TOP 15]}
WDT_USER_TOP_MAXVALUE	{ [~~function_of:~~::DW_apb_wdt::calc_user_top_limit WDT_CNT_WIDTH] }
WDT_VERSION_ID	32'~~h3131312a~~h3131322a

C Glossary

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

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

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