SYNOPSYS®

DesignWare® DW_ahb_ictl Databook

DW_ahb_ictl* – *Product Code

2.15a
December 2020

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 2.04b onward.

Version	Date	Description
2.15a	December 2020	**Updated:** ■ Version number changed for 2020.12a release ■ "Verification" on page 101 ■ "Performance" on page 119 ■ "Parameter Descriptions" on page 33, "Register Descriptions" on page 55, "Signal Descriptions" on page 45, and "Internal Parameter Descriptions" on page 121 are auto-extracted with change bars from the RTL **Renamed:** ■ Appendix Synchronizer Method is renamed to "Basic Core Module (BCM) Library" on page 123 **Removed:** ■ Index chapter
2.14a	July 2018	**Updated:** ■ Version number changed for 2018.07a release ■ "Performance" on page 119 ■ "Parameter Descriptions" on page 33, "Register Descriptions" on page 55, "Signal Descriptions" on page 45, and "Internal Parameter Descriptions" on page 121 are auto-extracted with change bars from the RTL ■ Added support for configurable synchronization depth through parameter ICT_ADD_VECTOR_PORT_SYNC_DEPTH **Removed:** ■ Chapter 2, "Building and Verifying a Component or Subsystem" and added the contents in the newly created user guide.

(Continued)

Version	Date	Description
2.13a	October 2016	■ Version number changed for 2016.10a release ■ "Parameter Descriptions" on page 33 and "Register Descriptions" on page 55 auto-extracted from the RTL ■ Removed the "Running Leda on Generated Code with coreConsultant" section, and reference to Leda directory in Table 2-1 on page 20 ■ Removed the "Running Leda on Generated Code with coreAssembler" section, and reference to Leda directory in Table 2-4 on page 27 ■ Replaced Figure 2-2 and Figure 2-3 to remove references to Leda ■ Xprop directory in Table 2-1 and Table 2-4 ■ Moved "Internal Parameter Descriptions" to Appendix ■ Added "Running VCS XPROP Analyzer"
2.12a	June 2015	■ Added "Running SpyGlass® Lint and SpyGlass® CDC" ■ Added "Running SpyGlass on Generated Code with coreAssembler" ■ "Signal Descriptions" on page 45 auto-extracted from the RTL ■ Added "Internal Parameter Descriptions" on page 121 ■ Added Appendix B, "Basic Core Module (BCM) Library"
2.11a	June 2014	■ Version change for 2014.06a release ■ Added "Performance" section in the "Integration Considerations" chapter ■ Corrected the External Input/Output delays in Signals chapter
2.10a	May 2014	■ Version change for 2013.05a release ■ Updated the template
2.09b	October 2012	Added the product code on the cover and in Table 1-1
2.09b	March 2012	Version change for 2012.03a release
2.09a	November 2011	Version change for 2011.11a release
2.08b	October 2011	Version change for 2011.10a release
2.08b	June 2011	■ Updated system diagram in Figure 1-1 ■ Enhanced "Related Documents" section in Preface
2.08a	April 2011	Clarified Note in "Register Memory Map" section
2.07a	September 2010	Corrected names of include files and vcs command used for simulation
2.06a	December 2009	■ Corrected ICT_HAS_VECTOR_PORT to ICT_ADD_VECTOR_PORT parameter name ■ Updated databook to new template for consistency with other IIP/VIP/PHY databooks
2.06a	May 2009	Removed references to QuickStarts, as they are no longer supported

(Continued)

Version	Date	Description
2.06a	October 2008	Version change for 2007.10a release
2.05c	August 2008	Added note about irq_intpfilt signal and changed names of signals in Figures 6 & 10
2.05c	June 2008	Version change for 2008.06a release
2.04b	June 2007	Version change for 2007.06a release

Preface

This databook provides information that you need to interface the DesignWare® APB Interrupt Controller (DW_ahb_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 component.

DatabookOrganization

The chapters of this databook are organized as follows:

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

Related Documentation

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

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

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

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

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

    File > Build Debug Tar-file

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

  a. https://solvnetplus.synopsys.com

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

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

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

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

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

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

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

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

Product Code

Table 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

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

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

1 Product Overview

This chapter describes the DesignWare AHB Interrupt Controller, referred to as the DW_ahb_ictl.

1.1 DesignWare System Overview

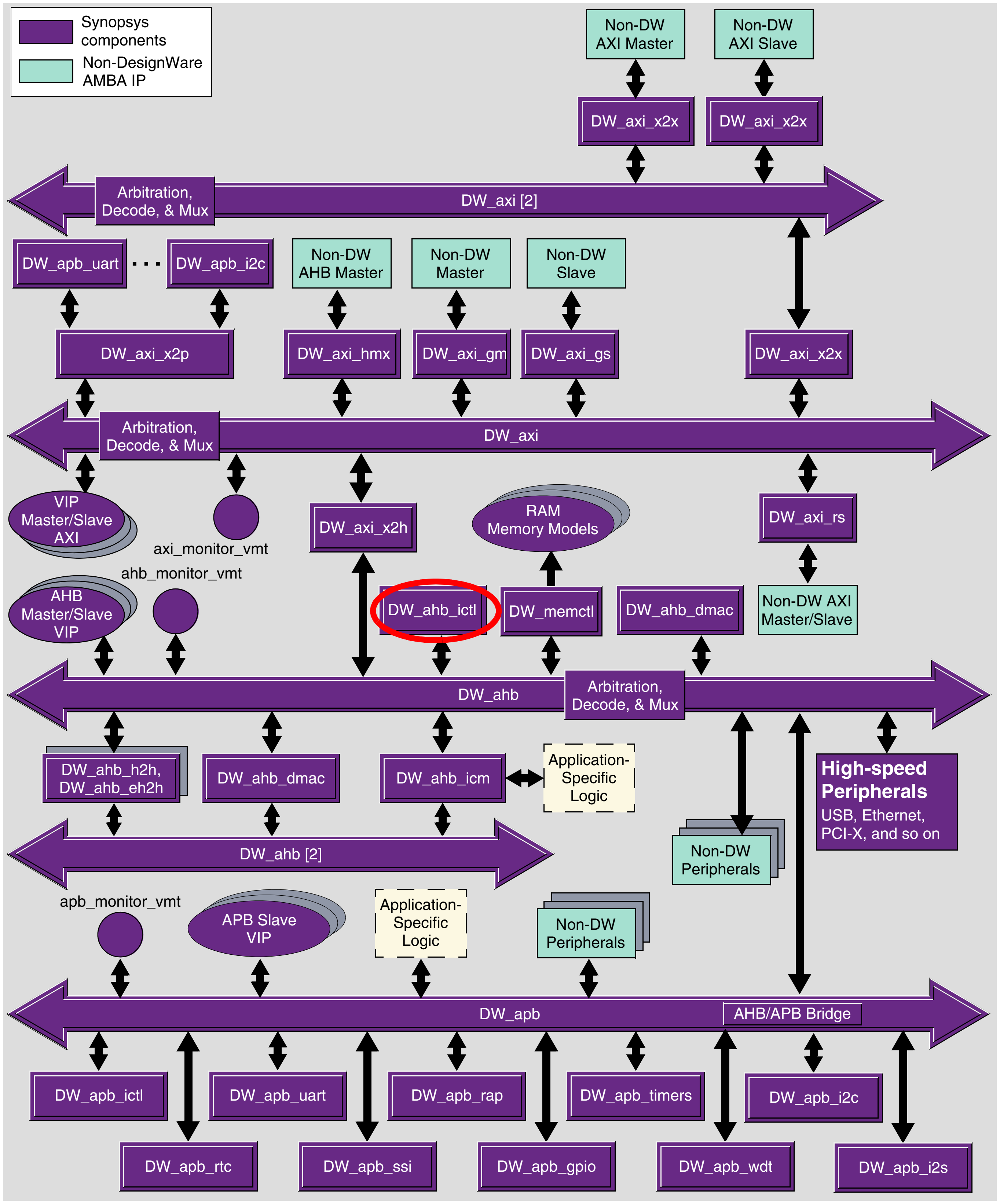
The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components.

1.1.1 DesignWare System Block Diagram

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

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

Figure 1-1 Example of DW_ahb_ictl in a Complete System



You can connect, configure, synthesize, and verify the DW_ahb_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_ahb_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_ahb_ictl is a configurable, vectored interrupt controller for AMBA-based systems. It is an AMBA 2.0-compliant Advanced High-speed Bus (AHB) slave device and is part of the DesignWare Synthesizable Components for AMBA 2.

