DesignWare DW_apb_ictl Databook

DW_apb_ictl – Product Code

2.09a
July 2018

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Third-Party Links

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

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

www.synopsys.com

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 2.03b onward.

Version	Date	Description
2.09a	July 2018	**Added:** ■ "Vector Port" on page 22 ■ Appendix A, "Synchronizer Methods" **Updated:** ■ "Performance" on page 108 ■ "Parameter Descriptions" on page 29 and "Register Descriptions" on page 47 "Signal Descriptions" on page 39, "Internal Parameter Descriptions" on page 115 are auto extracted with change bars from the RTL. **Removed:** ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.
2.08a	October 2016	■ Version number change to 2016.10a ■ "Parameter Descriptions" on page 29 and "Register Descriptions" on page 47 auto-extracted from the RTL ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 ■ Replaced Figure 2-2 and Figure 2-3 to remove references to Leda ■ Moved Appendix B, "Internal Parameter Descriptions" to Appendix ■ Added an entry for the xprop directory in Table 2-1 and Table 2-4.
2.07a	June 2015	■ Added "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added "Running SpyGlass on Generated Code with coreAssembler" ■ Chapter 4, "Signal Descriptions" auto-extracted from the RTL ■ Added Chapter B, "Internal Parameter Descriptions"

(Continued)

Version	Date	Description
2.06a	June 2014	■ Version change for 2014.06a release. ■ Added “Performance” section in the “Integration Considerations” chapter ■ Corrected Default Input/Output Delay in Signals chapter
2.05f	May 2013	■ Version change for 2013.05a release ■ Updated the template
2.05e	Oct 2012	Added the product code on the cover and in Table 1-1
2.05e	Mar 2012	Version change for 2012.03a release
2.05d	Nov 2011	Version change for 2011.11a release
2.05c	Oct 2011	Version change for 2011.10a release
2.05b	Jun 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced “Related Documents” section in Preface
2.05b	Apr 2011	Version change for 2011.03a release.
2.05a	Sep 2010	Corrected names of include files and vcs command used for simulation
2.04a	Dec 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks
2.04a	May 2009	Removed references to QuickStarts, as they are no longer supported
2.04a	Oct 2008	Version change for 2007.10a release.
2.03c	Aug 2008	Added note about irq_intpfilt signal and changed names of signals in Figures 6 and 10
2.03c	Jun 2008	Version change for 2008.06a release.
2.03b	Dec 2007	■ Updated for revised installation guide and consolidated release notes titles ■ Changed references of “Designware AMBA” to simply “DesignWare”
2.03b	Jun 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DesignWare APB Interrupt Controller (DW_apb_ictl) component to the Advanced Peripheral Bus (APB). This component conforms to the *AMBA Specification, Revision 2.0* from Arm®.

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

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_apb_ictl.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_apb_ictl.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_apb_ictl signals.
- Chapter 5, "Register Descriptions" describes the programmable registers of the DW_apb_ictl.
- Chapter 6, "Programming the DW_apb_ictl" provides information needed to program the configured DW_apb_ictl.
- Chapter 7, "Verification" provides information on verifying the configured DW_apb_ictl.
- Chapter 8, "Integration Considerations" includes information you need to integrate the configured DW_apb_ictl into your design.
- Appendix B, "Internal Parameter Descriptions" describes the programmable registers of the DW_apb_ictl.
- 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 to the Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI.

Web Resources

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

Customer Support

To obtain support for your product:

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

    File > Build Debug Tar-file

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

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

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

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

    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:

    https://www.synopsys.com/support/global-support-centers.html

Product Code

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

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

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multi-layer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI
DW_axi_a2x	Configurable bridge between AXI and AHB components or AXI and AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI fabric
DW_axi_hmx	Configurable high performance interface from and AHB master to an AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI subsystems
DW_axi_x2h	Bridge from AMBA 3 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI fabric

Component Name	Description
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or busses

1 Product Overview

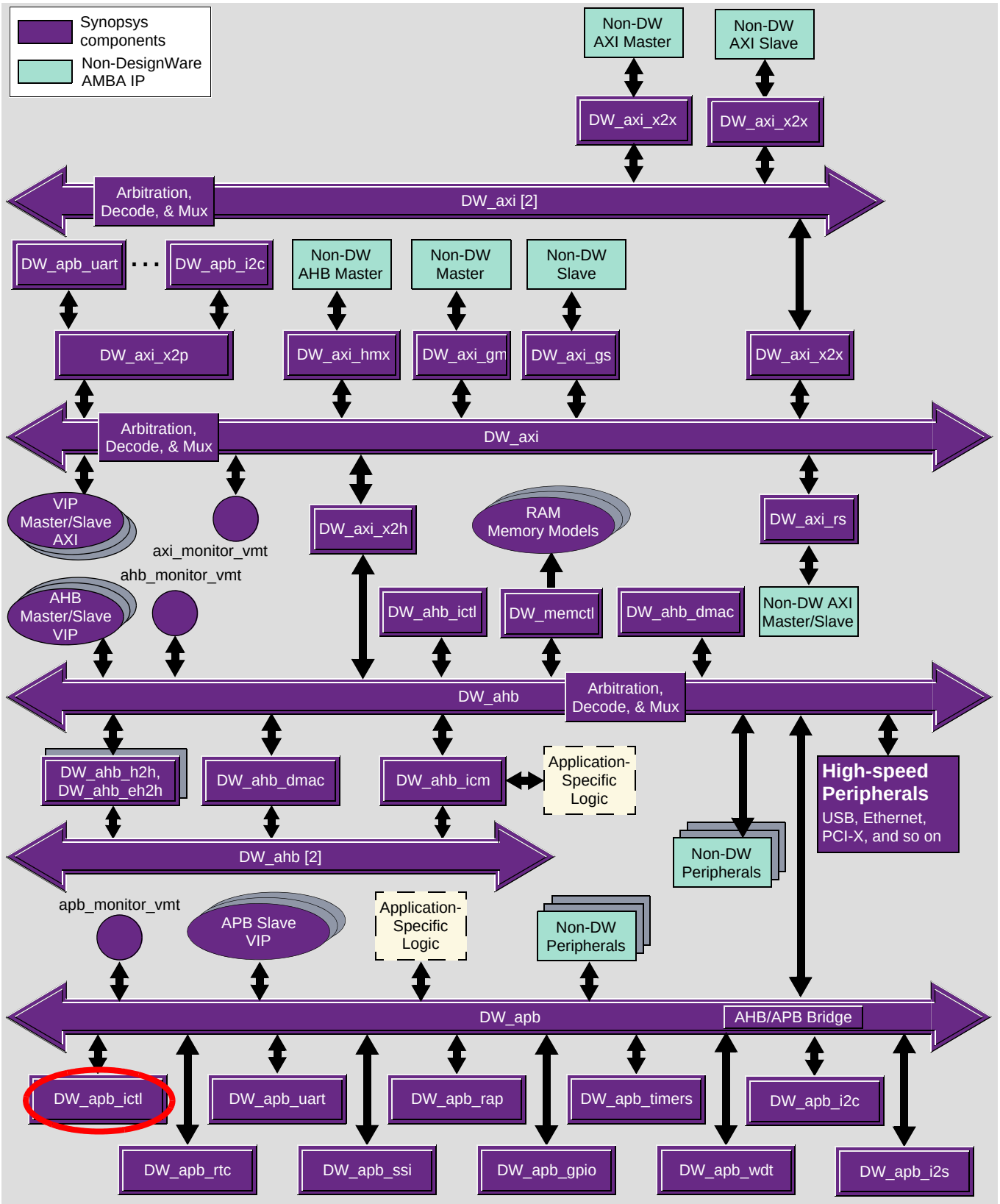
This chapter describes the DesignWare APB Interrupt Controller, referred to as DW_apb_ictl.

1.1 DesignWare System Overview

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

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

Figure 1-1 Example of DW_apb_ictl in a Complete System



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

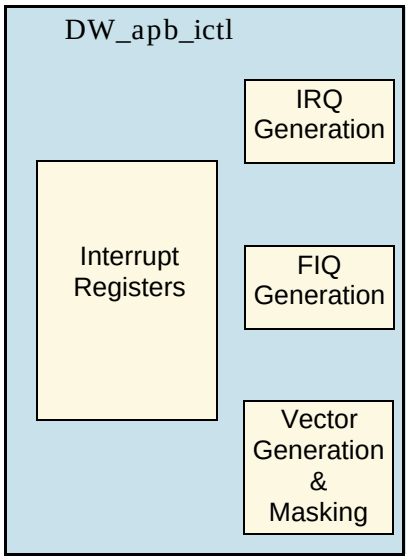
1.2 General Product Description

The DW_apb_ictl is a configurable, vectored interrupt controller for AMBA-based systems. It is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the DesignWare Synthesizable Components for AMBA 2.

1.2.1 DW_apb_ictl Block Diagram

Figure 1-2 shows a block diagram of the DW_apb_ictl.

Figure 1-2 DW_apb_ictl Block Diagram



1.3 Features

The DW_apb_ictl supports the following features:

- 2 to 64 IRQ normal interrupt sources
- 1 to 8 FIQ fast interrupt sources (optional)
- Vectored interrupts (optional)
- Vector Port
- Software interrupts
- Priority filtering (optional)
- Masking

- Scan mode (optional)
- Programmable interrupt priorities (after configuration)

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_ictl 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_ictl 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 89 chapter discusses the specific procedures for verifying the DW_apb_ictl.

1.6 Licenses

Before you begin using the DW_apb_ictl, you must have a valid license. For more information, see "Licenses" section in the *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide*

1.7 Where to Go From Here

At this point, you may want to get started working with the DW_apb_ictl 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_ictl 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_ictl 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 DesignWare APB Interrupt Controller, referred to as DW_apb_ictl. Following topics are covered in this chapter:

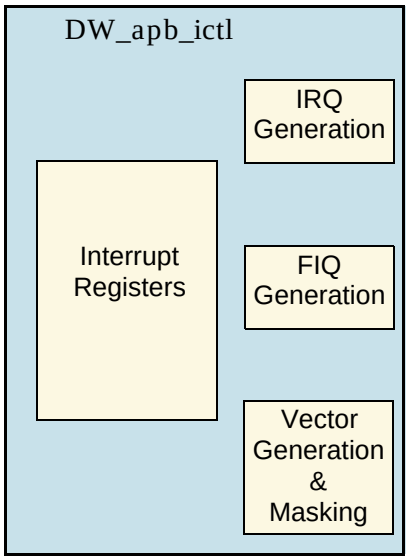
2.1 Overview

The DW_apb_ictl component is a configurable, vectored interrupt controller for DesignWare systems. It supports from 2 to 64 normal interrupt (IRQ) sources that are processed to produce a single IRQ interrupt to the processor. It supports from 1 to 8 fast interrupt (FIQ) sources that are processed to produce a single FIQ interrupt to the processor. All interrupt processing is combinational so that interrupts are propagated if the bus interface of the DW_apb_ictl is powered down. This means that reading any of the interrupt status registers (raw, status, or final_status) is simply returning the status of the combinational logic, since there are no flip-flops associated with these registers. It is the your responsibility to make sure that the interrupts stay asserted until they are serviced.

IRQ interrupts support software interrupts, priority filtering, and vector generation. They have configurable input and output polarity. FIQ interrupts are similar to IRQ interrupts, with the exception that priority filtering and vector generation are not included.

Figure 2-1 shows a block diagram of the DW_apb_ictl.

Figure 2-1 DW_apb_ictl Block Diagram



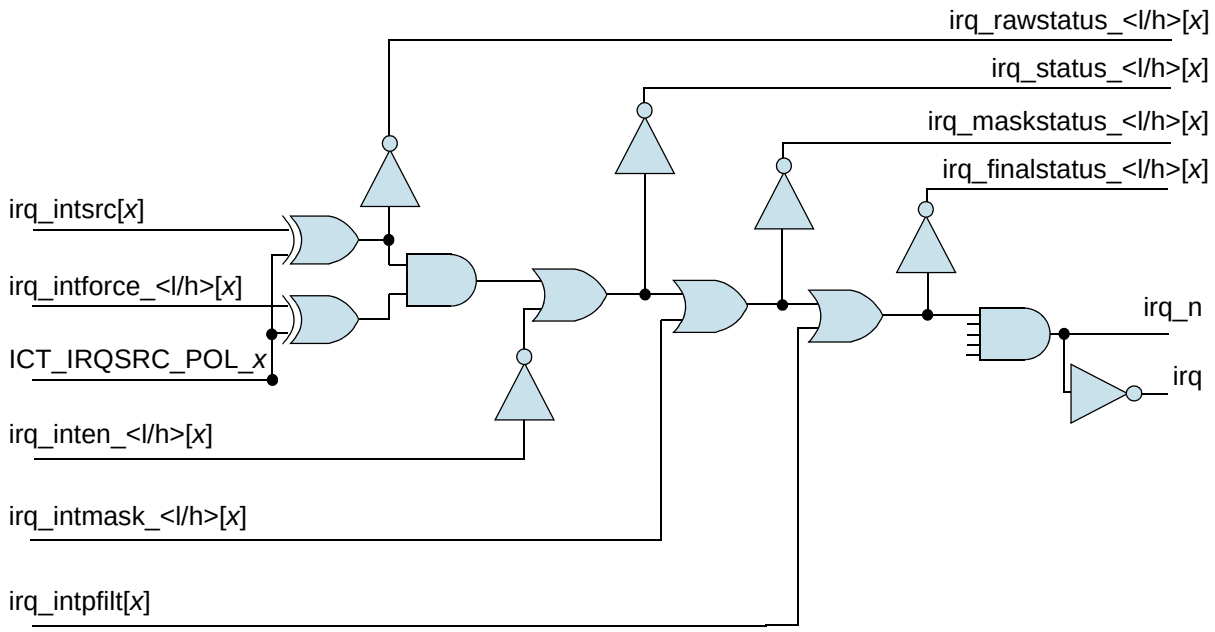
2.2 IRQ Interrupt Processing

The DW_apb_ictl can be configured to support from 2 to 64 IRQ interrupt sources (irq_intsrc*N*) using the ICT_IRQ_NUM configuration parameter. The DW_apb_ictl processes these interrupt sources to produce a single IRQ interrupt to the processor; irq or irq_n.

Note The irq_intpfilt signal is an internally-generated priority mask signal. Its purpose is to mask any IRQ sources with a priority level below the irq_plevel register.

The processing of the interrupt sources is shown in Figure 2-2 and described in the following sections.

Figure 2-2 Normal Interrupt Generation – Interrupt 1 Example



2.2.1 IRQ Interrupt Polarity

The input polarity of each IRQ interrupt source is individually configurable. To configure the input polarity, use the ICT_IRQSRC_POL_*n* parameter. This parameter exists for *n* = 0 to ICT_IRQ_NUM–1 (a polarity parameter for each irq_intsrc bit). Setting one of these parameters to 1 makes the corresponding interrupt source active-high; setting it to 0 makes the corresponding interrupt source active-low. This parameter also determines the polarity of the software-programmable interrupt force bits in the irq_intforce registers (irq_intforce_l or irq_intforce_h).

The output polarity of the interrupt signal is also configurable. The output polarity is set using the ICT_INT_POL parameter. Setting this parameter to 1 configures both the normal and fast interrupt outputs to be active-high; setting it to 0 configures them to be active-low. When configured to active-high, the signal names for the interrupt outputs are irq and fiq; when configured to active-low, the signal names are irq_n and fiq_n.

All interrupt status registers are always active-high, regardless of the polarity configured for the interrupt sources and outputs.

2.2.2 IRQ Software-Programmable Interrupts

The DW_apb_ictl supports forcing interrupts from software. To force an interrupt to be active, write to the corresponding bit in the irq_intforce registers (irq_intforce_l or irq_intforce_h). The polarity of each bit in these registers is the same as the polarity of the corresponding interrupt source signal.

To configure the polarity of both the irq_intsrc signal and the irq_intforce register bits, set the corresponding bits of the ICT_IRQSRC_POL_*n* parameter (*n* = ICT_IRQ_NUM–1). Setting one of these parameters to 1 configures the corresponding bit of irq_intsrc and irq_intforce to be active-high; setting one of these parameters to 0 configures them to be active-low.

Regardless of the polarity you configure, the reset state of each bit in the irq_intforce registers is always inactive.

2.2.3 IRQ Enable and Masking

To enable each interrupt source independently, write a 1 to the corresponding bit of the irq_inten registers (irq_inten_l or irq_inten_h). To configure the reset state of these registers, set the ICT_IRQ_DFLT_EN parameter. If you set a bit of the ICT_IRQ_DFLT_EN parameter to 1, the corresponding bits of the irq_inten registers are 1 on reset, which enables the corresponding interrupt source.

To independently mask each interrupt source, write a 1 to the corresponding bit of the interrupt mask register (irq_maskstatus_l/irq_maskstatus_h). The reset value for each mask bit is 0 (unmasked).

2.2.4 IRQ Software-Programmable Priority Levels

The DW_apb_ictl supports optional software programmable priority levels. To change the priority level of an interrupt, you write the priority value to the corresponding priority level register in the memory map. There is a priority register for each of the interrupt sources, which can be programmed to one of sixteen values from 0x0 to 0xf. Priority registers exist for only available interrupt sources.

The priority registers take on the reset state of the configuration parameter ICT_IRQSRC_PLEVEL_*n*.

- To enable reading the priorities, set the ICT_READ_PRIORITY parameter to 1. It is possible to read the priority registers when the priorities are not programmable.
- To enable programmable priorities, set the ICT_HC_PRIORITIES parameter to 0. This can be done only if priorities can be read; that is, if ICT_READ_PRIORITIES is 1.

2.2.5 IRQ Priority Filter

The DW_apb_ictl supports optional priority filtering. To enable the priority filtering logic, set the ICT_HAS_PFLT parameter to 1. If ICT_HAS_PFLT is set to 0, none of the associated logic is instantiated and vectored interrupts are not supported.

The function of the priority filtering logic is described as follows:

- Each interrupt source is configured to one of sixteen priority levels. To configure an interrupt source to a priority level, set the ICT_IRQSRC_PLEVEL_*n* to a value from 0 to 15, where 0 is the lowest priority, assuming that programmable priorities is not enabled. If programmable priorities are enabled (ICT_HC_PRIORITIES = 0), then the priority registers can be changed dynamically after configuration.
- A system priority level can be programmed into the irq_plevel register, which holds values from 0 to 15. The reset value of this register is set by programming the ICT_IRQ_PLEVEL parameter to a value from 0 to 15.
- The DW_apb_ictl filters out any interrupt source with a configured priority level less than the priority currently programmed in the irq_plevel register.

2.2.6 IRQ Interrupt Status Registers

The DW_apb_ictl includes up to four status registers used for querying the current status of any interrupt at various stages of the processing. Refer to Figure 2-2 for an illustration of the register values. All of the following status registers have the same polarity; a 1 indicates that an interrupt is active, a 0 indicates it is inactive.

- irq_rawstatus (irq_rawstatus_l/irq_rawstatus_h) – Contains the state of the interrupt sources after being adjusted for input polarity. Each bit of this register is set to 1 if the corresponding interrupt source bit is active, and is set to 0 if it is inactive.
- irq_status (irq_status_l/irq_status_h) – Contains the state of all interrupts after the enabling stage; that is, an active-high bit indicates that a particular interrupt source is active and enabled.
- irq_maskstatus (irq_maskstatus_l/irq_maskstatus_h) – Contains the state of all interrupts after the masking stage; that is, an active-high bit indicates that a particular interrupt source is active, enabled, and not masked.
- irq_finalstatus (irq_finalstatus_l/irq_finalstatus_h) – Contains the state of all interrupts after the priority filtering stage; that is, an active-high bit indicates that particular interrupt source is active, enabled, not masked, and its configured priority level is greater or equal to the value programmed in the irq_plevel register. If priority filtering has not been selected, this register contains the same value as the irq_maskstatus register (the final stage of processing).

2.2.7 IRQ Interrupt Vectors

The DW_apb_ictl can be configured to support interrupt vectors. To enable vectored interrupt support, the user must include priority filtering logic by setting the ICT_HAS_PFLT parameter to 1. To include vector generation logic, set the ICT_HAS_VECTOR parameter to 1. If ICT_HAS_VECTOR is set to 0, none of the interrupt vector logic is instantiated, and vectored interrupts are not supported.

The DW_apb_ictl has one vector register associated with each of the sixteen interrupt priority levels: irq_vector_0 through irq_vector_15. These registers are 32 bits wide.

The value of each interrupt vector register can be hardcoded or programmed. If the parameter ICT_HC_VECTOR_*n* is set to 1, irq_vector_n has a hardcoded value and is a read-only register. If the parameter ICT_HC_VECTOR_*n* is set to 0, irq_vector_n is programmable and is a read/write value.

If configuration parameter ICT_HAS_DEFAULT_VECTOR is set to 1, register irq_vector_default is included. This register can be used to return a known vector when the irq_vector register is read and no interrupts are active. The value of this register can be hardcoded or programmed. If ICT_HC_VECTOR_DEFAULT is set to 1, irq_vector_default has a hardcoded value and is read-only. If ICT_HC_VECTOR_DEFAULT is set to 0, irq_vector_default is programmable and has a read/write value.

To configure the vector register value for hardcoded vector registers and the reset value for programmable vector registers, set the ICT_VECTOR_*n* parameter to the desired value.

Vector processing proceeds as follows:

- Active interrupts are conditioned by their enable and mask control bits.
- All active interrupts that have a configured priority level (ICT_IRQSRC_PLEVEL_*n*) less than the current value programmed into the irq_plevel register are filtered out.

- The highest priority level from among the remaining active interrupts is used to select one of the sixteen interrupt vectors that have been programmed or configured into the irq_vector registers.
- The user retrieves the vector associated with the highest priority level that has an active interrupt source by reading the irq_vector register

The irq_vector register is “read coherent” – that is, you need to be guaranteed that you are reading a valid value for the entire vector. In a system where the APB data width is less than the width of the irq_vector register, the contents of irq_vector is stored in a shadow location when the user starts to read the irq_vector register so that the irq_vector register can be read without being corrupted by it being changed by subsequent interrupts occurring. For more information on coherency, see “Integration Considerations” on page 93.

2.3 Vector Port

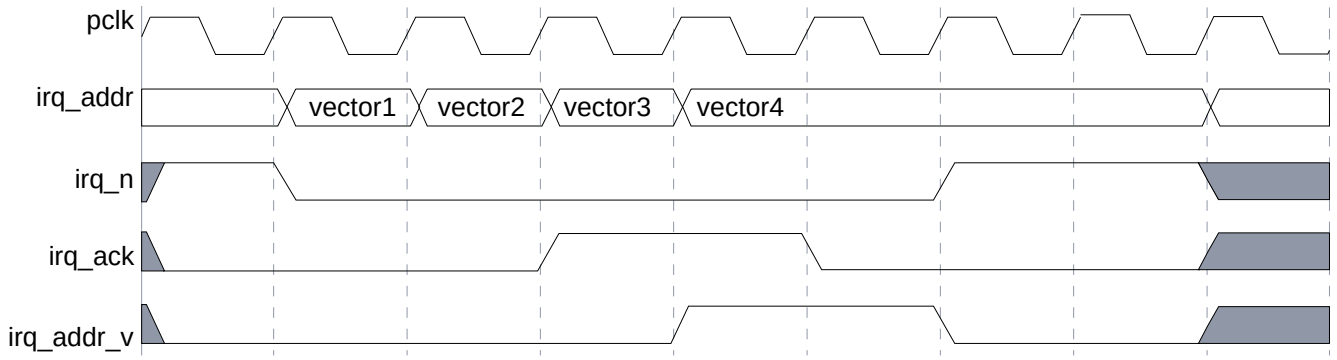
You can configure the DW_apb_ictl to include vector port functionality through the ICT_ADD_VECTOR_PORT parameter. If you enable this parameter, signals are added to the top level of the DW_apb_ictl that allows a processor to quickly sample the vector address associated with a currently pending IRQ.

Using the vector port potentially decreases IRQ service latency, as the processor does not need to initiate an APB bus transaction in order to discover the vector address associated with the current highest-priority interrupt.

The DW_apb_ictl vector port supports the ARM11 and ARM1026EJ processor VIC ports. Vector port functionality applies only to IRQ processing; FIQ processing remains unaffected.

Figure 2-3 shows the operation of the vector port where the processor clock and bus clock are running at the same frequency.

Figure 2-3 Vector Port Handshaking (Synchronous Processor Clock)



2.3.1 Handshaking Operation

The irq_addr output becomes valid in the same cycle that irq_n asserts. At some point after this, the processor asserts irq_ack to acknowledge receipt of the IRQ, which then causes the DW_apb_ictl to assert irq_addr_v in order to inform the processor that it may sample irq_addr. Until the DW_apb_ictl asserts irq_addr_v, irq_addr changes combinatorially to reflect the vector associated with the currently highest-priority active interrupt source. The DW_apb_ictl holds irq_addr static while irq_addr_v is asserted.

In the same cycle that the DW_apb_ictl de-asserts irq_addr_v, irq_n also de-asserts. At this point in the handshaking, the DW_apb_ictl has reset its internal priority filter to mask out all interrupt sources of a

priority level less than or equal to the priority of the interrupt source just sampled by the processor; this prevents the processor from re-sampling the same interrupt and also prevents lower-priority interrupt sources from causing an assertion of irq_n during the interrupt service routine (ISR) of the sampled interrupt. If the interrupt source currently being processed has the highest possible priority level (4'hF), then all IRQs are masked until software resets the priority filter.

The processor can reset the level of the priority filter to its previous setting by writing a new priority level to the irq_internal_plevel register. Ideally this should be one of the last steps in the ISR code.

If a higher-priority interrupt occurs during the handshaking process, irq_n stays asserted when irq_addr_v de-asserts, and the DW_apb_ictl waits for the processor to assert irq_ack in order to start the handshaking process for the new interrupt.

Figure 2-3 shows the vector port handshaking when the bus clock and processor clock are identical (synchronous). In this case, there is a one-cycle delay from the assertion of irq_ack by the processor to the assertion of irq_addr_v by the DW_apb_ictl. Also there is a one-cycle delay from the de-assertion of irq_ack by the processor to the de-assertion of irq_addr_v by the DW_apb_ictl.

2.3.2 Priority-Level Registers

If the ICT_ADD_VECTOR_PORT parameter is equal to 1, you can use a priority-level memory-mapped register—irq_internal_plevel—to sample the priority level currently being applied by the DW_apb_ictl; if ICT_ADD_VECTOR_PORT is equal to 0, irq_internal_plevel does not exist.

Unless an IRQ is currently being handled, the irq_internal_plevel register reflects the value of the system priority level register, irq_plevel. While an IRQ is being handled, irq_internal_plevel is set to one priority level greater than the priority of the IRQ being processed. In contrast to the irq_plevel register that is 4 bits wide, the irq_internal_plevel register is 5 bits wide in order to accommodate stacking an IRQ source with a priority level of 15; in this case, irq_internal_plevel is set to 16.

At the end of the ISR, the processor writes to the irq_internal_plevel register in order to reset it to the value stored in the irq_plevel register. Writing to the irq_internal_plevel register at any other time may break the operation of the DW_apb_ictl.

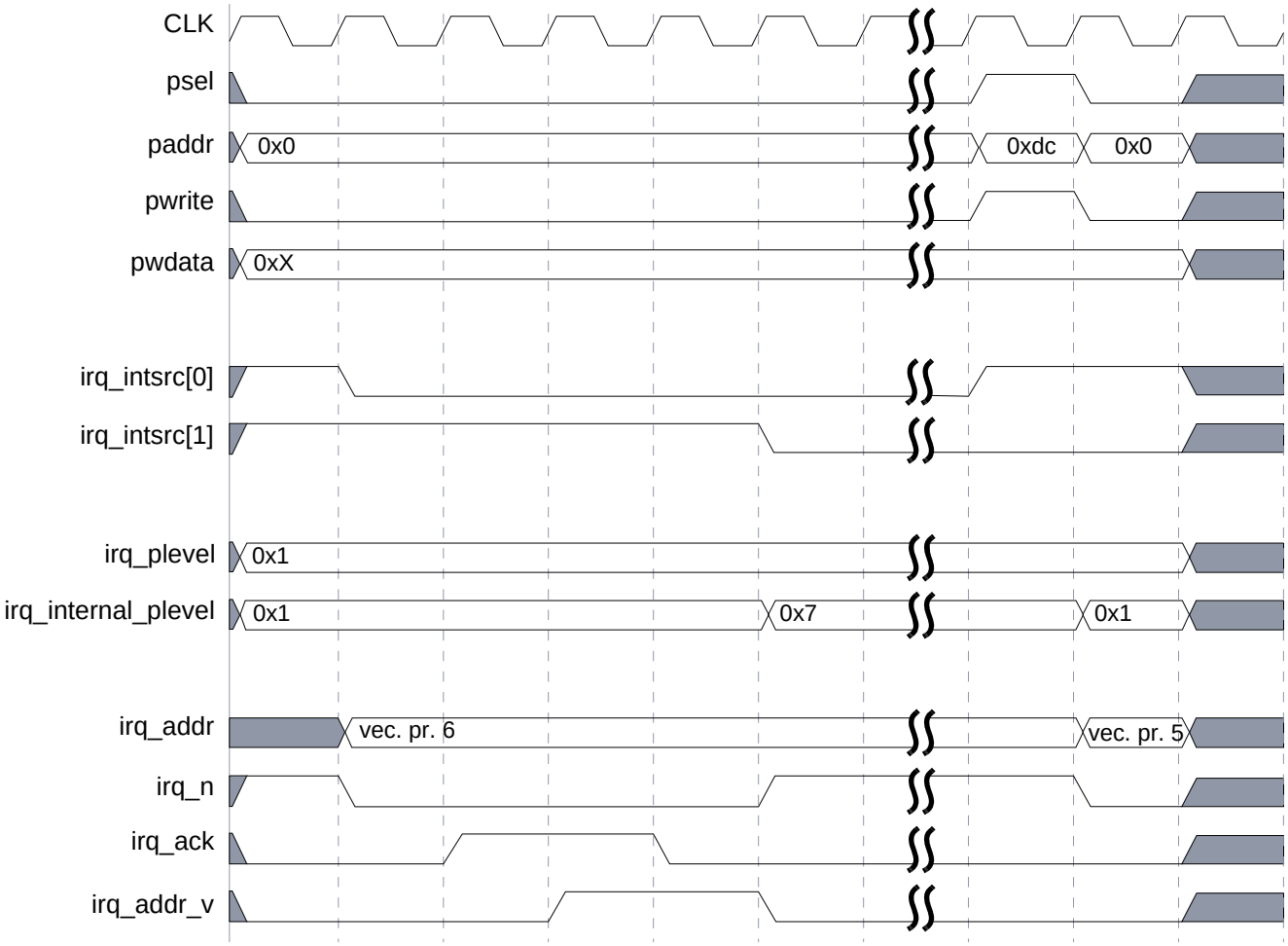
Note For writes to the irq_internal_plevel register, the write data on the bus is ignored. The only effect of a write to the irq_internal_plevel register is to reset its value to that of the irq_plevel register.

Splitting up the system priority level into two separate locations enables you to have read / write access to the current system priority level independent of the temporary priority level used during the ISR. If a higher priority IRQ occurs while a stacked priority is being used, irq_internal_plevel is reset to the value of irq_plevel from the next hclk cycle. This new IRQ is stacked after the handshaking procedure, described in “Handshaking Operation” on page 22.

If the ICT_ADD_VECTOR_PORT parameter is set to false, the irq_internal_plevel register does not exist, and only the irq_plevel register is used to set the current priority level of the DW_apb_ictl.

Figure 2-4 shows how the priority filter value is changed by the DW_apb_ictl for the duration of the ISR.

Figure 2-4 Priority Masking During ISR



In this example, a lower-priority interrupt occurs after the handshaking but before the irq_internal_plevel register is reset by the processor. Sometime later the processor writes to the irq_internal_plevel register in order to reset the priority filter level, and the lower-priority interrupt is allowed to propagate to the processor; the bus data is ignored in the write to the irq_internal_plevel register.

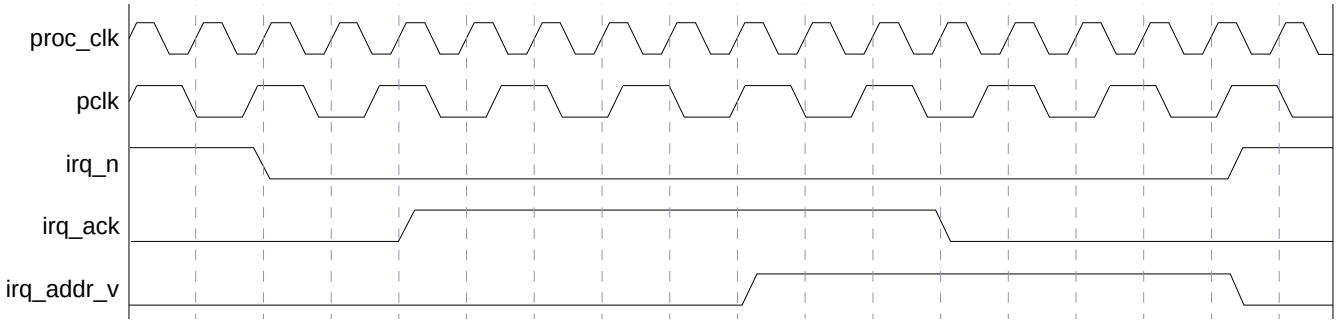
2.3.3 Synchronization

You can configure the DW_apb_ictl using the ICT_ADD_VECTOR_PORT_SYNC parameter to add synchronization to vector port signals in order to support processors running at a different asynchronous frequency to the bus clock. If this parameter is enabled, the DW_apb_ictl adds N-stages of pclk register synchronization to all signals coming from the processor. Where N= ICT_ADD_VECTOR_PORT_SYNC_DEPTH. By default, DW_apb_ictl adds 2-stages of pclk register synchronization to all signals coming from the processor.