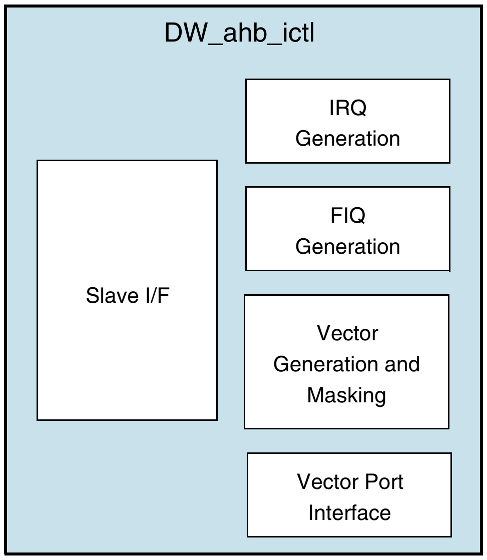
The DW_ahb_ictl supports two to sixty-four normal interrupt (IRQ) sources that are processed to produce a single IRQ interrupt to the processor. It supports one to eight fast interrupt (FIQ) sources that are processed to produce a single FIQ interrupt to the processor. All interrupt processing is combinational so that interrupts are propagated if the bus interface of the DW_ahb_ictl is powered down. This means that interrupts are registered only when the system accesses the interrupt registers (raw, status, or final_status). Ensure 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.

1.2.1 DW_ahb_ictl Block Diagram

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

Figure 1-2 Block Diagram DW_ahb_ictl



1.3 Features

The DW_ahb_ictl supports the following features:

- 2 to 64 IRQ normal interrupt sources
- 1 to 8 FIQ fast interrupt sources (optional)
- Vectored interrupts (optional)
- Software interrupts
- Priority filtering (optional)
- Masking
- Scan mode (optional)
- Programmable interrupt priorities (after configuration)
- Configuration ID registers
- Encoded parameters
- Vector port interface that allows a processor to sample the vector address associated with the current highest priority irq without a bus access
- Configuration of DesignWare AHB Lite system

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_ahb_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_ahb_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 101 chapter discusses the specific procedures for verifying the DW_ahb_ictl.

1.6 Licenses

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

1.7 Where to Go From Here

At this point, you may want to get started working with the DW_ahb_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_ahb_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_ahb_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 AHB Interrupt Controller, referred to as DW_ahb_ictl.

There is an option to configure AHB Lite, which is the DesignWare implementation of AMBA 2.0 AHB-Lite. The DesignWare AHB Lite configuration,

- Does not include requesting/granting protocols to the arbiter and split/retry responses from the slaves; all slaves are made non-split capable
- Does not include arbiter as the signals associated with the component are not used: hbusreq and hgrant
- Does not include write data, address, or control multiplexers
- Pause mode is not enabled
- Default master number is changed to 1
- Number of masters is changed to 1

For more information about AHB Lite, see the *DesignWare DW_ahb Databook*.

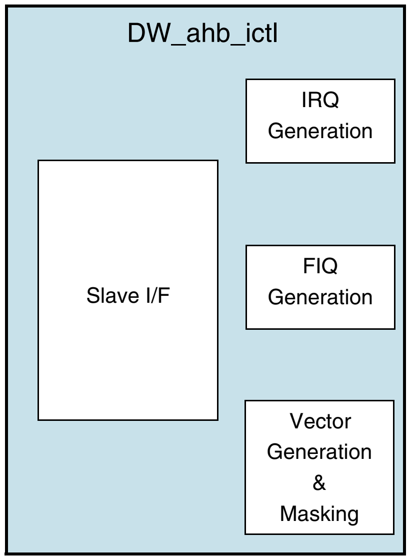
2.1 Overview

The DW_ahb_ictl component is a configurable, vectored interrupt controller for DesignWare Synthesizable Components. It supports from two to sixty-four normal interrupt (IRQ) sources that are processed to produce a single IRQ interrupt to the processor. It supports from one to eight 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_ahb_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. Ensure 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_ahb_ictl.

Figure 2-1 DW_ahb_ictl Block Diagram



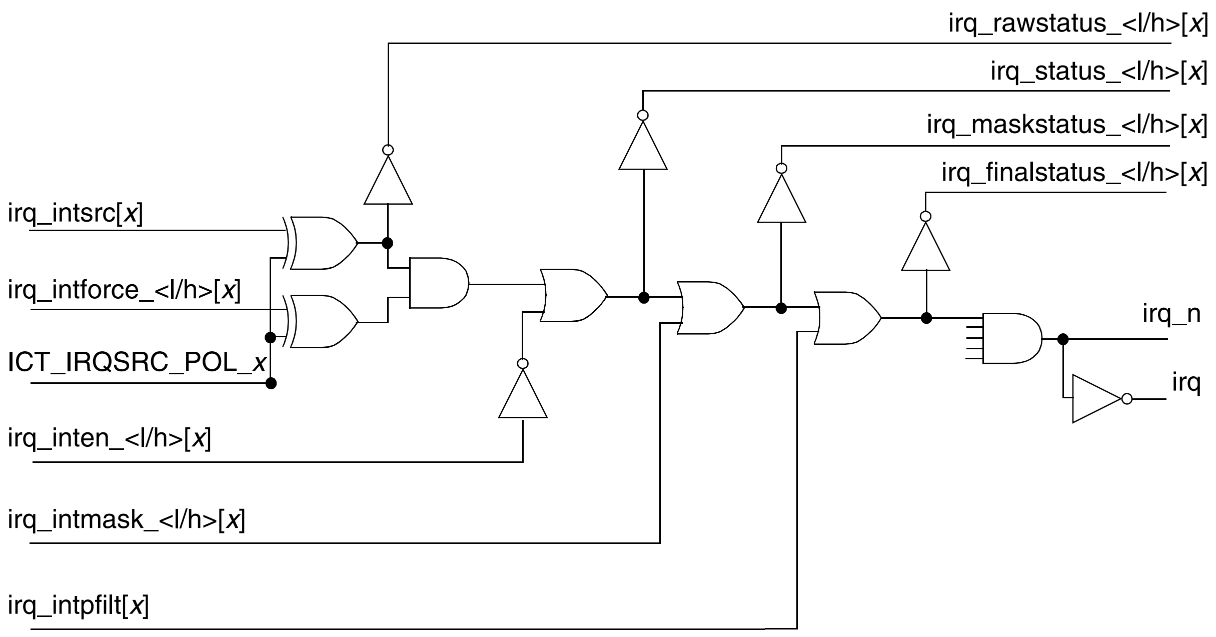
2.2 IRQ Interrupt Processing

The DW_ahb_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_ahb_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 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_ahb_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 bit of the irq_inten registers is 1 on reset, which enables the corresponding interrupt source.

To mask each interrupt source independently, 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_ahb_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 16 values from 0x0 to 0xf. Priority registers only exist for 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 (True). It is possible to read the priority registers when they (priorities) are not programmable.
- To enable programmable priorities, set the ICT_HC_PRIORITIES parameter to 0 (False). 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_ahb_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 16 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 = 1), 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_ahb_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_ahb_ictl includes up to four status registers used for querying the current status of any interrupt at various stages of the processing. see 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

  The irq_rawstatus register (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

  The irq_status register (irq_status_l/irq_status_h) contains the state of all interrupts after the enabling stage, meaning that an active-high bit indicates that particular interrupt source is active and enabled.

- irq_maskstatus

  The irq_maskstatus register (irq_maskstatus_l/irq_maskstatus_h) contains the state of all interrupts after the masking stage, meaning that an active-high bit indicates that particular interrupt source is active, enabled, and not masked.

- irq_finalstatus

  This register (irq_finalstatus_l/irq_finalstatus_h) contains the state of all interrupts after the priority filtering stage, meaning 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_ahb_ictl can be configured to support interrupt vectors. To enable vectored interrupt support, you 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_ahb_ictl has one vector register associated with each of the 16 interrupt priority levels: irq_vector_x 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.

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 with 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 16 interrupt vectors that have been programmed or configured into the irq_vector registers.
- By reading the irq_vector register, retrieve the vector associated with the highest priority level that has an active interrupt source.

The irq_vector register may be accessed only with hsize = 32. By imposing this restriction, the user is guaranteed to read a valid value for the irq_vector register. Accesses with hsize less that 32 would require shadow registers to guarantee a coherent value for the irq_vector register.