Figure 2-5 shows vector port handshaking with synchronization enabled. Processor signals coming into the DW_apb_ictl go through two levels of metastability registers that give protection from one hclk cycle of

metastability. In this situation, signals from the DW_apb_ictl to the processor should be synchronized external to the DW_apb_ictl.

Figure 2-5 Vector Port Handshaking with Synchronization



Note N cycles of synchronization time are additional to the one cycle latency described for the synchronous processor and AHB bus clocks, which yields a total of three cycles of latency between irq_ack and irq_addr_v.

Synchronization is not required for integer multiple or quasi-synchronous processor and bus clocks. This is achieved when the parameter ICT_ADD_VECTOR_PORT_SYNC = 0.

2.3.4 Interrupt Timing

The following describes basic interrupt handling steps using the vector port feature of the DW_apb_ictl:

1. Sample vector associated with IRQ using DW_apb_ictl vector port.
2. If priority level is shared among IRQ sources, read irq_final_status to determine which interrupt source caused the interrupt.
3. Execute interrupt service routine.
4. Optionally read irq_status to see if other interrupts are pending and service as required.
5. Clear interrupt at source (hardware peripheral or interrupt controller for s/w interrupt).
6. Write to irq_internal_plevel to reset the priority filter.

2.4 FIQ Interrupt Processing

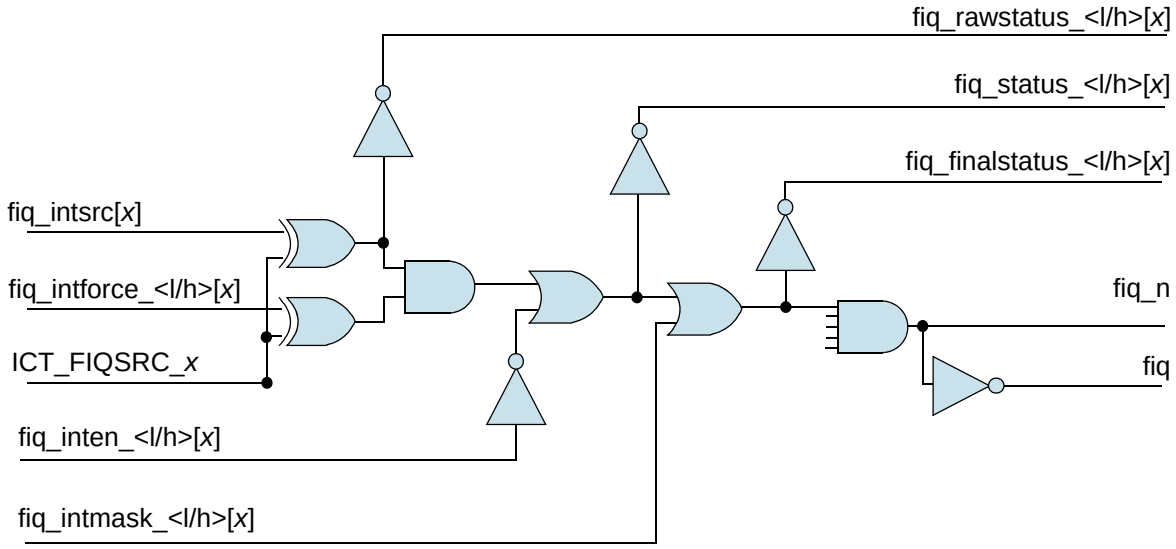
FIQ interrupts are an optional feature of the DW_apb_ictl interrupt controller. To include FIQ interrupt logic in the DW_apb_ictl, set the value of the ICT_HAS_FIQ parameter to 1. If ICT_HAS_FIQ is set to 0, the FIQ logic is not included, and the associated signals are not present.

You can configure the DW_apb_ictl to support from 1 to 8 FIQ interrupt sources (fiq_intsrc*N*) using the ICT_FIQ_NUM parameter. The DW_apb_ictl processes these interrupt sources to produce a single FIQ interrupt to the processor; fiq or fiq_n.

FIQ interrupt processing is similar to IRQ interrupt processing, except that priority filtering and interrupt vectors are not supported for the FIQ interrupts. This section describes how the DW_apb_ictl handles the FIQ interrupt processing.

Figure 2-6 shows the processing of the interrupt sources, which is described in the following sections.

Figure 2-6 Fast Interrupt Generation – Interrupt 1 Example



2.4.1 FIQ Interrupt Polarity

The input polarity of each FIQ interrupt source can be individually configured. To configure the input polarity, use the ICT_FIQSRC_POL parameter. This parameter exists for *n* = 0 to ICT_FIQ_NUM–1 (a polarity parameter for each fiq_intsrc bit). Setting one of these parameters to 1 makes the corresponding interrupt source active-high; setting it to 0 makes the corresponding interrupt source active-low. This parameter also determines the polarity of the software programmable interrupt force bits in the fiq_intforce register.

The output polarity of the interrupt signal can also be configured. You configure the output polarity using the ICT_INT_POL parameter. Setting this parameter to 1 configures both the IRQ and FIQ interrupt outputs to be active-high; setting the parameter to 0 configures them to be active-low. When configured as active-high, the signal names for the interrupt outputs are irq and fiq; when configured active-low, the signal names are irq_n and fiq_n.

All interrupt status registers are always active-high, regardless of the polarity configured for the interrupt sources and outputs.

2.4.2 FIQ Software-Programmable Interrupts

The DW_apb_ictl supports forcing interrupts from software. You force an interrupt to be active by writing to the corresponding bit in the fiq_intforce register. The polarity of each bit in this register is the same as the polarity of the corresponding interrupt source signal.

To configure the polarity of both the fiq_intsrc signal and the fiq_intforce register bits, set the corresponding bits of the ICT_FIQSRC_POL_*n* parameter (*n* = ICT_FIQ_NUM–1). Setting one of these parameters to 1 configures the corresponding bit of fiq_intsrc and fiq_intforce to be active-high; setting one of these parameters to 0 configures them to be active-low.

Regardless of the polarity you configure, the reset state of each bit in the fiq_intforce register is always inactive.

2.4.3 FIQ Enable and Masking

You can enable each interrupt source independently by writing a 1 to the corresponding bit of the fiq_inten register. You can configure the reset state of this register by setting the ICT_FIQ_DFLT_EN parameter. If you set a bit of the ICT_FIQ_DFLT_EN parameter to 1, the corresponding bit of the fiq_inten register is 1 on reset, which enables the corresponding interrupt source.

You can mask each interrupt source independently by writing a 1 to the corresponding bit of the fiq_intmask register. The reset value for each mask bit is 0; that is, unmasked.

2.4.4 FIQ Interrupt Status Registers

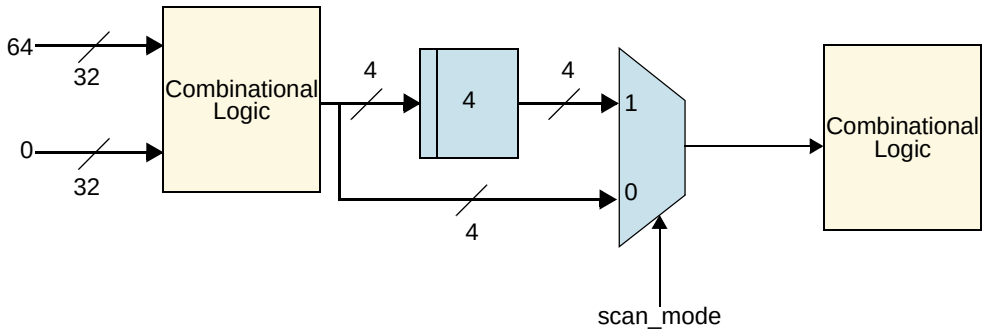
The DW_apb_ictl includes three status registers that you can use to query the current status of any FIQ interrupt at various stages of the processing. For an illustration of the register values, see Figure 2-6. All of the following status registers have the same polarity; a 1 indicates that an interrupt is active, a 0 indicates inactive.

- fiq_rawstatus – Contains the state of the interrupt sources after being adjusted for input polarity. Each bit of this register is set to 1 if the corresponding interrupt source bit is active; it is set to 0 if it is inactive.
- fiq_status – Contains the state of all interrupts after the enabling stage; that is, an active-high bit indicates that a particular interrupt source is active and enabled.
- fiq_finalstatus – Contains the state of all interrupts after the masking; that is an active-high bit indicates that a particular interrupt source is active, enabled, and unmasked.

2.5 Scan Mode

This input is an optional signal and is included only when vector generation is enabled. The scan_mode input signal should be asserted high when scan testing is performed on the design. The DW_apb_ictl includes shadow registers in a complicated combinational path so that when test_mode is asserted, the output of these shadow registers is multiplexed into the data path to provide increased observability and controllability of the logic in this path. This is illustrated in Figure 2-7.

Figure 2-7 Combinational Logic of Scan Mode Shadow Registers



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.

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
ICTL Source Code Configuration	
Use DesignWare Foundation Synthesis Library	Specifies whether the DesignWare Foundation Synthesis Library must be used. The component code utilizes DesignWare Foundation parts for optimal Synthesis QoR. Customers with only a DesignWare license must use Foundation parts. Customers with only a Source license cannot use Foundation parts. Customers with both Source and DesignWare licenses have the option of using Foundation parts. **Values:** ■ false (0) ■ true (1) **Default Value:** True if DesignWare License is available; False if no DesignWare License is available **Enabled:** Parameter is enabled if customer has both Source and DesignWare licenses. **Parameter Name:** USE_FOUNDATION
System Configuration	
APB Data bus width	Specifies the APB system data bus width. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** APB_DATA_WIDTH
Make irq_intforce and fiq_intforce active high?	When this parameter is set to 1, the irq_intforce and fiq_intforce register become active-high. Writing a 1 to the corresponding interrupt source bit forces an interrupt for that source, regardless of the interrupt sources' configured polarity. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** ICT_FORCEREG_ACTIVE_HIGH
Active-high Level for IRQ/FIQ outputs?	When this parameter is set to 1, the polarity of the FIQ and IRQ interrupt output signals is active-high. Both fast and normal interrupts are of the same polarity. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** ICT_INT_POL

Table 3-1 Top Level Parameters (Continued)

Label	Description
IRQ Configuration	
Install Priority Controller?	When this parameter is set to 1, it allows interrupts that are assigned a priority level to be compared against a system-level priority level. If the interrupt source has a priority level that is greater than or equal to the system level, then it is not masked. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** ICT_HAS_PFLT
Be able to read back priorities?	When this parameter is set to 1, the priority levels can be read. If set to 0, the priority levels cannot be read. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_READ_PRIORITY
Hard-Coded Priorities?	When this parameter is set to 1, the priority levels cannot be programmed. If set to 0, the priority levels can be programmed. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** ICT_READ_PRIORITY==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_HC_PRIORITIES
Install Vector Generation?	Instantiates the interrupt vector generation circuitry and registers in the DW_apb_ictl. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_HAS_VECTOR_USER

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include Default Vector Logic?	Instantiates the irq_vector_default register and circuitry. When activated, the value read from the irq_vector register, when no interrupts are pending, is the value in the irq_vecter_default register. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_HAS_PFLT==1 && ICT_HAS_VECTOR_USER==1 **Parameter Name:** ICT_HAS_DEFAULT_VECTOR
Number of irq sources	Defines the number of interrupt sources to generate. **Values:** 2, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** ICT_IRQ_NUM
IRQ Source Polarity Type	Specifies the IRQ Source Polarity Type. You can use either the individual interrupt polarities or override these to be all active high or all active low. **Values:** ■ Individual (0) ■ All-Active-Low (1) ■ All-Active-High (2) **Default Value:** Individual **Enabled:** Always **Parameter Name:** ICT_IRQSRC_POL_TYPE
Individual irq enables on reset	Specifies the reset value of the IRQ interrupt source enable register (irq_inten). A logic '1' in any bit position indicates that the interrupt source corresponding to that bit is enabled on reset; a logic '0' indicates that it is not enabled on reset. **Values:** 0x0 to 0xffffffffffffffff **Default Value:** {multi} {ICT_IRQ_NUM} {0b0} **Enabled:** Always **Parameter Name:** ICT_IRQ_DFLT_EN
FIQ Configuration	
Install Fast Interrupt Generation?	Instantiates the generation logic for fast interrupts. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** ICT_HAS_FIQ

Table 3-1 Top Level Parameters (Continued)

Label	Description
Number of fiq sources	Defines the number of fast interrupt sources to generate. This parameter can be set only if ICT_HAS_FIQ is set to True (1). **Values:** 1, 2, 3, 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** ICT_HAS_FIQ==1 **Parameter Name:** ICT_FIQ_NUM
FIQ Source Polarities	Specifies the FIQ Source Polarity Type. You can use either the individual interrupt polarities or override these to be all active high or all active low. **Values:** ■ Individual (0) ■ All-Active-Low (1) ■ All-Active-High (2) **Default Value:** Individual **Enabled:** ICT_HAS_FIQ==1 **Parameter Name:** ICT_FIQSRC_POL_TYPE
Individual fiq enables on reset	Specifies the reset state of the FIQ interrupt source enable register. A logic '1' in any bit position indicates that the fast interrupt source corresponding to that bit is enabled on reset; a logic '0' indicates that it is not enabled on reset. **Values:** 0x0 to 0xff **Default Value:** {multi} {ICT_FIQ_NUM} {0b0} **Enabled:** ICT_HAS_FIQ==1 **Parameter Name:** ICT_FIQ_DFLT_EN

3.2 Vector Port Interface Parameters

Table 3-2 Vector Port Interface Parameters

Label	Description
Vector Port Configuration	
Add Vector Port Interface ?	Selects whether or not to include vector port signals and functionality in the interrupt controller. The vector port allows a processor with similar functionality to sample the IRQ vector address directly without performing an APB bus access, thereby potentially improving interrupt service latency. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ((ICT_HAS_PFLT==1)?ICT_HAS_VECTOR_USER:0)==1 **Parameter Name:** ICT_ADD_VECTOR_PORT
Add Vector Port Interface Synchronisation ?	Adds a N-stage (where N = ICT_ADD_VECTOR_PORT_SYNC_DEPTH) pclk synchronization on vector port signals coming from the processor (irq_ack). This parameter is used when the processor clock and pclk are asynchronous. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_ADD_VECTOR_PORT==1 **Parameter Name:** ICT_ADD_VECTOR_PORT_SYNC
Number of synchronization stages on vector port signal coming from processor?	Selects the number of pclk synchronization stages on vector port signal. **Values:** 2, 3, 4 **Default Value:** 2 **Enabled:** ICT_ADD_VECTOR_PORT_SYNC==1 **Parameter Name:** ICT_ADD_VECTOR_PORT_SYNC_DEPTH

3.3 Priority Controller Configuration Parameters

Table 3-3 Priority Controller Configuration Parameters

Label	Description
Priority Controller Configuration	
System priority controller filter level	Defines the default system priority controller filter level. This is the reset value of the interrupt priority level filter register. Interrupts must have a priority greater than or equal to this value to be propagated to the CPU. This value may always be overwritten by the software and is required only when the priority filter is installed. **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf **Default Value:** 0x0 **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_IRQ_PLEVEL
Priority level of IRQ Source n (for n = 0; n <= ICT_IRQ_NUM-1)	This parameter sets the default priority level for each normal interrupt source. 0 is the lowest priority. **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf **Default Value:** 0x0 **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_ISRC_PLEVEL_n

3.4 Configuration of Vector Generation Module Parameters

Table 3-4 Configuration of Vector Generation Module Parameters

Label	Description
Configuration of Vector n	
Priority n vector (for n = 0; n <= ICT_IRQ_PLEVEL)	Specifies the vector for interrupts with a priority setting of n. The value must be less than HADDR_WIDTH. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0x0 **Enabled:** ICT_HAS_VECTOR_USER==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_VECTOR_n
Hardcode Vector n (for n = 0; n <= ICT_IRQ_PLEVEL)	Specifies that the corresponding priority vector is hardcoded and the irq_vector_n register is read only. If this parameter is set to False (0), the corresponding priority vector is programmable and the corresponding irq_vector_n register is read/write. **Values:** 0, 1 **Default Value:** 0 **Enabled:** ICT_HAS_VECTOR_USER==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_HC_VECTOR_n
Configuration of Default Vector	
Default Priority vector	Specifies the reset value for the irq_vector_default register. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0x0 **Enabled:** ICT_HAS_DEFAULT_VECTOR==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_VECTOR_DEFAULT
Hardcode Default Vector	Specifies that the default priority vector (returned on a read from the irq_vector register when no source IRQs are active) is hardcoded and the irq_vector_default register is read only. If this parameter is set to False (0), the corresponding vector is programmable and the corresponding irq_vector_default register is read/write. **Values:** 0, 1 **Default Value:** 0 **Enabled:** ICT_HAS_DEFAULT_VECTOR==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_HC_VECTOR_DEFAULT

3.5 Individual IRQ Polarity Configuration Parameters

Table 3-5 Individual IRQ Polarity Configuration Parameters

Label	Description
Individual IRQ Polarity Configuration	
Interrupt irq n Active High ? (for n = 0; n <= ICT_IRQ_NUM-1)	Sets the interrupt level of interrupt n to either active high (1) or active low (0). **Values:** 0x0, 0x1 **Default Value:** True (1) if ICT_IRQSRC_POL_TYPE is not 1; that is, if it is not All-active-low. **Enabled:** ICT_IRQSRC_POL_TYPE == 0 && ICT_IRQ_NUM > 0 **Parameter Name:** ICT_IRQSRC_POL_n

3.6 Individual FIQ Polarity Configuration Parameters

Table 3-6 Individual FIQ Polarity Configuration Parameters

Label	Description
Individual FIQ Polarity Configuration	
Interrupt n Active High ? (for n = 0; n <= ICT_FIQ_NUM-1)	Sets the interrupt level of interrupt n to either active high (1) or active low (0). **Values:** 0x0, 0x1 **Default Value:** True (1) if ICT_FIQSRC_POL_TYPE is not 1; that is, if it is not All-active-low. **Enabled:** ICT_HAS_FIQ==1 && ICT_FIQSRC_POL_TYPE == 0 && ICT_FIQ_NUM > 0 **Parameter Name:** ICT_FIQSRC_POL_n

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 clock(s) 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: Name of configuration parameter(s) that populates this signal in your configuration.

Validated by: Assertion or de-assertion of signal(s) 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 41
- Miscellaneous on page 43
- Interrupt on page 44
- Vector Interrupt and Handshake Signals on page 45
- Interrupt Source on page 46

4.1 APB Interface Signals

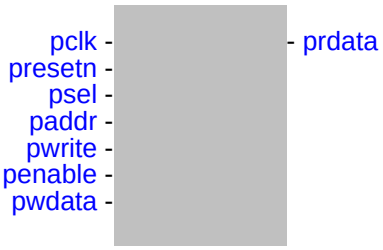


Table 4-1 APB Interface Signals

Port Name	I/O	Description
pclk	I	APB clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
presetn	I	APB interface domain reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of pclk. DW_apb_ictl 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
psel	I	APB peripheral select. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
paddr[HADDR_REGFILE_SLICE_LHS:0]	I	APB address bus. **Exists:** Always **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
pwrite	I	APB write control. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
penable	I	APB enable control that indicates the second cycle of the APB frame. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
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
prdata[(APB_DATA_WIDTH-1):0]	O	APB read back data. **Exists:** Always **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Miscellaneous Signals

Table 4-2 Miscellaneous Signals

Port Name	I/O	Description
scan_mode	I	Optional. Scan mode. This signal helps increase fault coverage in the design. During scan testing, scan_mode must be asserted that is, tied to logic 1. At all other times, this signal must be deasserted tied to logic 0. **Exists:** (ICT_HAS_PFLT==1) **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
irq_ack	I	Optional. Asserted by processor to acknowledge receipt of the IRQ. **Exists:** (ICT_ADD_VECTOR_PORT==1) **Synchronous To:** ICT_ADD_VECTOR_PORT_SYNC==0 ? "pclk" : "Asynchronous" **Registered:** ICT_ADD_VECTOR_PORT_SYNC==1 ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.3 Interrupt Signals

Table 4-3 Interrupt Signals

Port Name	I/O	Description
irq	O	Optional. Active-high normal interrupt. **Exists:** ICT_INT_POL==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
fiq	O	Optional. Active-high fast interrupt. **Exists:** (ICT_INT_POL==1) && (ICT_HAS_FIQ==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
irq_n	O	Optional. Active-low normal interrupt. **Exists:** ICT_INT_POL==0 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
fiq_n	O	Optional. Active-low fast interrupt. **Exists:** (ICT_INT_POL==0) && (ICT_HAS_FIQ==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.4 Vector Interrupt and Handshake Signals

Table 4-4 Vector Interrupt and Handshake Signals

Port Name	I/O	Description
irq_addr[(HADDR_WIDTH-1):0]	O	Optional. Address vector sampled by the processor during vector port handshaking. **Exists:** (ICT_ADD_VECTOR_PORT==1) **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
irq_addr_v	O	Optional. Asserted by DW_apb_ictl to inform processor that it may sample irq_addr. **Exists:** (ICT_ADD_VECTOR_PORT==1) **Synchronous To:** pclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Interrupt Source Signals

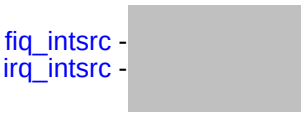


Table 4-5 Interrupt Source Signals

Port Name	I/O	Description
fiq_intsrc[(ICT_FIQ_NUM-1):0]	I	Interrupt source bus for fast interrupt generation. **Exists:** ICT_HAS_FIQ==1 **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
irq_intsrc[(ICT_IRQ_NUM-1):0]	I	Interrupt source bus for normal interrupt generation. **Exists:** Always **Synchronous To:** pclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5 Register Descriptions

This chapter details all possible registers in the controller. They are arranged hierarchically into maps and blocks (banks). For configurable IP titles, 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 to this register once field.
W1C	A write of 1 clears this register field.
W1S	A write of 1 sets this register field.
W1T	A write of 1 toggles this register field.
W0C	A write of 0 clears this register field.
W0S	A write of 0 sets this register field.
W0T	A write of 0 toggles this register field.
WC	Any write clears this register field.
WS	Any write sets this register field.
WM	Any write toggles this register field.
no Read Behavior attribute	You cannot read this register. It is Write-Only.
no Write Behavior attribute	You cannot write to this register. It is Read-Only.

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
DW_apb_ictl_addr_block1 on page 50	DW_apb_ictl address block **Exists:** Always

5.1 DW_apb_ictl_mem_map/DW_apb_ictl_addr_block1 Registers

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

Table 5-5 Registers for Address Block: DW_apb_ictl_mem_map/DW_apb_ictl_addr_block1

Register	Offset	Description
IRQ_INTEN_L on page 52	0x0	This register specifies the interrupt enable bits for lower 32 interrupt sources.
IRQ_INTEN_H on page 53	0x4	This register enables the upper 32 interrupt sources.
IRQ_INTMASK_L on page 54	0x8	This register masks the lower 32 interrupt sources.
IRQ_INTMASK_H on page 55	0xc	This register masks the upper 32 interrupt sources.
IRQ_INTFORCE_L on page 56	0x10	This register specifies the interrupt force bits for the lower 32 interrupt sources.
IRQ_INTFORCE_H on page 58	0x14	This register specifies the interrupt force bits for the upper 32 interrupt sources.
IRQ_RAWSTATUS_L on page 60	0x18	This register specifies the raw status of lower 32 interrupt sources.
IRQ_RAWSTATUS_H on page 62	0x1c	This register specifies the raw status of the upper 32 interrupt sources.
IRQ_STATUS_L on page 63	0x20	This register specifies the interrupt Status of the lower 32 interrupt sources.
IRQ_STATUS_H on page 64	0x24	This register specifies the interrupt status of the upper 32 interrupt sources.
IRQ_MASKSTATUS_L on page 65	0x28	This register specifies the interrupt mask status of the lower 32 interrupt sources.
IRQ_MASKSTATUS_H on page 67	0x2c	This register specifies the interrupt mask status of the upper 32 interrupt sources.
IRQ_FINALSTATUS_L on page 68	0x30	This register specifies the interrupt final status of the lower 32 interrupt sources.
IRQ_FINALSTATUS_H on page 70	0x34	This register specifies the interrupt final status of the upper 32 interrupt sources.
IRQ_VECTOR on page 72	0x38	This register specifies the interrupt vector of the highest pending interrupt.
IRQ_VECTOR_n (for n = 0; n <= ICT_IRQ_PLEVEL) on page 73	0x40 + 8*n	This register specifies the Interrupt Vector (Priority Level n).
FIQ_INTEN on page 74	0xc0	This register specifies the bits to enable the fast interrupt.

Table 5-5 Registers for Address Block: DW_apb_ictl_mem_map/DW_apb_ictl_addr_block1 (Continued)

Register	Offset	Description
FIQ_INTMASK on page 75	0xc4	This register specifies the bit to mask an interrupt.
FIQ_INTFORCE on page 76	0xc8	This register specifies the fast interrupt force bits.
FIQ_RAWSTATUS on page 77	0xcc	This register specifies the fast interrupt source raw status.
FIQ_STATUS on page 78	0xd0	This register specifies the fast interrupt status.
FIQ_FINALSTATUS on page 79	0xd4	This register specifies the fast interrupt final status.
IRQ_PLEVEL on page 80	0xd8	This register specifies the IRQ system priority level.
IRQ_INTERNAL_PLEVEL on page 81	0xdc	This register specifies the internal IRQ system priority level.
ICTL_VERSION_ID on page 83	0xe0	This register specifies the component version.
IRQ_PR_n (for n = 0; n <= ICT_IRQ_NUM-1) on page 84	0xe8 + 4*n	This register specifies the IRQ Individual Interrupt n Priority Level.
IRQ_VECTOR_DEFAULT on page 85	0x1e8	This register specifies the default interrupt vector register.

5.1.1 IRQ_INTEN_L

- **Name:** Interrupt Source Enable (Low) Register
- **Description:** This register specifies the interrupt enable bits for lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

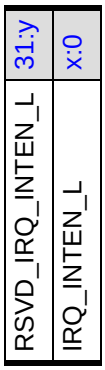


Table 5-6 Fields for Register: IRQ_INTEN_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_INTEN_L	R	IRQ_INTEN_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_IRQ_NUM
x:0	IRQ_INTEN_L	R/W	These bits specify the interrupt enable bits for lower 32 interrupt sources. A 1 in any bit position enables the corresponding interrupt. **Values:** ■ 0x0 (DISABLED): Interrupt disabled ■ 0x1 (ENABLED): Interrupt enabled **Value After Reset:** The corresponding bits of the ICT_IRQ_DFLT_EN configuration parameter. **Exists:** Always **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.2 IRQ_INTEN_H

- **Name:** Interrupt Source Enable (High) Register
- **Description:** This register enables the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** ICT_IRQ_NUM > 32

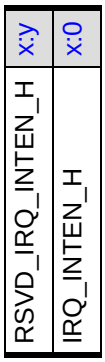


Table 5-7 Fields for Register: IRQ_INTEN_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_INTEN_H	R	IRQ_INTEN_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + (ICT_IRQ_NUM-32) - 1 **Range Variable[y]:** ICT_IRQ_NUM - 32
x:0	IRQ_INTEN_H	R/W	These bits specify the interrupt enable bit for upper 32 interrupt sources. A 1 in any bit position enables the corresponding interrupt. If there are less than 32 interrupt sources, this address location and register do not exist for a write or a read. By default, all bits enabled. **Value After Reset:** The corresponding bits of the ICT_IRQ_DFLT_EN configuration parameter. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Range Variable[x]:** (ICT_IRQ_NUM-32) - 1

5.1.3 IRQ_INTMASK_L

- **Name:** Interrupt Source Mask (Low) Register
- **Description:** This register masks the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

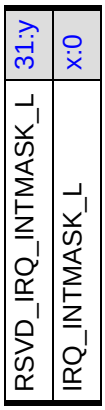


Table 5-8 Fields for Register: IRQ_INTMASK_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_INTMASK_L	R	IRQ_INTMASK_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_IRQ_NUM
x:0	IRQ_INTMASK_L	R/W	These bits specify the interrupt mask bits for the lower 32 interrupt sources. A 1 in any bit position masks (disables) the corresponding interrupt. By default, all bits are unmasked. **Values:** ■ 0x0 (UNMASK): Unmasks the interrupts ■ 0x1 (MASK): Masks the interrupt **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.4 IRQ_INTMASK_H

- **Name:** Interrupt Source Mask (High) Register
- **Description:** This register masks the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** ICT_IRQ_NUM > 32

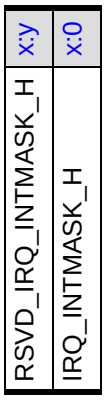


Table 5-9 Fields for Register: IRQ_INTMASK_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_INTMASK_H	R	IRQ_INTMASK_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + (ICT_IRQ_NUM-32) - 1 **Range Variable[y]:** ICT_IRQ_NUM - 32
x:0	IRQ_INTMASK_H	R/W	These bits specify the interrupt mask bits for the upper 32 interrupt sources. If there are less than 32 interrupt sources, this address location does not exist for a write or a read. By default, all bits are unmasked. **Value After Reset:** 0x0 **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Range Variable[x]:** (ICT_IRQ_NUM-32) - 1

5.1.5 IRQ_INTFORCE_L

- **Name:** Interrupt Force (Low) Register
- **Description:** This register specifies the interrupt force bits for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always

Table 5-10 Fields for Register: IRQ_INTFORCE_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_INTFORCE_L	R	IRQ_INTFORCE_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_IRQ_NUM

Table 5-10 Fields for Register: IRQ_INTFORCE_L (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_INTFORCE_L	R/W	These bits specify the interrupt force bits for the lower 32 interrupt sources. Each bit corresponds to one bit of the irq_intsrc input. The polarity of the bits in the register correspond to the polarity of the associated irq_intsrc input. If the interrupt input is configured to be active-high, the corresponding bit in the register is also active-high. **Values:** ■ 0x0 (ACTIVE_LOW): Active low polarity ■ 0x1 (ACTIVE_HIGH): Active high polarity **Value After Reset:** The reset state of the force bits is always inactive. It is derived by the configuration parameter ICT_IRQSRC_POL or ICT_FORCEREG_ACTIVE_HIGH. **Exists:** Always **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.6 IRQ_INTFORCE_H

- **Name:** Interrupt Force (High) Register
- **Description:** This register specifies the interrupt force bits for the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** ICT_IRQ_NUM > 32

Table 5-11 Fields for Register: IRQ_INTFORCE_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_INTFORCE_H	R	IRQ_INTFORCE_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + "(ICT_IRQ_NUM - 32)" - 1 **Range Variable[y]:** "(ICT_IRQ_NUM - 32)"

Table 5-11 Fields for Register: IRQ_INTFORCE_H (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_INTFORCE_H	R/W	These bits specify the interrupt force bits for the upper 32 interrupt sources. Each bit in this register corresponds to one bit of the irq_intsrc input. The polarity of the bits in the register correspond to the polarity of the associated irq_intsrc input. If the interrupt input is configured to be active-high, the corresponding bit in the register is also active-high. The reset state of the force bits is always inactive. **Value After Reset:** The reset state of the force bits is always inactive. It is derived by the configuration parameter ICT_IRQSRC_POL or ICT_FORCEREG_ACTIVE_HIGH. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Range Variable[x]:** "(ICT_IRQ_NUM-32)" - 1

5.1.7 IRQ_RAWSTATUS_L

- **Name:** Interrupt Raw Status (Low) Register
- **Description:** This register specifies the raw status of lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always

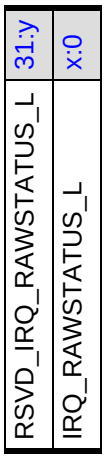


Table 5-12 Fields for Register: IRQ_RAWSTATUS_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_RAWSTATUS_L	R	IRQ_RAWSTATUS_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM

Table 5-12 Fields for Register: IRQ_RAWSTATUS_L (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_RAWSTATUS_L	R	These bits specify the actual interrupt source. These are the lower 32 interrupt sources. **Values:** ■ 0x0 (INACTIVE): Inactive Raw Interrupt Status ■ 0x1 (ACTIVE): Active Raw Interrupt Status **Value After Reset:** IRQ_RAWSTATUS_L - Dependent on setting of corresponding interrupt source bit. **Exists:** Always **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.8 IRQ_RAWSTATUS_H

- **Name:** Interrupt Raw Status (High) Register
- **Description:** This register specifies the raw status of the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** ICT_IRQ_NUM > 32

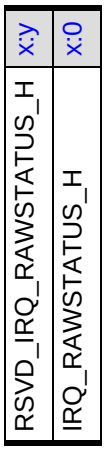


Table 5-13 Fields for Register: IRQ_RAWSTATUS_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_RAWSTATUS_H	R	IRQ_RAWSTATUS_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + "(ICT_IRQ_NUM - 32)" - 1 **Range Variable[y]:** "(ICT_IRQ_NUM - 32)"
x:0	IRQ_RAWSTATUS_H	R	These bits specify the actual interrupt source. These are the upper 32 interrupt sources. If there are less than 32 interrupt sources, this address location does not exist for a read. **Value After Reset:** IRQ_RAWSTATUS_H - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM-32)" - 1