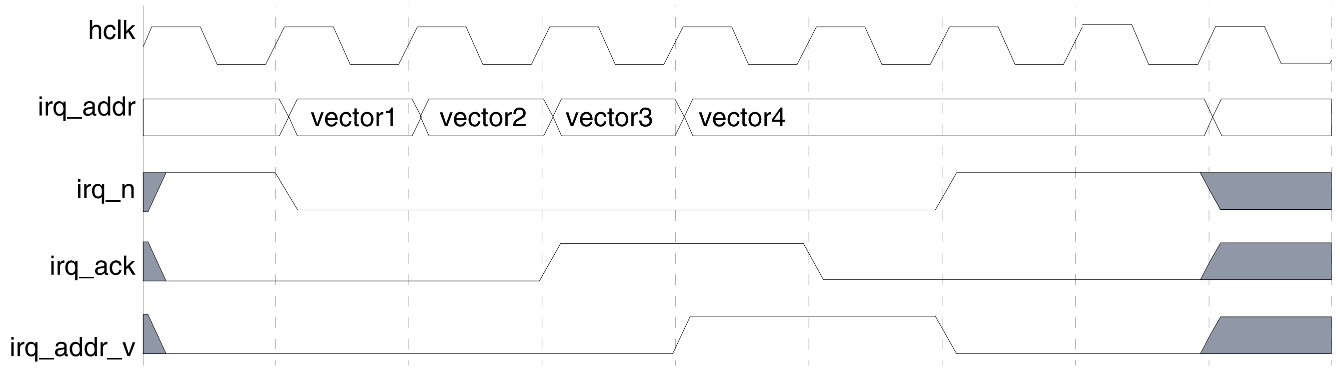
2.2.8 Vector Port

You can configure the DW_ahb_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_ahb_ictl that allow 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 AHB bus transaction in order to discover the vector address associated with the current highest-priority interrupt.

The DW_ahb_ictl vector port meets the requirements of the ARM11 and ARM1026EJ processor VIC ports. Vector port functionality applies to only IRQ processing; FIQ processing is 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)



2.2.8.1 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_ahb_ictl to assert irq_addr_v in order to inform the processor that it may sample irq_addr. Until the DW_ahb_ictl asserts irq_addr_v, irq_addr changes combinatorially to reflect the vector associated with the currently highest-priority active interrupt source. The DW_ahb_ictl holds irq_addr static while irq_addr_v is asserted.

In the same cycle that the DW_ahb_ictl de-asserts irq_addr_v, irq_n also de-asserts. At this point in the handshaking, the DW_ahb_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_ahb_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_ahb_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_ahb_ictl.

2.2.8.2 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_ahb_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_ahb_ictl.

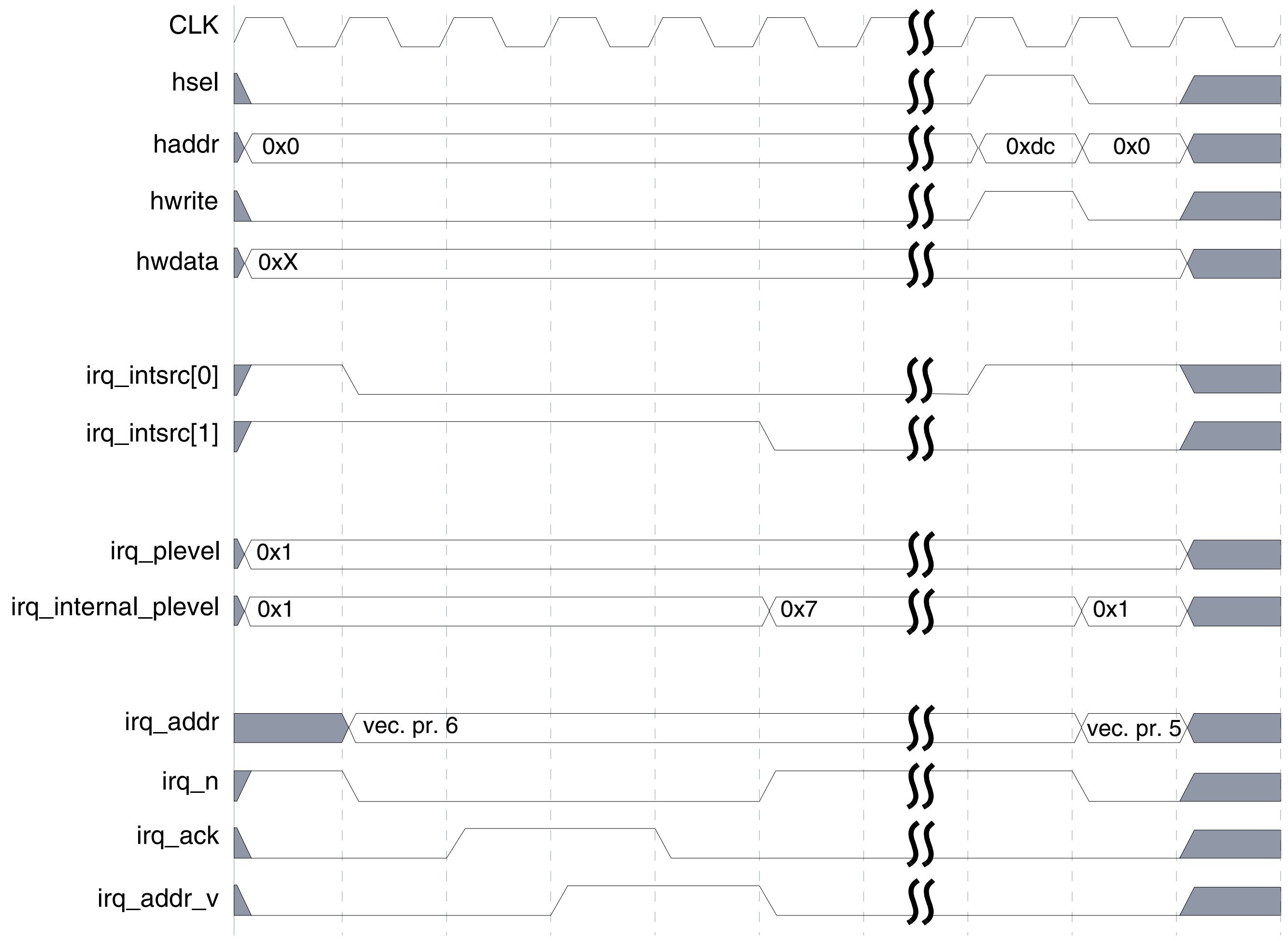
Note For writes to the irq_internal_plevel register, the write data on the bus is ignored. The only effect of a write to the irq_internal_plevel register is to reset its value to that of the irq_plevel register.

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 26.

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_ahb_ictl.

Figure 2-4 shows how the priority filter value is changed by the DW_ahb_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.

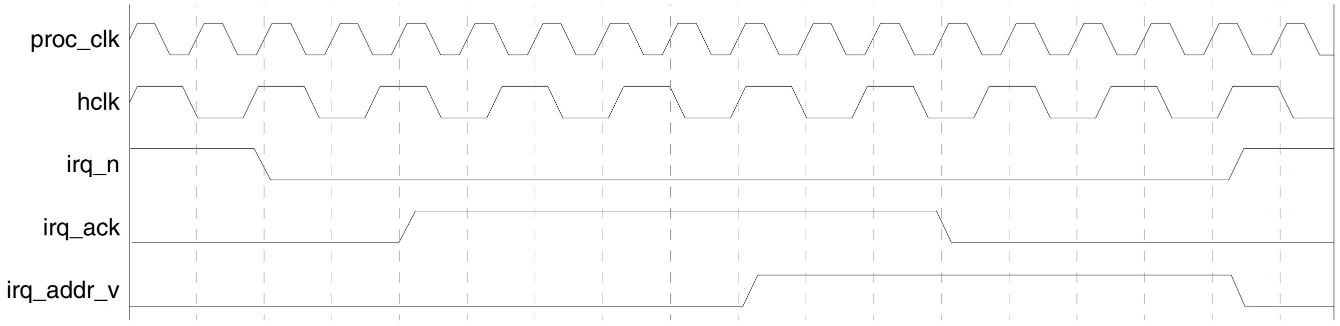
2.2.8.3 Synchronization

You can configure the DW_ahb_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_ahb_ictl adds N-stages of hclk register synchronization to all signals coming from the processor, where N= ICT_ADD_VECTOR_PORT_SYNC_DEPTH. By default, DW_ahb_ictl adds 2-stages of hclk 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_ahb_ictl go through two levels of metastability registers that give protection from one hclk cycle of metastability.