5.1.9 IRQ_STATUS_L

- **Name:** Interrupt Status (Low) Register
- **Description:** This register specifies the interrupt Status of the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x20
- **Exists:** Always

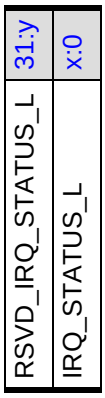


Table 5-14 Fields for Register: IRQ_STATUS_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_STATUS_L	R	IRQ_STATUS_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM
x:0	IRQ_STATUS_L	R	These bits specify the interrupt status after the forcing and interrupt enabling stage. These are the interrupt status signals for the lower 32 interrupt sources. **Values:** ■ 0x0 (INACTIVE): Inactive interrupt status ■ 0x1 (ACTIVE): Active interrupt status **Value After Reset:** IRQ_STATUS_L - Dependent on setting of corresponding interrupt source bit. **Exists:** Always **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.10 IRQ_STATUS_H

- **Name:** Interrupt Status (High) Register
- **Description:** This register specifies the interrupt status of the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x24
- **Exists:** ICT_IRQ_NUM > 32

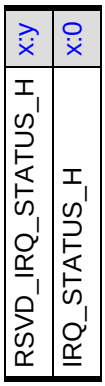


Table 5-15 Fields for Register: IRQ_STATUS_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_STATUS_H	R	IRQ_STATUS_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + "(ICT_IRQ_NUM - 32)" - 1 **Range Variable[y]:** "(ICT_IRQ_NUM - 32)"
x:0	IRQ_STATUS_H	R	These bits specify the interrupt status after the forcing and interrupt enabling stage. These are the interrupt status signals for the upper 32 interrupt sources. If there are less than 32 interrupt sources, this address location does not exist for a write or a read. **Value After Reset:** IRQ_STATUS_H - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM-32)" - 1

5.1.11 IRQ_MASKSTATUS_L

- **Name:** Interrupt Mask Status (Low) Register
- **Description:** This register specifies the interrupt mask status of the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** Always

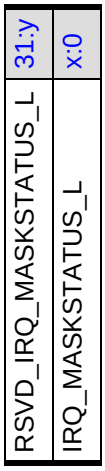


Table 5-16 Fields for Register: IRQ_MASKSTATUS_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_MASKSTATUS_L	R	IRQ_MASKSTATUS_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM

Table 5-16 Fields for Register: IRQ_MASKSTATUS_L (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_MASKSTATUS_L	R	These bits specify the interrupt status after the masking stage. These are the interrupt status signals for the lower 32 interrupt sources. **Values:** ■ 0x0 (INACTIVE): Inactive interrupt mask status ■ 0x1 (ACTIVE): Active interrupt mask status **Value After Reset:** IRQ_MASKSTATUS_L - Dependent on setting of corresponding interrupt source bit. **Exists:** Always **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.12 IRQ_MASKSTATUS_H

- **Name:** Interrupt Mask Status (High) Register
- **Description:** This register specifies the interrupt mask status of the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** ICT_IRQ_NUM > 32

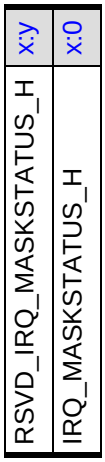


Table 5-17 Fields for Register: IRQ_MASKSTATUS_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_MASKSTATUS_H	R	IRQ_MASKSTATUS_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + "(ICT_IRQ_NUM - 32)" - 1 **Range Variable[y]:** "(ICT_IRQ_NUM - 32)"
x:0	IRQ_MASKSTATUS_H	R	These bits specify the interrupt status after the masking stage. These are the interrupt status signals for the upper 32 interrupt sources. If there are less than 32 interrupt sources, this address location does not exist for a write or a read. **Value After Reset:** IRQ_MASKSTATUS_H - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM-32)" - 1

5.1.13 IRQ_FINALSTATUS_L

- **Name:** Interrupt Final Status (Low) Register
- **Description:** This register specifies the interrupt final status of the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x30
- **Exists:** Always

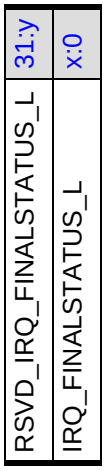


Table 5-18 Fields for Register: IRQ_FINALSTATUS_L

Bits	Name	Memory Access	Description
31:y	RSVD_IRQ_FINALSTATUS_L	R	IRQ_FINALSTATUS_L 31toICT_IRQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM

Table 5-18 Fields for Register: IRQ_FINALSTATUS_L (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_FINALSTATUS_L	R	These bits specify the interrupt status after the priority level filtering stage. These are the interrupt status signals for the lower 32 interrupt sources. If there is no priority interrupt scheme configured, then this location contains the same value as irq_maskstatus_l. **Values:** ■ 0x0 (INACTIVE): Inactive interrupt final status ■ 0x1 (ACTIVE): Active interrupt final status **Value After Reset:** IRQ_FINALSTATUS_L - Dependent on setting of corresponding interrupt source bit. **Exists:** Always **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM > 32) ? 32 : ICT_IRQ_NUM" - 1

5.1.14 IRQ_FINALSTATUS_H

- **Name:** Interrupt Final Status (High) Register
- **Description:** This register specifies the interrupt final status of the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x34
- **Exists:** ICT_IRQ_NUM > 32

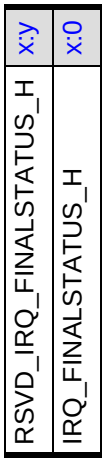


Table 5-19 Fields for Register: IRQ_FINALSTATUS_H

Bits	Name	Memory Access	Description
x:y	RSVD_IRQ_FINALSTATUS_H	R	IRQ_FINALSTATUS_H 31toICT_IRQ_NUM-32 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[x]:** "32-(ICT_IRQ_NUM-32)" + "(ICT_IRQ_NUM - 32)" - 1 **Range Variable[y]:** "(ICT_IRQ_NUM - 32)"

Table 5-19 Fields for Register: IRQ_FINALSTATUS_H (Continued)

Bits	Name	Memory Access	Description
x:0	IRQ_FINALSTATUS_H	R	These bits specify the interrupt status after the priority level filtering stage. These are the interrupt status signals for the upper 32 interrupt sources. If there are less than 32 interrupt sources, this location does not exist. If there is no priority interrupt scheme, this location contains the same value as irq_maskstatus_h. **Value After Reset:** IRQ_FINALSTATUS_H - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0 **Volatile:** true **Range Variable[x]:** "(ICT_IRQ_NUM-32)" - 1

5.1.15 IRQ_VECTOR

- **Name:** IRQ Vector Register
- **Description:** This register specifies the interrupt vector of the highest pending interrupt.
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always

Table 5-20 Fields for Register: IRQ_VECTOR

Bits	Name	Memory Access	Description
31:0	IRQ_VECTOR	R	This location can be read when an interrupt occurs, and the provided vectored interrupts are supported. This register field returns one of the 16 vectors. The returned vector corresponds to the highest priority level interrupt. When no final status interrupts are active, the read value depends on the ICT_HAS_DEFAULT_VECTOR configuration parameter: ■ ICT_HAS_DEFAULT_VECTOR = 1: Contains the vector value in the irq_vector_default register location. ■ ICT_HAS_DEFAULT_VECTOR = 0: Contains the vector value corresponding to the priority programmed into the irq_plevel register. This register returns 0 when ICT_HAS_VECTOR = 0. This register is read coherent, allowing the register to be read, regardless of the data bus width. **Value After Reset:** If ICT_HAS_VECTOR = 0, resets to 0 else, if ICT_HAS_DEFAULT_VECTOR = 1, resets to ICT_VECTOR_DEFAULT or resets to ICT_VECTOR_n; where n = ICT_IRQ_PLEVEL. **Exists:** Always **Volatile:** true

5.1.16 IRQ_VECTOR_n (for n = 0; n <= ICT_IRQ_PLEVEL)

- **Name:** Interrupt Vector (Priority Level 0) Register
- **Description:** This register specifies the Interrupt Vector (Priority Level n).
- **Size:** 32 bits
- **Offset:** 0x40 + 8*n
- **Exists:** ((ICT_HAS_PFLT==1)?ICT_HAS_VECTOR_USER:0)==1

Table 5-21 Fields for Register: IRQ_VECTOR_n (for n = 0; n <= ICT_IRQ_PLEVEL)

Bits	Name	Memory Access	Description
31:0	IRQ_VECTOR_n	(ICT_HC_VECTOR_n==0) ? read-write : read-only	These bits specify the interrupt vector for priority level 0. This register does not exist when ICT_HAS_VECTOR = 0. **Value After Reset:** If ICT_HC_VECTOR_n is set to 1, this is a read-only register and the interrupt vector is set by ICT_VECTOR_n. If ICT_HC_VECTOR_n is set to 0, this is a read/write register used to program the interrupt vector; the reset value is determined by ICT_VECTOR_n. **Exists:** Always

5.1.17 FIQ_INTEN

- **Name:** Fast Interrupt Enable Register
- **Description:** This register specifies the bits to enable the fast interrupt.
- **Size:** 32 bits
- **Offset:** 0xc0
- **Exists:** ICT_HAS_FIQ==1

Table 5-22 Fields for Register: FIQ_INTEN

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_INTEN	R	FIQ_INTEN 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_INTEN	R/W	These bits specify the fast interrupt enable bits. A 1 in any bit position enables the corresponding interrupt. **Values:** ■ 0x0 (DISABLED): Fast interrupt disabled ■ 0x1 (ENABLED): Fast interrupt enabled **Value After Reset:** ICT_FIQ_DFLT_EN **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.18 FIQ_INTMASK

- **Name:** Fast Interrupt Mask Register
- **Description:** This register specifies the bit to mask an interrupt.
- **Size:** 32 bits
- **Offset:** 0xc4
- **Exists:** ICT_HAS_FIQ==1

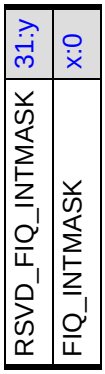


Table 5-23 Fields for Register: FIQ_INTMASK

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_INTMASK	R	FIQ_INTMASK 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_INTMASK	R/W	These bits specify the fast interrupt mask bits. A 1 in any bit position masks the corresponding interrupt. **Values:** ■ 0x0 (UNMASK): Unmasks the interrupts ■ 0x1 (MASK): Masks the interrupts **Value After Reset:** 0x0 **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.19 FIQ_INTFORCE

- **Name:** Fast Interrupt Force Register
- **Description:** This register specifies the fast interrupt force bits.
- **Size:** 32 bits
- **Offset:** 0xc8
- **Exists:** ICT_HAS_FIQ==1

Table 5-24 Fields for Register: FIQ_INTFORCE

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_INTFORCE	R	FIQ_INTFORCE 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_INTFORCE	R/W	These bits specify the fast interrupt force bits. Each bit in this register corresponds to one bit of the irq_intsrc input. The polarity of the bits in the register corresponds to the polarity of the associated fiq_intsrc input. If the interrupt input is configured to be active-high, the corresponding bit in the register is also active-high. **Values:** ■ 0x0 (ACTIVE_LOW): Active low interrupts ■ 0x1 (ACTIVE_HIGH): Active high interrupts **Value After Reset:** The reset state of the force bits is always inactive. It is derived by the configuration parameter ICT_FORCEREG_ACTIVE_HIGH. **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.20 FIQ_RAWSTATUS

- **Name:** Fast Interrupt Source Raw Status Register
- **Description:** This register specifies the fast interrupt source raw status.
- **Size:** 32 bits
- **Offset:** 0xcc
- **Exists:** ICT_HAS_FIQ==1

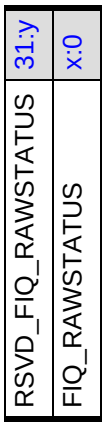


Table 5-25 Fields for Register: FIQ_RAWSTATUS

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_RAWSTATUS	R	FIQ_RAWSTATUS 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_RAWSTATUS	R	These bits specify the fast interrupt source raw input status. **Values:** ■ 0x0 (INACTIVE): Inactive raw interrupt status ■ 0x1 (ACTIVE): Active raw interrupt status **Value After Reset:** FIQ_RAWSTATUS - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.21 FIQ_STATUS

- **Name:** Fast Interrupt Status Register
- **Description:** This register specifies the fast interrupt status.
- **Size:** 32 bits
- **Offset:** 0xd0
- **Exists:** ICT_HAS_FIQ==1

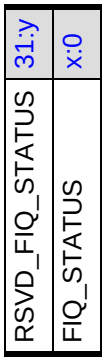


Table 5-26 Fields for Register: FIQ_STATUS

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_STATUS	R	FIQ_STATUS 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_STATUS	R	These bits specify the fast interrupt status after the forcing and interrupt enabling stage. **Values:** ■ 0x0 (INACTIVE): Fast interrupt status is inactive ■ 0x1 (ACTIVE): Fast interrupt status is active **Value After Reset:** FIQ_STATUS - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Volatile:** true **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.22 FIQ_FINALSTATUS

- **Name:** Fast Interrupt Final Status Register
- **Description:** This register specifies the fast interrupt final status.
- **Size:** 32 bits
- **Offset:** 0xd4
- **Exists:** ICT_HAS_FIQ==1

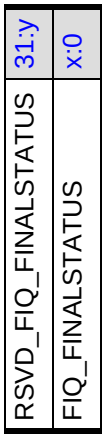


Table 5-27 Fields for Register: FIQ_FINALSTATUS

Bits	Name	Memory Access	Description
31:y	RSVD_FIQ_FINALSTATUS	R	FIQ_FINALSTATUS 31toICT_FIQ_NUM Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_FIQ_NUM
x:0	FIQ_FINALSTATUS	R	These bits specify the fast interrupt status after the masking stage. **Values:** ■ 0x0 (INACTIVE): Fast interrupt final status is inactive ■ 0x1 (ACTIVE): Fast interrupt final status is active **Value After Reset:** FIQ_FINALSTATUS - Dependent on setting of corresponding interrupt source bit. **Exists:** (ICT_HAS_FIQ==1) ? 1 : 0 **Volatile:** true **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.23 IRQ_PLEVEL

- **Name:** IRQ System Priority Level Register
- **Description:** This register specifies the IRQ system priority level.
- **Size:** 32 bits
- **Offset:** 0xd8
- **Exists:** Always

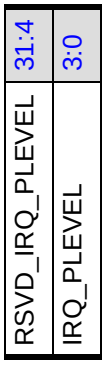


Table 5-28 Fields for Register: IRQ_PLEVEL

Bits	Name	Memory Access	Description
31:4	RSVD_IRQ_PLEVEL	R	IRQ_PLEVEL 31to4 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always
3:0	IRQ_PLEVEL	R/W	These bits specify the interrupt controller system priority level for normal interrupt sources. The default state can be configured so that after reset, the interrupt controller accepts only interrupts that are enabled and have a priority the same or greater than the system level priority setting. **Value After Reset:** ICT_IRQ_PLEVEL **Exists:** Always

5.1.24 IRQ_INTERNAL_PLEVEL

- **Name:** Internal IRQ System Priority Level Register
- **Description:** This register specifies the internal IRQ system priority level.
- **Size:** 32 bits
- **Offset:** 0xdc
- **Exists:** ICT_ADD_VECTOR_PORT==1

Table 5-29 Fields for Register: IRQ_INTERNAL_PLEVEL

Bits	Name	Memory Access	Description
31:5	RSVD_IRQ_INTERNAL_PLEVEL	R	IRQ_INTERNAL_PLEVEL 31to4 Reserved bits - Read Only. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-29 Fields for Register: IRQ_INTERNAL_PLEVEL (Continued)

Bits	Name	Memory Access	Description
4:0	IRQ_INTERNAL_PLEVEL	R/W	Internal interrupt system priority level register. If ICT_ADD_VECTOR_PORT is set to true, this register reflects the priority level currently being applied by the DW_apb_ictl. In the period between the completion of vector port handshaking and a write to irq_internal_plevel or arrival of a higher priority IRQ, it is set to one priority level greater than the priority level currently being handled by the processor. At all other times, it reflects the value of irq_plevel. This register should only be written to at the end of an interrupt service routine in order to reset the priority level to that of irq_plevel. Reads to this register return the current priority level. Writes to this register reset its value to irq_plevel; bus write data is ignored. **Value After Reset:** ICT_IRQ_PLEVEL **Exists:** (ICT_ADD_VECTOR_PORT==1) ? 1 : 0 **Volatile:** true

5.1.25 ICTL_VERSION_ID

- **Name:** Component Version Register
- **Description:** This register specifies the component version.
- **Size:** 32 bits
- **Offset:** 0xe0
- **Exists:** Always