In this situation, signals from the DW_ahb_ictl to the processor should be synchronized external to the DW_ahb_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 N+1 cycles of latency between irq_ack and irq_addr_v.

Synchronization is not required for integer multiple or quasi-synchronous processor and bus clocks.

2.2.8.4 Interrupt Timing

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

1. Sample vector associated with IRQ using DW_ahb_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.

2.3 FIQ Interrupt Processing

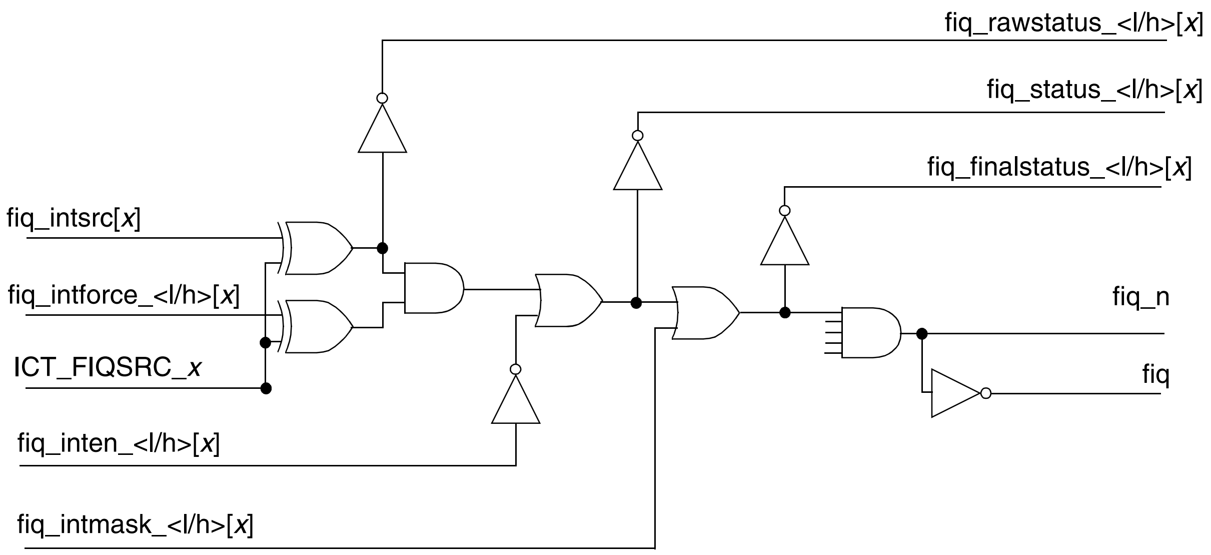
FIQ interrupts are an optional feature of the DW_ahb_ictl interrupt controller. To include FIQ interrupt logic in the DW_ahb_ictl, set the value of the ICT_HAS_FIQ parameter to 1. If ICT_HAS_FIQ is set to 0, the FIQ logic is not included and the associated signals are not present.

You can configure the DW_ahb_ictl to support from 1 to 8 FIQ interrupt sources (fiq_intsrc*N*) using the ICT_FIQ_NUM parameter. The DW_ahb_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_ahb_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



2.3.1 FIQ Interrupt Polarity

The input polarity of each FIQ interrupt source can be configured individually. 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.

2.3.2 FIQ Software-Programmable Interrupts

The DW_ahb_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.

2.3.3 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.

2.3.4 FIQ Interrupt Status Registers

The DW_ahb_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 on page 30. All of the following status registers have the same polarity; a 1 indicates that in interrupt is active, a 0 indicates inactive:

- fiq_rawstatus

  The fiq_rawstatus register 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.

- fiq_status

  The fiq_status register contains the state of all interrupts after the enabling stage, meaning that an active-high bit indicates that particular interrupt source is active and enabled.

- fiq_finalstatus

  The fiq_finalstatus register contains the state of all interrupts after the masking, meaning that an active-high bit indicates that particular interrupt source is active, enabled, and unmasked.

3 Parameter Descriptions

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

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

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

- Top Level Parameters on page 34
- Vector Port Interface on page 39
- Priority Controller Configuration on page 40
- Configuration of Vector Generation Module on page 41
- Individual IRQ Polarity Configuration on page 42
- Individual FIQ Polarity Configuration on page 43

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
	ICTL Source Code Configuration
Use DesignWare Foundation Synthesis Library	Specifies the availability of a DesignWare license. 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 DesignWare License is not available **Enabled:** When both Source and DesignWare licenses are available **Parameter Name:** USE_FOUNDATION
	System Configuration
AHB Data Bus Width	Specifies the width of the AHB data bus. **Values:** 32, 64, 128, 256 **Default Value:** 32 **Enabled:** Always **Parameter Name:** AHB_DATA_WIDTH
AHB System Address Width	Specifies the system address width of the AHB bus. **Note:** The current DW_ahb_ictl design supports AHB address width of 32-bits only. **Values:** ■ 32 (32) ■ 64 (64) **Default Value:** 32 **Enabled:** Always **Parameter Name:** HADDR_WIDTH
Single-bit AHB Response?	When set to true, hresp is a single-bit signal, else it is of 2 bits. **Values:** 0, 1 **Default Value:** 0 **Enabled:** Always **Parameter Name:** AHB_SCALAR_HRESP

Table 3-1 Top Level Parameters (Continued)

Label	Description
Big Endian AHB Data Bus ?	Selects the endianness (big-endian or little-endian) of the AHB system. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** BIG_ENDIAN
Internal Configuration	
Add Encoded Parameters	Provides firmware an easy and quick way of identifying the DesignWare component within an I/O memory map. Some critical design-time options determine how a driver must interact with the peripheral. There is a minimal area overhead by including these parameters. When set, this parameter allows a single driver to be developed for each component that is self-configurable. **Values:** ■ false (0x0) ■ true (0x1) **Default Value:** true **Enabled:** Always **Parameter Name:** ICT_ADD_ENCODED_PARAMS
Active-high Level for IRQ/FIQ outputs?	Drives the polarity of the FIQ and IRQ interrupt output signals to active high. Both the fast and normal interrupts are the same polarity. **Values:** ■ false (0) ■ true (1) **Default Value:** true **Enabled:** Always **Parameter Name:** ICT_INT_POL
Make irq_intforce and fiq_intforce active high?	Drives the irq_intforce and fiq_intforce registers to active high. By 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

Table 3-1 Top Level Parameters (Continued)

Label	Description
IRQ Configuration	
Install Priority Controller?	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?	Determines whether the priority levels can be read or not. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_READ_PRIORITY
Hard-Coded Priorities?	Determines whether the priority levels can be programmed or not. **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_ahb_ictl. This parameter can only be set if the ICT_HAS_PFLT parameter is set to True (1). **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_HAS_VECTOR_USER
Number of irq sources (2 -> 64)	Defines the number of interrupt sources to generate. **Values:** 2, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** ICT_IRQ_NUM

Table 3-1 Top Level Parameters (Continued)

Label	Description
IRQ Source Polarity Type	Specifies the IRQ Source Polarity Type. You must either use the individual interrupt polarities or override these to be either 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, ..., 0xffffffff **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
Number of fiq sources (1 -> 8)	Defines the number of fast interrupt sources to generate. This parameter can be set only if the ICT_HAS_FIQ parameter 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 must either use the individual interrupt polarities or override these to be either 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

Table 3-1 Top Level Parameters (Continued)

Label	Description
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, ..., 0xf **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 AHB 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 an N-stage (where N = ICT_ADD_VECTOR_PORT_SYNC_DEPTH) hclk synchronization on vector port signals coming from the processor (irq_ack). This parameter is used when the processor clock and hclk 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 hclk 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. The interrupts must have a priority greater than or equal to this value to be propagated to the CPU. This parameter value may always be overwritten by 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 x (for x = 1; x <= ICT_IRQ_NUM)	Sets the default priority level for interrupt source 0. It can be set only if the ICT_HAS_PFLT parameter is set to True (1). **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf **Default Value:** ~~0x0~~n for ICT_ISRC_PLEVEL_x; where n = x for 16 > x, n = x - 16 for 32 > x > 15, n = x - 32 for 48 > x > 31, n = x - 48 for 64 > x > 47 **Enabled:** ICT_HAS_PFLT==1 **Parameter Name:** ICT_ISRC_PLEVEL_x

3.4 Configuration of Vector Generation Module Parameters

Table 3-4 Configuration of Vector Generation Module Parameters

Label	Description
	Configuration of Vector 0
Priority x vector (for x = 1; x <= ICT_IRQ_PLEVEL)	Vector for interrupts with a priority setting of 0. Value must be less than HADDR_WIDTH bits wide. **Values:** 0x0, ..., 0xffffffff **Default Value:** 0x0 **Enabled:** ICT_HAS_VECTOR_USER==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_VECTOR_x
Hardcode Vector x (for x = 1; x <= ICT_IRQ_PLEVEL)	Hardcodes the corresponding priority vector and the irq_vector_x register is read only. If this parameter is set to False (0), the corresponding priority vector is programmable and the corresponding irq_vector_x register is read/write. This is configurable only if ICT_HAS_VECTOR=1 and ICT_HAS_PFLT=1. **Values:** 0, 1 **Default Value:** 0 **Enabled:** ICT_HAS_VECTOR_USER==1 && ICT_HAS_PFLT==1 **Parameter Name:** ICT_HC_VECTOR_x

3.5 Individual IRQ Polarity Configuration Parameters

Table 3-5 Individual IRQ Polarity Configuration Parameters

Label	Description
Individual IRQ Polarity Configuration	
Interrupt irq x Active High ? (for x = 1; x <= ICT_IRQ_NUM)	Sets the interrupt level of interrupt 0 to either active high (1) or active low (0). This is configurable only if ICT_IRQSRC_POL_TYPE = Individual (0). **Values:** 0x0, 0x1 **Default Value:** ICT_IRQSRC_POL_TYPE != 1 ? 1 : 0 **Enabled:** ICT_IRQSRC_POL_TYPE == 0 && ICT_IRQ_NUM > 0 **Parameter Name:** ~~CT_IRQSRC_POL_x~~ICT_IRQSRC_POL_x

3.6 Individual FIQ Polarity Configuration Parameters

Table 3-6 Individual FIQ Polarity Configuration Parameters

Label	Description
	Individual FIQ Polarity Configuration
Interrupt x Active High ? (for x = 1; x <= ICT_FIQ_NUM)	Sets the interrupt level of interrupt 0 to either active high (1) or active low (0). This is configurable only if ICT_FIQSRC_POL_TYPE = Individual (0). **Values:** 0x0, 0x1 **Default Value:** ICT_FIQSRC_POL_TYPE != 1 ? 1 : 0 **Enabled:** ICT_HAS_FIQ==1 && ICT_FIQSRC_POL_TYPE == 0 && ICT_FIQ_NUM > 0 **Parameter Name:** ICT_FIQSRC_POL_x

4 Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- AHB Interface on page 47
- Interrupt on page 50
- Interrupt Source on page 51
- Vector Interrupt and Handshake Signals on page 52
- Miscellaneous on page 53

4.1 AHB Interface Signals

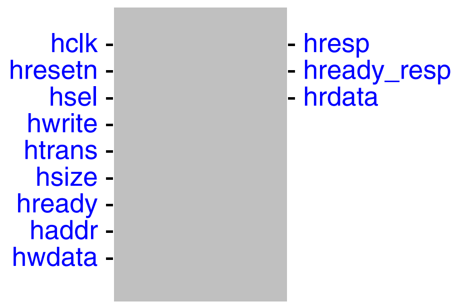


Table 4-1 AHB Interface Signals

Port Name	I/O	Description
hclk	I	Bus clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresetn	I	Bus reset. Removed synchronously to hclk **Exists:** Always **Synchronous To:** Asynchronous **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
hsel	I	Slave select line. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hwrite	I	Write control. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-1 AHB Interface Signals (Continued)

Port Name	I/O	Description
htrans[1:0]	I	Transfer control. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hsize[2:0]	I	Transfer size. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hready	I	Current transfer is complete. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
haddr[(HADDR_WIDTH-1):0]	I	Address. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwdata[(AHB_DATA_WIDTH-1):0]	I	Write data. **Exists:** Always **Synchronous To:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresp[~~1~~(HRESP_WIDTH-1):0]	O	Response type from slave. **Exists:** Always **Synchronous To:** hclk **Registered:** ~~0~~AHB_SCALAR_HRESP == 0 ? "0:0=Yes;1:1=No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-1 AHB Interface Signals (Continued)

Port Name	I/O	Description
hready_resp	O	Response from slave. When asserted, current transfer has completed. **Exists:** Always **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
hrdata[(AHB_DATA_WIDTH-1):0]	O	Data from slave. **Exists:** Always **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Interrupt Signals

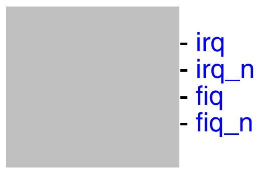


Table 4-2 Interrupt Signals

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

4.3 Interrupt Source Signals

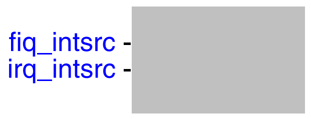


Table 4-3 Interrupt Source Signals

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

4.4 Vector Interrupt and Handshake Signals

irq_ack
irq_addr
irq_addr_v

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:** hclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
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==1 ? "Asynchronous" : "hclk" **Registered:** ICT_ADD_VECTOR_PORT_SYNC==1 ? Yes : No **Power Domain:** SINGLE_DOMAIN **Active State:** High
irq_addr_v	O	Optional. Asserted by DW_ahb_ictl to inform processor that it may sample irq_addr. **Exists:** ICT_ADD_VECTOR_PORT==1 **Synchronous To:** hclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Miscellaneous Signals

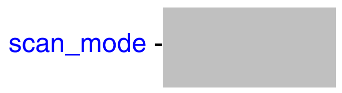


Table 4-5 Miscellaneous Signals

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 de-asserted- tied to logic 0. **Exists:** (ICT_HAS_PFLT==1) **Synchronous To:** Asynchronous **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
DW_ahb_ictl_addr_block1 on page 58	DW_ahb_ictl address block **Exists:** Always

5.1 DW_ahb_ictl_mem_map/DW_ahb_ictl_addr_block1 Registers

DW_ahb_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_ahb_ictl_mem_map/DW_ahb_ictl_addr_block1

Register	Offset	Description
irq_inten_l on page 60	0x0	~~This register specifies the interrupt enable bits for the lower 32 interrupt sources.~~ Interrupt Source Enable (Low) Register
irq_inten_h on page 61	0x4	~~This register specifies interrupt enable bit for upper 32 interrupt sources.~~ Interrupt Source Enable (High) Register
irq_intmask_l on page 62	0x8	~~This register masks interrupt bits for the lower 32 interrupt sources.~~ Interrupt Source Mask (Low) Register
irq_intmask_h on page 63	0xc	~~This register masks interrupt bits for the upper 32 interrupt sources.~~ Interrupt Source Mask (High) Register
irq_intforce_l on page 64	0x10	~~This register specifies interrupt force bits for the lower 32 interrupt sources.~~Interrupt Force (Low) Register
irq_intforce_h on page 65	0x14	~~This register specifies the interrupt force bits for the upper 32 interrupt sources.~~ Interrupt Force (High) Register
irq_rawstatus_l on page 66	0x18	~~This register specifies the lower 32 actual interrupt sources.~~ Interrupt Raw Status (Low) Register
irq_rawstatus_h on page 67	0x1c	~~This register specifies the upper 32 actual interrupt sources.~~ Interrupt Raw Status (High) Register
irq_status_l on page 68	0x20	~~This register specifies the interrupt status signal for the lower 32 interrupt sources.~~ Interrupt Status Low Register
irq_status_h on page 69	0x24	~~This register specifies the interrupt status signals for the upper 32 interrupt sources.~~ Interrupt Status (High) Register
irq_maskstatus_l on page 70	0x28	~~This register specifies the interrupt status signals for the lower 32 interrupt sources.~~Interrupt Mask Status (Low) Register
irq_maskstatus_h on page 71	0x2c	~~irq_maskstatus_h register specifies the interrupt status signals for the upper 32 interrupt...~~Interrupt Mask Status (High) Register
irq_finalstatus_l on page 72	0x30	~~This register specifies the interrupt status signals for the lower 32 interrupt sources.~~ Interrupt Final Status (Low) Register
irq_finalstatus_h on page 73	0x34	~~irq_finalstatus_h register specifies the interrupt status signals for the upper 32 interrupt...~~Interrupt Final Status (High) Register