Table 5-30 Fields for Register: ICTL_VERSION_ID

Bits	Name	Memory Access	Description
31:0	ICTL_VERSION_ID	R	These bits specify values of the ICTL_VERSION_ID. **Value After Reset:** ICTL_VERSION_ID **Exists:** Always

5.1.26 IRQ_PR_n (for n = 0; n <= ICT_IRQ_NUM-1)

- **Name:** IRQ Individual Interrupt n Priority Level Register
- **Description:** This register specifies the IRQ Individual Interrupt n Priority Level.
- **Size:** 32 bits
- **Offset:** 0xe8 + 4*n
- **Exists:** (ICT_READ_PRIORITY==1 && ICT_IRQ_NUM-1 >= 0) ? 1 : 0

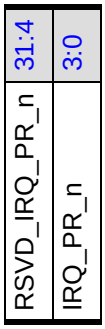


Table 5-31 Fields for Register: IRQ_PR_n (for n = 0; n <= ICT_IRQ_NUM-1)

Bits	Name	Memory Access	Description
31:4	RSVD_IRQ_PR_n	R	IRQ_PR_n 31to4 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
3:0	IRQ_PR_n	(ICT_HC_PRIORITIES==0) ? read-write : read-only	These bits specify the individual interrupt priority level. The number of Individual Interrupt Priority Level registers is from 0 to ICT_IRQ_NUM-1. The value of the register must be an integer from 0x0 to 0xf. Following are the Read/Write Values: ■ DNE: If ICT_READ_PRIORITY=0, then all irq_pN_priority registers do not exist. ■ R: If ICT_READ_PRIORITY=1 and ICT_HC_PRIORITIES=1, then interrupt 0 priority register is read-only R/W: If ICT_READ_PRIORITY=1 and ICT_HC_PRIORITIES=0, then interrupt 0 priority register is read/write **Value After Reset:** ICT_ISRC_PLEVEL_n **Exists:** Always

5.1.27 IRQ_VECTOR_DEFAULT

- **Name:** Default Interrupt Vector Register
- **Description:** This register specifies the default interrupt vector register.
- **Size:** 32 bits
- **Offset:** 0x1e8
- **Exists:** ICT_HAS_DEFAULT_VECTOR==1

Table 5-32 Fields for Register: IRQ_VECTOR_DEFAULT

Bits	Name	Memory Access	Description
31:0	IRQ_VECTOR_DEFAULT	(ICT_HC_VECTOR_DEFAULT==0) ? read-write : read-only	These bits specify the default interrupt vector. The value in this register is returned on a read to the irq_vector register when no interrupts are active. Final status is used to decode an active interrupt. This register can be read and, if not hardcoded, can be reconfigured by a master. This register does not exist when ICT_HAS_DEFAULT_VECTOR = 0. ■ If ICT_HC_VECTOR_DEFAULT is set to 1, this is a read-only register and the interrupt vector is set by ICT_VECTOR_DEFAULT. ■ If ICT_HC_VECTOR_DEFAULT is set to 0, this is a read/write register used to program the default interrupt vector; the reset value is determined by ICT_VECTOR_DEFAULT. **Value After Reset:** If ICT_HC_VECTOR_DEFAULT is set to 1, this is a read-only register and the interrupt vector is set by ICT_VECTOR_DEFAULT. IfICT_HC_VECTOR_DEFAULTis set to 0, this is a read/write register used to program the default interrupt vector; the reset value is determined by ICT_VECTOR_DEFAULT. **Exists:** ICT_HAS_DEFAULT_VECTOR==1

6 Programming the DW_apb_ictl

This section describes the some of the programmable features of the DW_apb_ictl.

6.1 Programming Considerations

The APB data bus width is independently configurable from the width of registers internal to the DW_apb_ictl. If the APB data bus width is narrower than the width of an internal register, multiple reads and writes are necessary to access the complete register.

6.2 Reading/Writing Registers Wider than APB_DATA_WIDTH

Attention Because interrupt processing is combinational, care must be taken when reading and writing registers that are wider than APB_DATA_WIDTH.

Values of status registers can change if new interrupts arrive between reading slices of the register. Configuration registers should not be programmed while interrupts are enabled. Therefore, all IRQ interrupts should be disabled using the irq_inten register prior to programming the vector registers.

6.3 Initialization

A normal initialization sequence is as follows:

1. Disable all interrupts by writing to the irq_inten and fiq_inten. You can also configure the DW_apb_ictl so that this is the reset state using the ICT_IRQ_DFLT_EN and ICT_FIQ_DFLT_EN parameters.
2. Initialize peripheral devices that could generate interrupts.
3. Program the irq_vector, irq_plevel, irq_intmask, and fiq_intmask registers as appropriate.
4. Enable interrupts.

6.4 Interrupt Service

Without vectored interrupts, a normal interrupt servicing sequence is as follows:

1. Poll the interrupt status register (irq_finalstatus, irq_maskstatus, or fiq_finalstatus, as appropriate) to determine which interrupt source caused the interrupt.
2. Service the interrupt.
3. Optionally read the irq_status or fiq_status register to see if other currently masked interrupts are pending; service these as required.

With vectored interrupt support, a normal interrupt servicing sequence is as follows:

1. Read the irq_vector register to get the address of the service routine.
2. Service routine reads irq_finalstatus to see which interrupt sources caused the interrupt.
3. Service the interrupt.
4. Optionally read the irq_status to see if other interrupts are pending; service these as required.

7 Verification

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

Note The DW_apb_ictl 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, and AMBA 4 AXI Installation Guide*

7.1 Overview of Vera Tests

The DW_apb_ictl can be configured as an APB slave. The DW_apb_ictl verification testbench performs the following set of tests, which exhaustively verify the functionality of the component and have also achieved maximum RTL code coverage.

Note All tests use the APB Interface to program memory mapped registers dynamically during tests.

7.1.1 Reset

The objective of this test is to ensure that all configurable registers are reset to the correct values defined for the DW_apb_ictl under test.

7.1.2 Slave Interface

Accesses to registers that do not contain configured interrupt or fast interrupt information return zeros. Registers are valid for only the number of active bits contained. For example, bytes 0 and 1 are valid for an access to irq_*register name* when ICT_IRQ_NUM is 9 or greater. If ICT_IRQ_NUM is 8 or less, then only byte 0 is valid.

7.1.3 FIQ

This sequence of tests verifies the operation of the FIQ generation circuitry and is repeated for active-low and active-high input FIQ source levels (fiq_intrsrc). These tests are independent of the active level of the fast interrupt source inputs and are executed on each fiq_intrsrc bit to verify all paths in the FIQ generation circuitry. The active level of fiq_intsrc*N* and fiq_force*N* is determined by the ICT_FIQ_INTSRC_POL_n configuration parameter.

7.1.4 IRQ

This sequence of tests verifies the operation of the IRQ generation circuitry and is repeated for active-low and active-high input IRQ source levels (irq_intrsrc). These tests are independent of the active level of the interrupt source inputs and are executed on each irq_intrsrc bit to verify all paths in the IRQ generation circuitry. The active level of irq_intsrc*N* and irq_force*N* is determined by the ICT_IRQ_INTSRC_POL_n configuration parameter.

7.1.5 Priority Controller

This sequence of tests verifies the operation of the priority controller and vector generation. It verifies the generation of irq_intpmask bus. This test is run when the DW_apb_ictl is configured to have a priority controller. The vector generation sections of this test can be run only when the ICT_HAS_VECTOR configuration parameter is set to 1. These tests are independent of the active level of the interrupt source inputs.

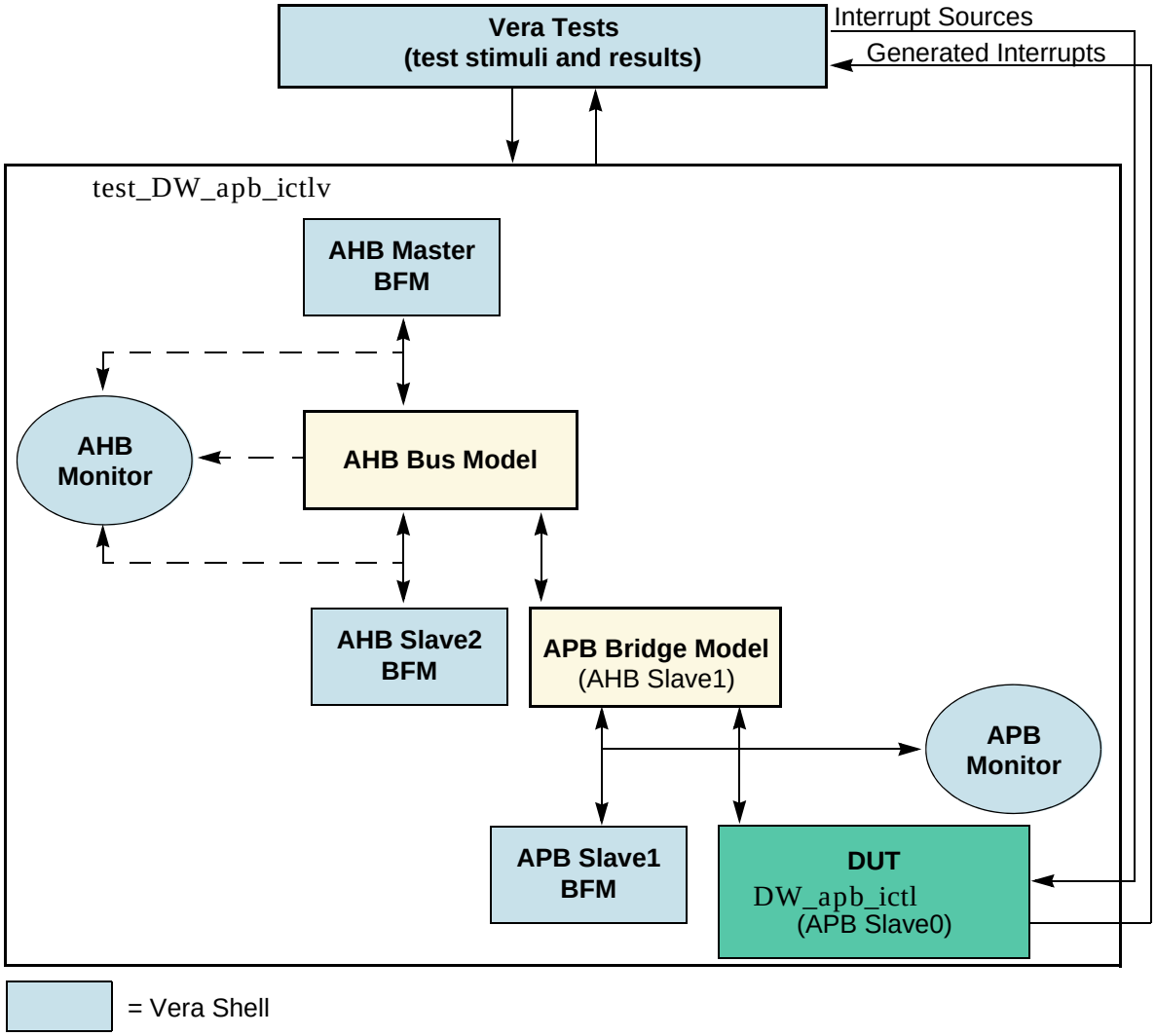
7.1.6 Dynamic Reconfiguration

The tests associated with Priority Controller and IRQ tests verify the operation of the priority controller and of the enable, mask, and status registers. The user must also be able to dynamically change the priority and vector register values without affecting the operation of the DW_apb_ictl. This test is run only when the DW_apb_ictl is configured to have a priority controller with vector generation. These tests are independent of the active level of the interrupt source inputs.

7.2 Overview of DW_apb_ictl Testbench

As illustrated in Figure 7-1, the Verilog DW_apb_ictl testbench includes an instantiation of the design under test (DUT), AHB and APB Bridge bus models, and a Vera shell.

Figure 7-1 DW_apb_ictl Testbench



The Vera shell consists of an AHB Master bus functional model (BFM), one AHB Slave BFM, an AHB Monitor, APB Slave BFMs, an APB Monitor, test stimuli, BFM configuration, and test results. The AHB Monitor monitors activity from the AHB Master and Slave BFMs; the APB Monitor oversees activity from the APB Slave BFMs.

The testbench tests for all possible user configurations chosen in the Specify Configuration task of coreConsultant. The testbench also tests that the component is AMBA-compliant and includes a self-checking mechanism.

7.3 Running Simulations from the Command Line

To run simulations from a UNIX command line, a simulation model must be generated through the coreConsultant GUI. In addition, all tests and test options must be configured in the Verification tab of the GUI. Then, simulations can be run as follows:

- To run all tests selected in the GUI, change your working directory to DW_apb_ictl/sim and then execute the following command:

```
runtest.sh
```

- To run single tests, change the working directory to DW_apb_ictl/sim and run the following:

```
runtest --simulator selected_simulator --test test_name
```

The *selected_simulator* is the one chosen in the GUI; it does not work if not configured in the GUI. The *test_name* is the name of the selected test and the sub directory where the test is located. For example, to run the simple register write/read test using VCS, run the following:

```
runtest --simulator vcs --test test_reg_wr_rd
```

The results of running tests through the command line are available only in the test.log file in each test directory.

7.4 Command Line Output Files

The runtest.log file is generated in *workspace*/sim/ only as a result of running simulations from coreConsultant or coreAssembler. The runtest.log file provides a pass/fail result for the particular simulation, as well as some detailed information. The test.log file located in *workspace*/sim/test_*name* is generated when tests are run from the GUI or command line, and provides more detail on each specific test simulation, in addition to the pass/fail status. The waveforms are also written to this directory, when enabled.

To enable waveform generation from the command line, the switch DumpEnabled must be set as follows:

```
runtest --simulator vcs --DumpEnabled 1 --test test_reg_wr_rd
```

If the simulation results match expected results, the simulation completes successfully and the simulation status in the test.log file is PASSED. If the simulation results do not match expected results, the simulation terminates and the simulation status in the test.log file is FAILED.

8 Integration Considerations

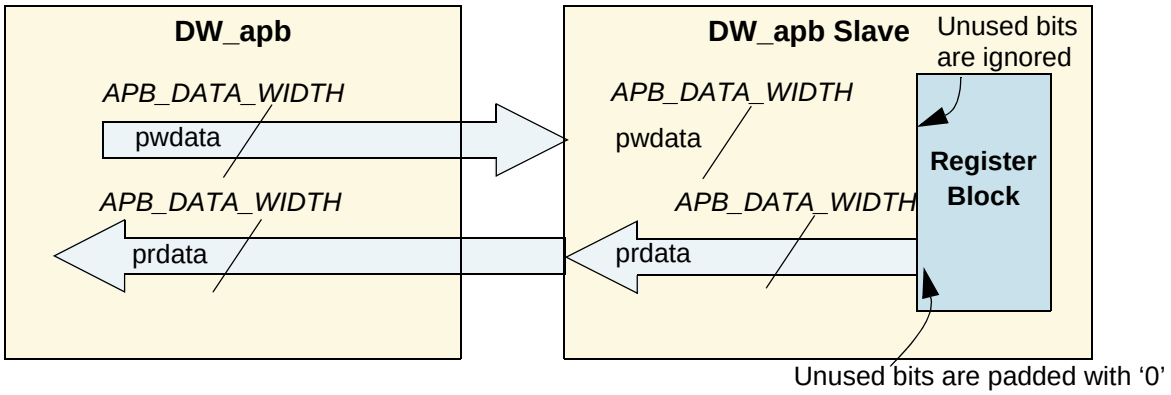
After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment. The following sections discuss general integration considerations.

8.1 Bus Interface

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

Figure 8-1 shows the read / write buses between the DW_apb and the APB slave.

Figure 8-1 Relationship Between DW_apb and Slave Data Widths



For more information about the APB interface, refer to "Integration Considerations", "Integration Considerations" on page 93.

8.2 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.