Table 5-5 Registers for Address Block: DW_ahb_ictl_mem_map/DW_ahb_ictl_addr_block1 (Continued)

Register	Offset	Description
irq_vector on page 74	0x38	~~This register denotes the vector with the highest priority level interrupt.~~ IRQ Vector Register
irq_vector_x (for x = 1; x <= ICT_IRQ_PLEVEL) on page 75	0x40	Interrupt Vector (Priority Level 0) Register
fiq_inten on page 76	0xc0	~~This register indicates the bits to enable the Fast Interrupt.~~ Fast Interrupt Enable Register
fiq_intmask on page 77	0xc4	~~This register indicates the bits to mask a Fast Interrupt.~~ Fast Interrupt Mask Register
fiq_intforce on page 78	0xc8	~~This register indicate the~~ Fast Interrupt Force ~~bits.~~ Register
fiq_rawstatus on page 79	0xcc	~~This register indicates the~~ Fast Interrupt Source Raw Status ~~bits.~~ Register
fiq_status on page 80	0xd0	~~This register indicates the~~ Fast Interrupt Status Register ~~bits.~~
fiq_finalstatus on page 81	0xd4	~~This register indicates the~~ Fast Interrupt Final Status ~~bits.~~ Register
irq_plevel on page 82	0xd8	~~This register specifies the normal interrupt (IRQ)~~IRQ System Priority Level ~~bits.~~ Register
irq_internal_plevel on page 83	0xdc	~~This register specifies the internal IRQ system priority level.~~Internal IRQ System Priority Level Register
irq_pr_x (for x = 1; x <= ICT_IRQ_NUM) on page 85	0xe8	~~This register specifies the normal interrupt (IRQ) Individual Interrupt x Priority Level bits.~~IRQ Individual Interrupt 0 Priority Level Register
ICTL_COMP_PARAMS_2 on page 87	0x3f0	~~This register denotes the~~ Component Parameter Register 2 ~~bits.~~
ICTL_COMP_PARAMS_1 on page 91	0x3f4	~~This register denotes the~~ Component Parameter Register 1 ~~bits.~~
AHB_ICTL_COMP_VERSION on page 95	0x3f8	~~This register specifies the version of the Component.~~ Component Version Register
ICTL_COMP_TYPE on page 96	0x3fc	~~This register specifies the~~ Component Type ~~.~~Register

5.1.1 irq_inten_l

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

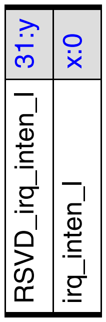


Table 5-6 Fields for Register: irq_inten_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_inten_l	R	RSVD_irq_inten_l 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 is disabled ■ 0x1 (ENABLED): Interrupt is enabled **Value After Reset:** {(The corresponding bits of the ICT_IRQ_DFLT_EN configuration parameter.)} **Exists:** Always **Range Variable[x]:** "(ICT_IRQ_NUM<32) ? ICT_IRQ_NUM : 32" - 1

5.1.2 irq_inten_h

- **Name:** Interrupt Source Enable (High) Register
- **Description:** This register specifies interrupt enable bit for upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

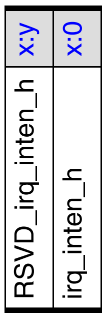


Table 5-7 Fields for Register: irq_inten_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_inten_h	R	RSVD_irq_inten_h 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 interrupt bits for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** Always

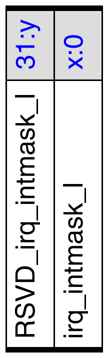


Table 5-8 Fields for Register: irq_intmask_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_intmask_l	R	RSVD_irq_intmask_l 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 indicate 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) ? ICT_IRQ_NUM : 32" - 1

5.1.4 irq_intmask_h

- **Name:** Interrupt Source Mask (High) Register
- **Description:** This register masks interrupt bits for the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

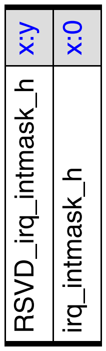


Table 5-9 Fields for Register: irq_intmask_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_intmask_h	R	RSVD_irq_intmask_h 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 indicate 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 interrupt force bits for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** Always

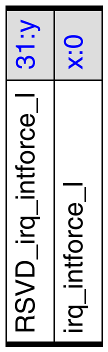


Table 5-10 Fields for Register: irq_intforce_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_intforce_l	R	RSVD_irq_intforce_l Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_IRQ_NUM
x:0	irq_intforce_l	R/W	These bits indicate the interrupt force bits for the lower 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. **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) ? ICT_IRQ_NUM : 32" - 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) ? 1 : 0

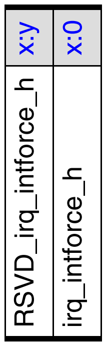


Table 5-11 Fields for Register: irq_intforce_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_intforce_h	R	RSVD_irq_intforce_h 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_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 lower 32 actual interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** Always

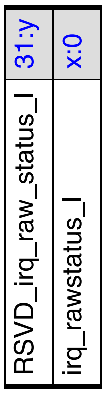


Table 5-12 Fields for Register: irq_rawstatus_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_raw_status_l	R	RSVD_irq_raw_status_l Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM
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) ? ICT_IRQ_NUM : 32" - 1

5.1.8 irq_rawstatus_h

- **Name:** Interrupt Raw Status (High) Register
- **Description:** This register specifies the upper 32 actual interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

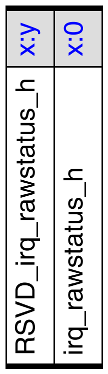


Table 5-13 Fields for Register: irq_rawstatus_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_rawstatus_h	R	RSVD_irq_rawstatus_h 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 signal for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x20
- **Exists:** Always

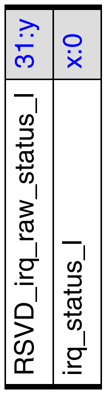


Table 5-14 Fields for Register: irq_status_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_raw_status_l	R	RSVD_irq_raw_status_l 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): Interrupt status is inactive ■ 0x1 (ACTIVE): Interrupt status is active **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) ? ICT_IRQ_NUM : 32" - 1

5.1.10 irq_status_h

- **Name:** Interrupt Status (High) Register
- **Description:** This register specifies the interrupt status signals for the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x24
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

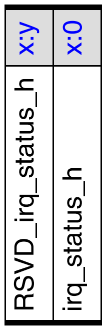


Table 5-15 **Fields for Register: irq_status_h**

Bits	Name	Memory Access	Description
x:y	RSVD_irq_status_h	R	RSVD_irq_status_h 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 status signals for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** Always

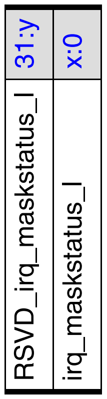


Table 5-16 Fields for Register: irq_maskstatus_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_maskstatus_l	R	RSVD_irq_maskstatus_l Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM
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) ? ICT_IRQ_NUM : 32" - 1

5.1.12 irq_maskstatus_h

- **Name:** Interrupt Mask Status (High) Register
- **Description:** irq_maskstatus_h register specifies the interrupt status signals for the upper 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

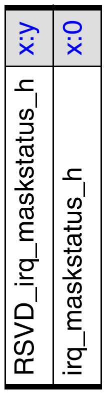


Table 5-17 Fields for Register: irq_maskstatus_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_maskstatus_h	R	RSVD_irq_maskstatus_h 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 status signals for the lower 32 interrupt sources.
- **Size:** 32 bits
- **Offset:** 0x30
- **Exists:** Always

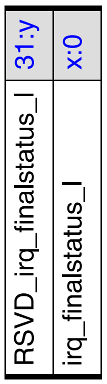


Table 5-18 Fields for Register: irq_finalstatus_l

Bits	Name	Memory Access	Description
31:y	RSVD_irq_finalstatus_l	R	RSVD_irq_finalstatus_l Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true **Range Variable[y]:** ICT_IRQ_NUM
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) ? ICT_IRQ_NUM : 32" - 1

5.1.14 irq_finalstatus_h

- **Name:** Interrupt Final Status (High) Register
- **Description:** {(irq_finalstatus_h register specifies the interrupt status signals for the upper 32 interrupt sources.)}
- **Size:** 32 bits
- **Offset:** 0x34
- **Exists:** (ICT_IRQ_NUM>32) ? 1 : 0

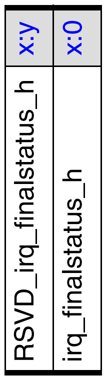


Table 5-19 Fields for Register: irq_finalstatus_h

Bits	Name	Memory Access	Description
x:y	RSVD_irq_finalstatus_h	R	RSVD_irq_finalstatus_h 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_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 denotes the vector with the highest priority level interrupt.
- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** Always

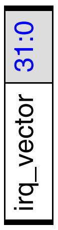


Table 5-20 Fields for Register: irq_vector

Bits	Name	Memory Access	Description
31:0	irq_vector	R	These bits indicate the vector with the highest priority level interrupt. When an interrupt occurs and the provided vectored interrupts are supported, this location can be read and it returns one of the 16 vectors. The returned vector corresponds to the highest priority level interrupt. This register returns 0 when ICT_HAS_VECTOR = 0. To ensure valid data, this location may be accessed only with an hsize of 32. Otherwise, the slave interface issues an error response. **Value After Reset:** {(System priority vector ICT_VECTOR_n value when there are no pending interrupts. Else, 0x0 when vectored interrupts not supported (ICT_HAS_VECTOR = 0).)} **Exists:** Always **Volatile:** true

5.1.16 irq_vector_x (for x = 1; x <= ICT_IRQ_PLEVEL)

- **Name:** Interrupt Vector (Priority Level 0) Register
- **Description:** Interrupt Vector (Priority Level 0) Register
- **Size:** 32 bits
- **Offset:** 0x40
- **Exists:** (((ICT_HAS_PFLT==1)?ICT_HAS_VECTOR_USER:0)==0) ? 0 : 1

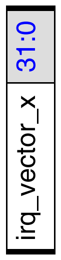


Table 5-21 Fields for Register: irq_vector_x (for x = 1; x <= ICT_IRQ_PLEVEL)

Bits	Name	Memory Access	Description
31:0	irq_vector_x	* Varies	Interrupt vector for priority level x. This register does not exist when ICT_HAS_VECTOR = 0. ■ If ICT_HC_VECTOR_x is set to 1, this is a read-only register and the interrupt vector is set by ICT_VECTOR_x. ■ If ICT_HC_VECTOR_x is set to 0, this is a read/write register used to program the interrupt vector and the reset value is determined by ICT_VECTOR_x. **Value After Reset:** System priority vector value when there are no pending interrupts. Else, 0x0 when vectored interrupts not supported (ICT_HAS_VECTOR = 0). **Exists:** Always **Memory Access:** "(ICT_HC_VECTOR_0==0) ? \"read-write\" : \"read-only\""

5.1.17 fiq_inten

- **Name:** Fast Interrupt Enable Register
- **Description:** This register indicates the bits to enable the Fast Interrupt.
- **Size:** 32 bits
- **Offset:** 0xc0
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

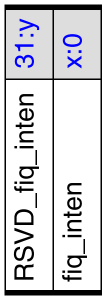


Table 5-22 Fields for Register: fiq_inten

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_inten	R	RSVD_fiq_inten Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	fiq_inten	R/W	These bits enable the Fast interrupt. A 1 in any bit position enables the corresponding interrupt. This register does not exist when ICT_HAS_FIQ = 0. **Values:** ■ 0x0 (DISABLED): Fast interrupt disabled ■ 0x1 (ENABLED): Fast interrupt enabled **Value After Reset:** ICT_FIQ_DFLT_EN **Exists:** Always **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.18 fiq_intmask

- **Name:** Fast Interrupt Mask Register
- **Description:** This register indicates the bits to mask a Fast Interrupt.
- **Size:** 32 bits
- **Offset:** 0xc4
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

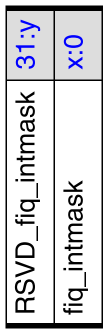


Table 5-23 Fields for Register: fiq_intmask

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_intmask	R	RSVD_fiq_intmask Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	fiq_intmask	R/W	These bits mask the Fast interrupt mask. A 1 in any bit position masks the corresponding interrupt. This register does not exist when ICT_HAS_FIQ = 0. **Values:** ■ 0x0 (UNMASK): Unmasks the Fast interrupts ■ 0x1 (MASK): Masks the Fast interrupts **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.19 fiq_intforce

- **Name:** Fast Interrupt Force Register
- **Description:** This register indicate the Fast Interrupt Force bits.
- **Size:** 32 bits
- **Offset:** 0xc8
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

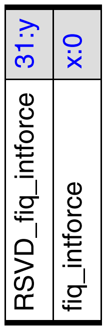


Table 5-24 Fields for Register: fiq_intforce

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_intforce	R	RSVD_fiq_intforce Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	fiq_intforce	R/W	These bits indicate the Fast interrupt force bits. Each bit in this register corresponds to one ICT_READ_PRIORITY bit of the irq_intsrc input. The polarity of the bits in the register correspond 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. This register does not exist when ICT_HAS_FIQ = 0. **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_FIQSRC_POL_x (where x ranges from 0 to ICT_FIQ_NUM) or ICT_FORCEREG_ACTIVE_HIGH.)} **Exists:** Always **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.20 fiq_rawstatus

- **Name:** Fast Interrupt Source Raw Status Register
- **Description:** This register indicates the Fast Interrupt Source Raw Status bits.
- **Size:** 32 bits
- **Offset:** 0xcc
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

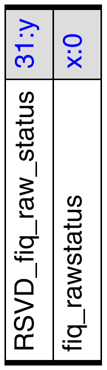


Table 5-25 Fields for Register: fiq_rawstatus

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_raw_status	R	RSVD_fiq_raw_status Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** ICT_FIQ_NUM
x:0	fiq_rawstatus	R	These bits indicate the Fast interrupt source raw input status. This register does not exist when ICT_HAS_FIQ = 0. **Values:** ■ 0x0 (INACTIVE): Fast interrupt status is inactive ■ 0x1 (ACTIVE): Fast interrupt status is active **Value After Reset:** {(fiq_rawstatus - Dependent on setting of corresponding interrupt source bit.)} **Exists:** Always **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.21 fiq_status

- **Name:** Fast Interrupt Status Register
- **Description:** This register indicates the Fast Interrupt Status Register bits.
- **Size:** 32 bits
- **Offset:** 0xd0
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

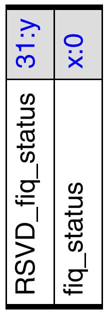


Table 5-26 Fields for Register: fiq_status

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_status	R	RSVD_fiq_status 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 indicate the Fast interrupt status after the forcing and interrupt enabling stage. This register does not exist when ICT_HAS_FIQ = 0. **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:** Always **Volatile:** true **Range Variable[x]:** ICT_FIQ_NUM - 1

5.1.22 fiq_finalstatus

- **Name:** Fast Interrupt Final Status Register
- **Description:** This register indicates the Fast Interrupt Final Status bits.
- **Size:** 32 bits
- **Offset:** 0xd4
- **Exists:** (ICT_HAS_FIQ==0) ? 0 : 1

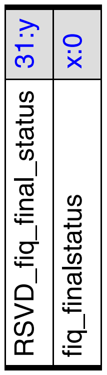


Table 5-27 Fields for Register: fiq_finalstatus

Bits	Name	Memory Access	Description
31:y	RSVD_fiq_final_status	R	RSVD_fiq_final_status 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 indicate the Fast interrupt status after the masking stage. This register does not exist when ICT_HAS_FIQ = 0. **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:** Always **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 normal interrupt (IRQ) System Priority Level bits.
- **Size:** 32 bits
- **Offset:** 0xd8
- **Exists:** Always

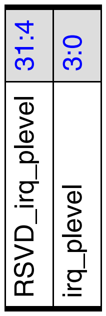


Table 5-28 Fields for Register: irq_plevel

Bits	Name	Memory Access	Description
31:4	RSVD_irq_plevel	R	RSVD_irq_plevel 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 will accept only interrupts that are enabled and have a priority the same or greater than the system level priority setting. If ICT_ADD_VECTOR_PORT is set to true, this register does not reflect the priority level being applied by the DW_ahb_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. During this time, irq_internal_plevel reflects the priority level currently being applied. **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

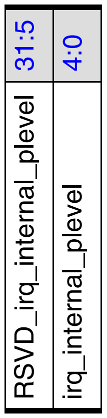


Table 5-29 Fields for Register: irq_internal_plevel

Bits	Name	Memory Access	Description
31:5	RSVD_irq_internal_plevel	R	irq_internal_plevel 31to5 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_ahb_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 irq_pr_x (for x = 1; x <= ICT_IRQ_NUM)