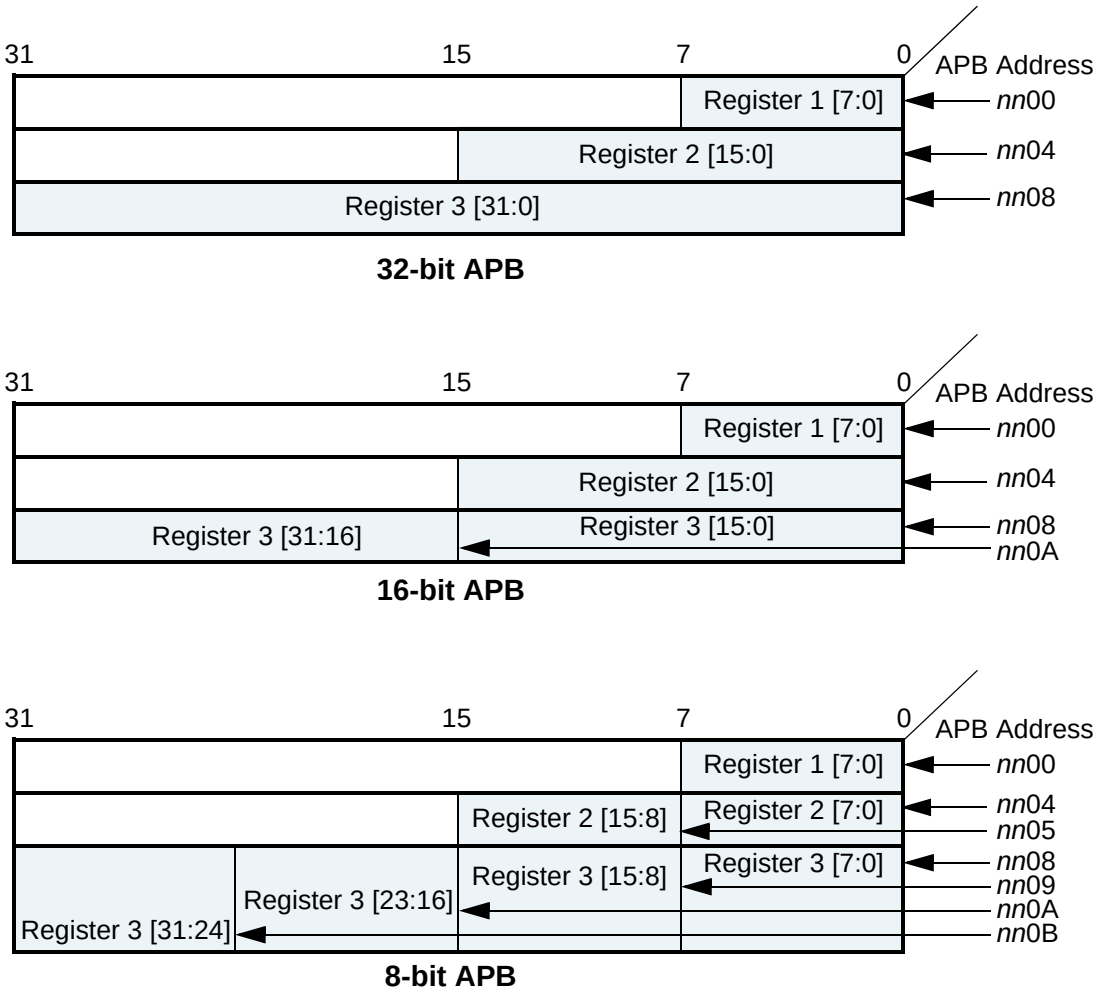
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this 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.

8.2.1 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 8-2 Read/Write Locations for Different APB Bus Data Widths



8.2.2 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.

8.2.3 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.

8.2.4 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

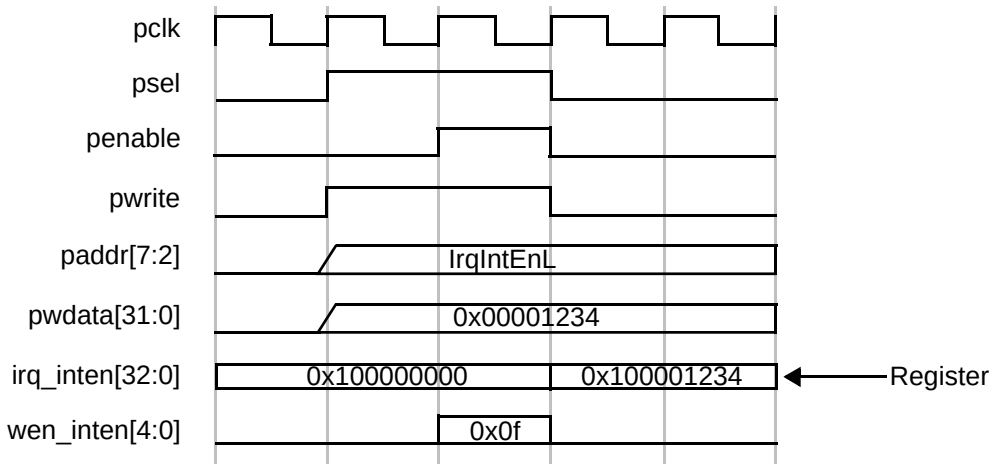
   In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read / write the lower byte and the second transaction the second byte, and so on.

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

8.3 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 8-3 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.

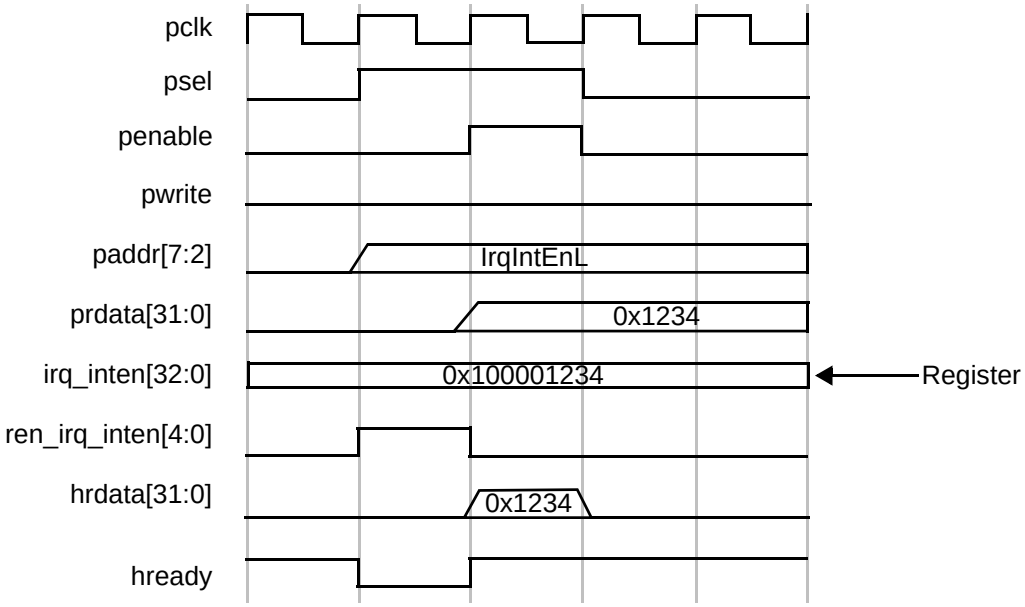
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

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.

8.4 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 8-4 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.

8.5 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>
```

8.6 Coherency

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

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

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

Throughout this appendix the following terms are used:

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

8.6.1 Writing Coherently

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

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

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

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

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

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

Table 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

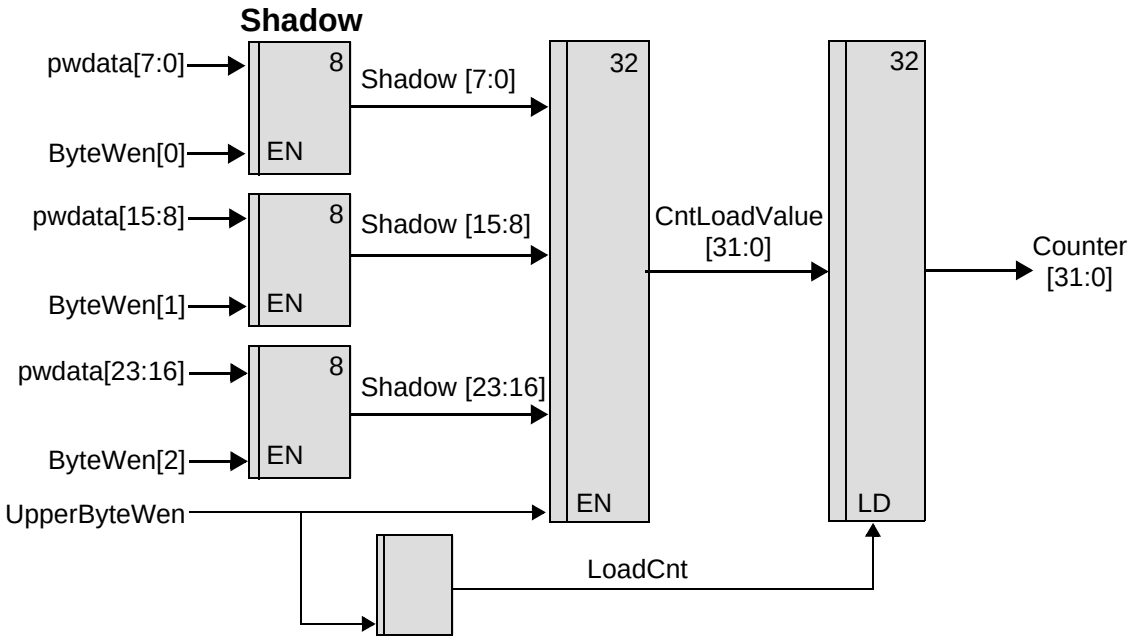
There are three relationship cases to be considered for the processor and peripheral clocks:

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

8.6.1.1 Identical Clocks

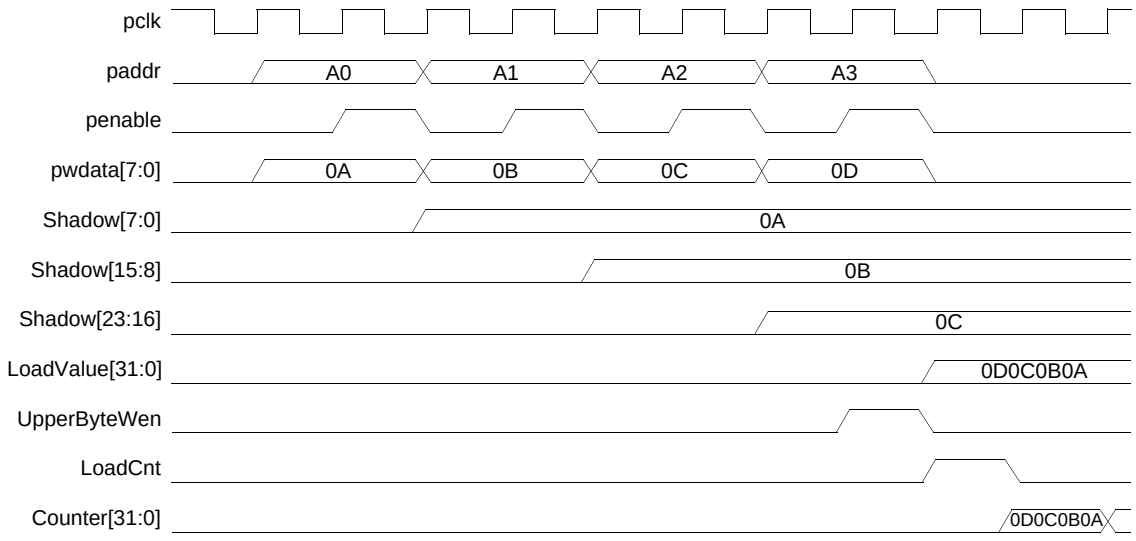
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 8-5 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 8-6 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.

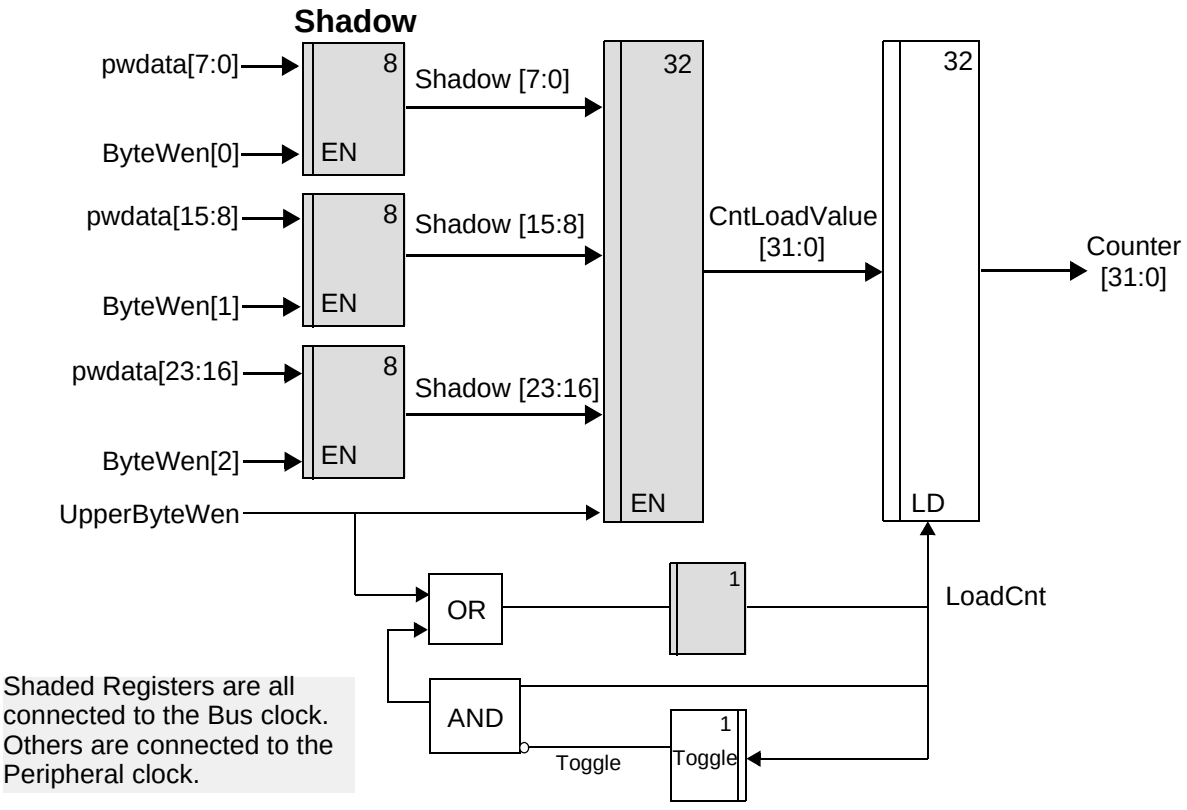
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this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

8.6.1.2 Synchronous Clocks

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

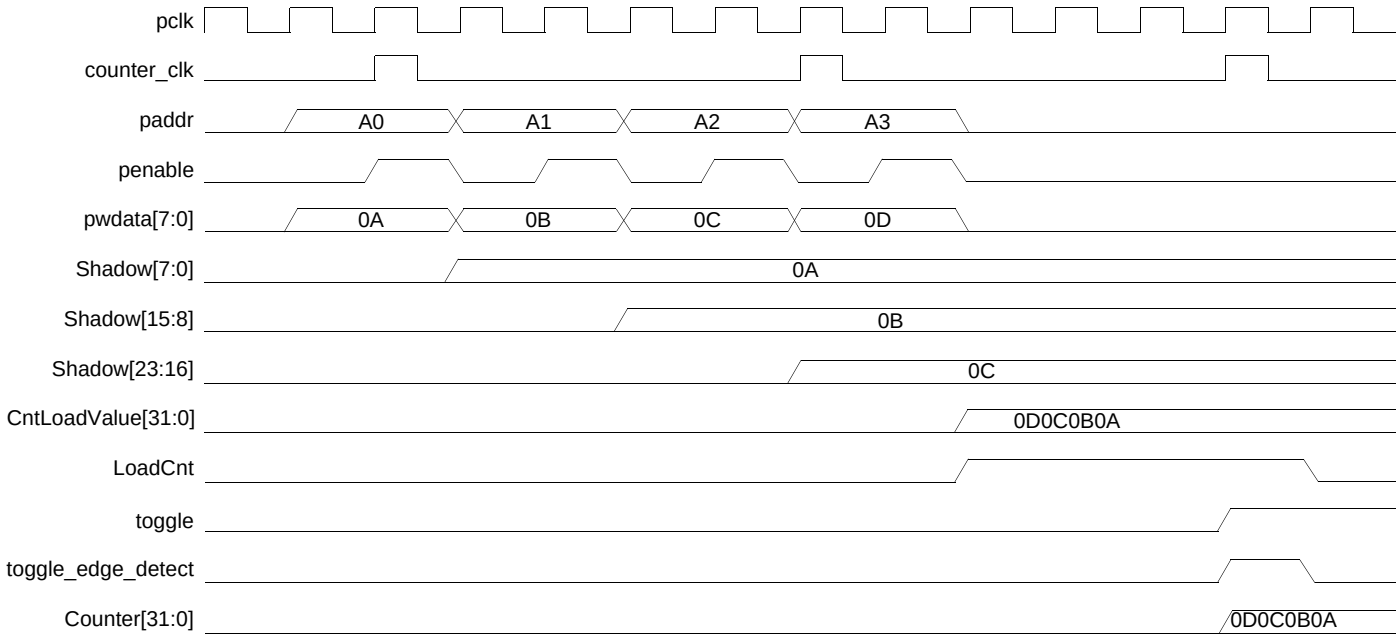
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 8-7 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.

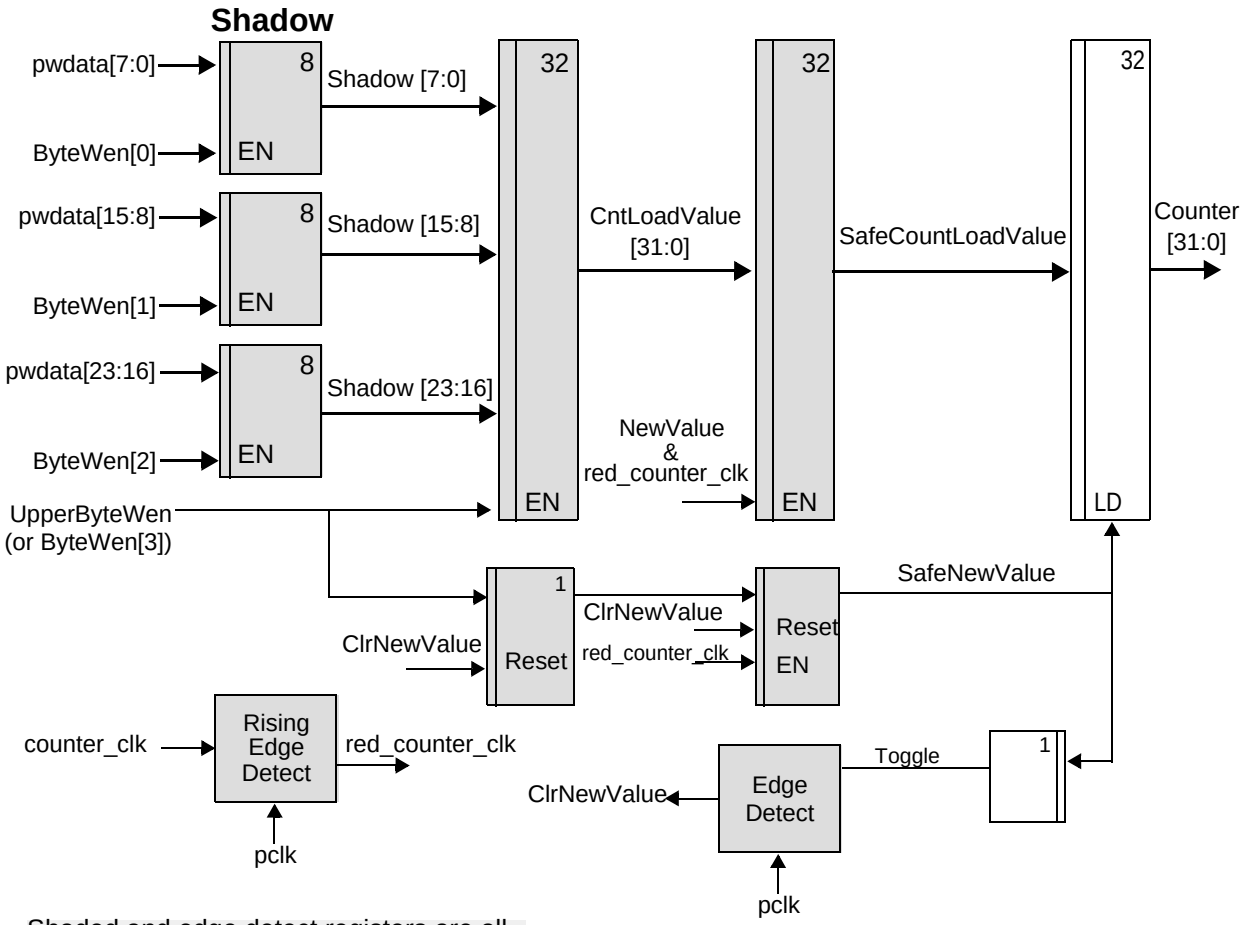
Figure 8-8 Coherent Loading – Synchronous Clocks



8.6.1.3 Asynchronous Clocks

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

Figure 8-9 Coherent Loading – Asynchronous Clocks



Shaded and edge detect registers are all connected to the Bus clock. Others are connected to the Peripheral clock.

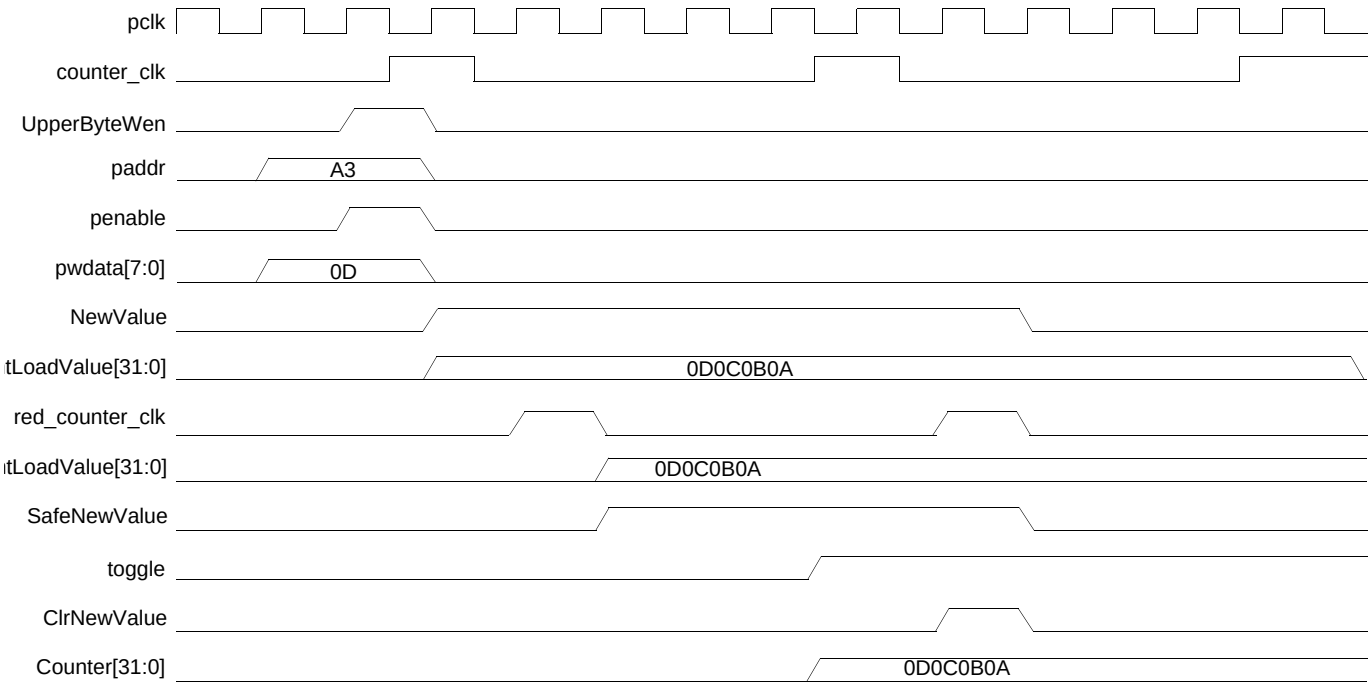
When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock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

Every time there is a rising edge of the peripheral clock detected, the CntLoadValue is transferred into a SafeCntLoadValue. This value is used to transfer the load value across the clock domains. The SafeCntLoadValue only changes a number of bus clock cycles after the peripheral clock edge changes. A

counter running on the peripheral clock is able to use this value safely. It could be up to two peripheral clock periods before the value is loaded into the counter. Along with this loaded value, there also is a single bit transferred that is used to qualify the loading of the value into the counter.

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

Figure 8-10 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.

8.6.2 Reading Coherently

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

Table 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

8.6.2.1 Synchronous Clocks

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

Figure 8-11 Coherent Registering – Synchronous Clocks

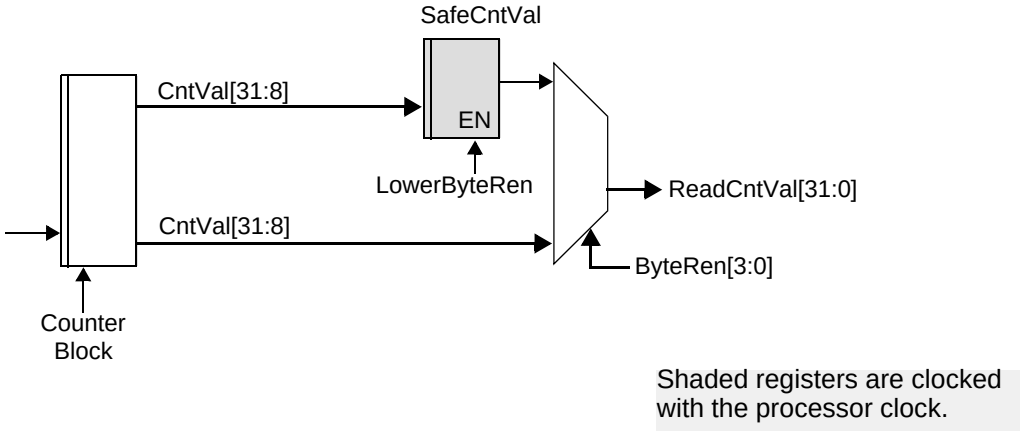
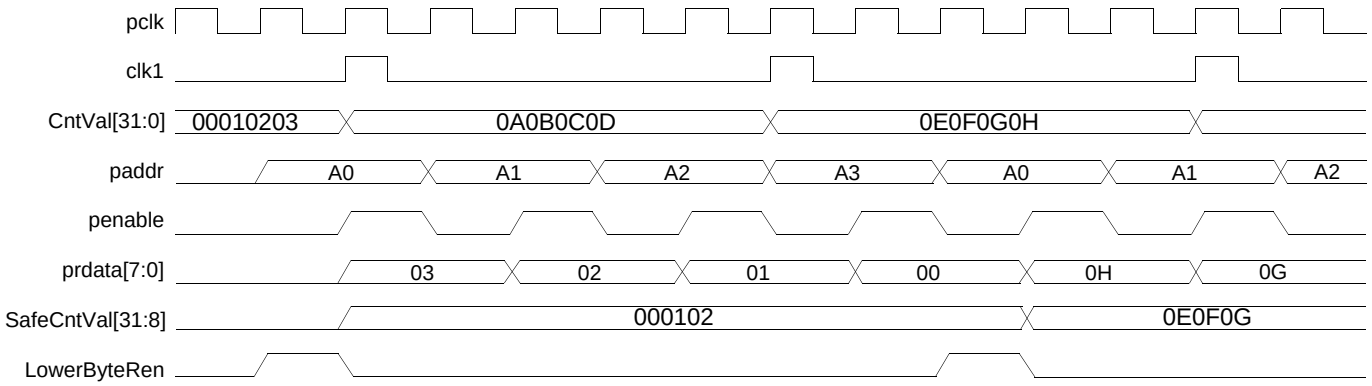


Figure 8-12 Coherent Registering – Synchronous Clocks



8.6.2.2 Asynchronous Clocks

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

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

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

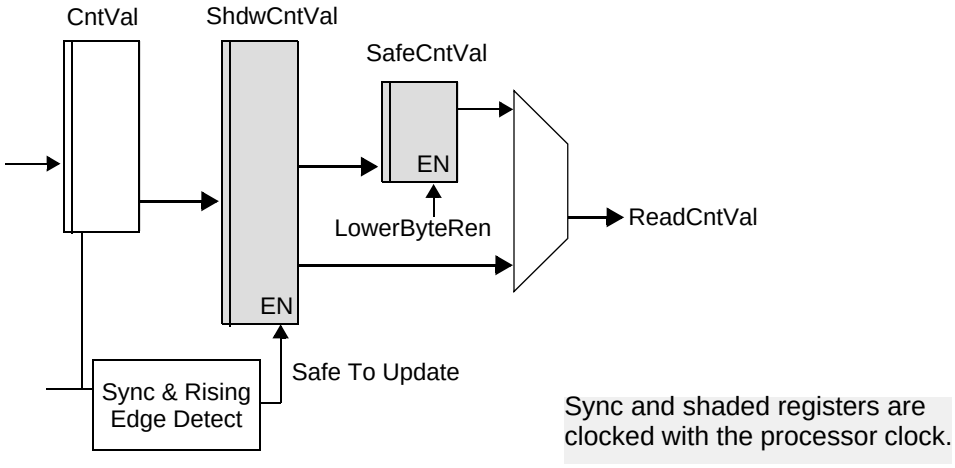
Note You must read LSB to MSB when the bus width is narrower than the counter width.

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

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

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

Figure 8-13 Coherency and Shadow Registering – Asynchronous Clocks



Performance

8.7 Performance

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

8.7.1 Power Consumption, Frequency, and Area Results

Table 8-3 provides information about the synthesis results (power consumption, frequency, and area) of the DW_apb_ictl using the industry standard 28nm technology library and how it affects performance.

Table 8-3 Power Consumption, Frequency, and Area Results for DW_apb_ictl Using 28nm Technology Library

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Default Configuration	hclk: 300 MHz	1550 gates	26.5 nW	4.661 uW
Minimum Configuration: APB_DATA_WIDTH 8 ICT_IRQ_NUM 2 ICT_HAS_FIQ 0 ICT_HAS_PFLT 0	hclk: 300 MHz	196 gates	3.07 nW	0.52 uW

Configuration	Operating Frequency	Gate Count	Static Power Consumption	Dynamic Power Consumption
Maximum Configuration: APB_DATA_WIDTH 32 ICT_IRQ_NUM 64 ICT_HAS_FIQ 1 ICT_FIQ_NUM 8 ICT_HAS_PFLT 1 ICT_HAS_VECTOR_USER 1 ICT_IRQ_PLEVEL 5	hclk: 300 MHz	9302 gates	156 nW	24.998 uW

A
Synchronizer Methods

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

This appendix contains the following sections:

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.1 Synchronizers Used in DW_apb_ictl

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_ictl are listed and cross referenced to the synchronizer type in Table A-1. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table A-1 Synchronizer used in DW_apb_ictl

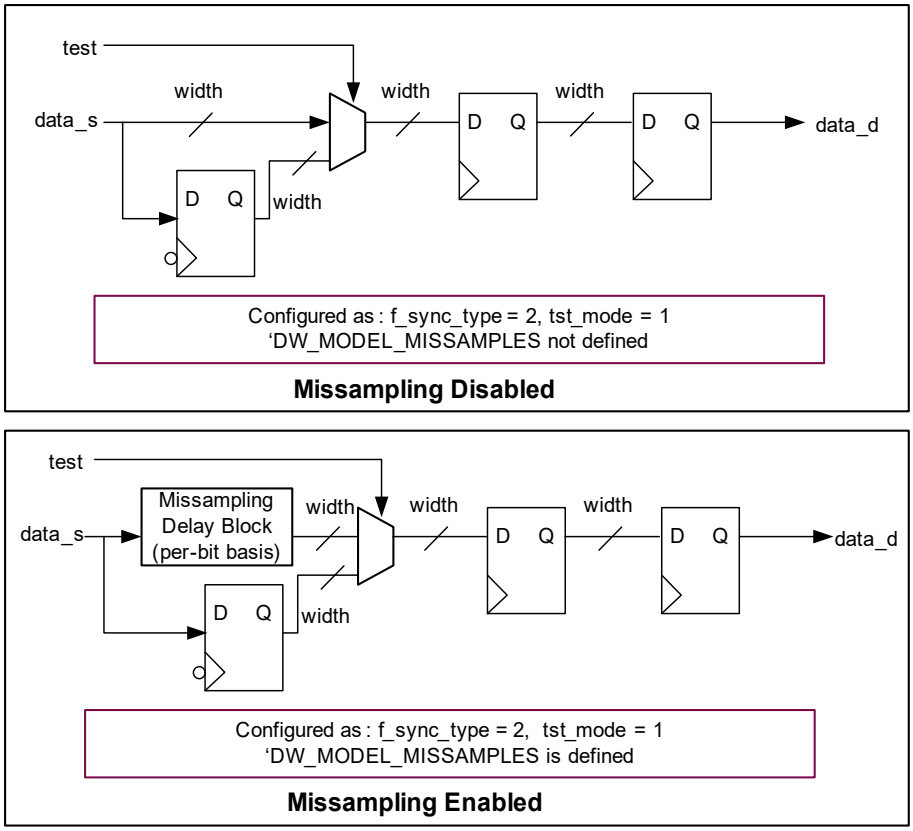
Synchronizer Module File	Synchronizer Type and Number
DW_apb_ictl_bcm21.v	Synchronizer 1: Simple Multiple Register Synchronizer

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 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_ictl)

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
HADDR_REGFILE_SLICE_LHS	9
HADDR_WIDTH	32
ICT_HAS_VECTOR	(((ICT_HAS_PFLT == 1)) ? ICT_HAS_VECTOR_USER : 0)
ICTL_VERSION_ID	32'h3230392a

C
Glossary

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

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

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

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

Index

Z