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

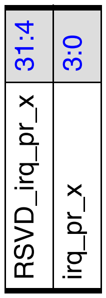


Table 5-30 Fields for Register: irq_pr_x (for x = 1; x <= ICT_IRQ_NUM)

Bits	Name	Memory Access	Description
31:4	RSVD_irq_pr_x	R	RSVD_irq_pr_x Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-30 Fields for Register: irq_pr_x (for x = 1; x <= ICT_IRQ_NUM) (Continued)

Bits	Name	Memory Access	Description
3:0	irq_pr_x	* Varies	These bits specify the Individual interrupt x priority level. The number of Individual Interrupt Priority Level registers is from 0 to ICT_IRQ_NUM-1. A register's value must be an integer from 0x0 to 0xf. **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_IRQ_PLEVEL_x~~{(ICT_IRQ_PLEVEL_x)} **Exists:** Always **Memory Access:** "(ICT_HC_PRIORITIES==0) ? \"read-write\" : \"read-only\""

5.1.26 ICTL_COMP_PARAMS_2

- **Name:** Component Parameter Register 2
- **Description:** This register denotes the Component Parameter Register 2 bits.
- **Size:** 32 bits
- **Offset:** 0x3f0
- **Exists:** Always

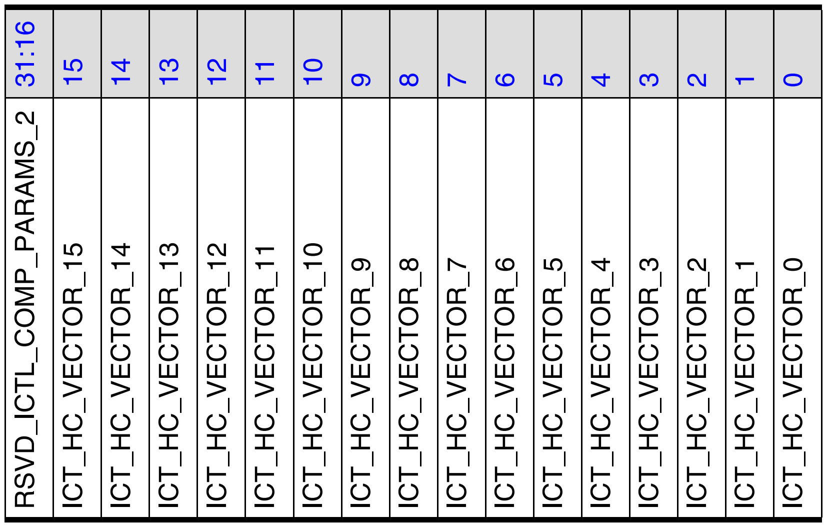


Table 5-31 Fields for Register: ICTL_COMP_PARAMS_2

Bits	Name	Memory Access	Description
31:16	RSVD_ICTL_COMP_PARAMS_2	R	RSVD_ICTL_COMP_PARAMS_2 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
15	ICT_HC_VECTOR_15	R	The value of this register is derived from the ICT_HC_VECTOR_15 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_15 **Exists:** Always

Table 5-31 Fields for Register: ICTL_COMP_PARAMS_2 (Continued)

Bits	Name	Memory Access	Description
14	ICT_HC_VECTOR_14	R	The value of this register is derived from the ICT_HC_VECTOR_14 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_14 **Exists:** Always
13	ICT_HC_VECTOR_13	R	The value of this register is derived from the ICT_HC_VECTOR_13 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_13 **Exists:** Always
12	ICT_HC_VECTOR_12	R	The value of this register is derived from the ICT_HC_VECTOR_12 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_12 **Exists:** Always
11	ICT_HC_VECTOR_11	R	The value of this register is derived from the ICT_HC_VECTOR_11 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_11 **Exists:** Always
10	ICT_HC_VECTOR_10	R	The value of this register is derived from the ICT_HC_VECTOR_10 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_10 **Exists:** Always

Table 5-31 Fields for Register: ICTL_COMP_PARAMS_2 (Continued)

Bits	Name	Memory Access	Description
9	ICT_HC_VECTOR_9	R	The value of this register is derived from the ICT_HC_VECTOR_9 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_9 **Exists:** Always
8	ICT_HC_VECTOR_8	R	The value of this register is derived from the ICT_HC_VECTOR_8 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_8 **Exists:** Always
7	ICT_HC_VECTOR_7	R	The value of this register is derived from the ICT_HC_VECTOR_7 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_7 **Exists:** Always
6	ICT_HC_VECTOR_6	R	The value of this register is derived from the ICT_HC_VECTOR_6 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_6 **Exists:** Always
5	ICT_HC_VECTOR_5	R	The value of this register is derived from the ICT_HC_VECTOR_5 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_5 **Exists:** Always

Table 5-31 Fields for Register: ICTL_COMP_PARAMS_2 (Continued)

Bits	Name	Memory Access	Description
4	ICT_HC_VECTOR_4	R	The value of this register is derived from the ICT_HC_VECTOR_4 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_4 **Exists:** Always
3	ICT_HC_VECTOR_3	R	The value of this register is derived from the ICT_HC_VECTOR_3 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_3 **Exists:** Always
2	ICT_HC_VECTOR_2	R	The value of this register is derived from the ICT_HC_VECTOR_2 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_2 **Exists:** Always
1	ICT_HC_VECTOR_1	R	The value of this register is derived from the ICT_HC_VECTOR_1 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_1 **Exists:** Always
0	ICT_HC_VECTOR_0	R	The value of this register is derived from the ICT_HC_VECTOR_0 parameter. **Values:** ■ 0x0 (FALSE): Programmable interrupt vector. ■ 0x1 (TRUE): Hard coded interrupt vector. **Value After Reset:** ICT_HC_VECTOR_0 **Exists:** Always

5.1.27 ICTL_COMP_PARAMS_1

- **Name:** Component Parameter Register 1
- **Description:** This register denotes the Component Parameter Register 1 bits.
- **Size:** 32 bits
- **Offset:** 0x3f4
- **Exists:** Always

31:29	28	27	26:24	23:y	x:16	15	14	13:8	7:4	3	2	1:0
RSVD_ICTL_COMP_PARAMS_1	ICT_HC_PRIORITIES	ICT_READ_PRIORITY	ENCODED_ICT_FIQ_NUM	RSVD_ICT_FIQ_DFLT_EN	ICT_FIQ_DFLT_EN	ICT_HAS_VECTOR	ICT_HAS_PFLT	ENCODED_ICT_IRQ_NUM	ICT_IRQ_PLEVEL	ICT_HAS_FIQ	ICT_FORCEREG_ACTIVE_HIGH	ENCODED_AHB_DATA_WIDTH

Table 5-32 Fields for Register: ICTL_COMP_PARAMS_1

Bits	Name	Memory Access	Description
31:29	RSVD_ICTL_COMP_PARAMS_1	R	RSVD_ICTL_COMP_PARAMS_1 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
28	ICT_HC_PRIORITIES	R	The value of this register is derived from the ICT_HC_PRIORITIES parameter. **Values:** ■ 0x0 (FALSE): Not able to read Priorities ■ 0x1 (TRUE): Able to read Priorities **Value After Reset:** ICT_HC_PRIORITIES **Exists:** Always

Table 5-32 Fields for Register: ICTL_COMP_PARAMS_1 (Continued)

Bits	Name	Memory Access	Description
27	ICT_READ_PRIORITY	R	The value of this register is derived from the ICT_READ_PRIORITY parameter. **Values:** ■ 0x0 (FALSE): Priorities are not hardcoded ■ 0x1 (TRUE): Priorities are hardcoded **Value After Reset:** ICT_READ_PRIORITY **Exists:** Always
26:24	ENCODED_ICT_FIQ_NUM	R	This value is an encoded representation of the ICT_FIQ_NUM parameter. **Values:** ■ 0x0 (FIQ_NUM_1): Number of FIQ sources is equals to 1. ■ 0x1 (FIQ_NUM_2): Number of FIQ sources is equals to 2. ■ 0x2 (FIQ_NUM_3): Number of FIQ sources is equals to 3. ■ 0x3 (FIQ_NUM_4): Number of FIQ sources is equals to 4. ■ 0x4 (FIQ_NUM_5): Number of FIQ sources is equals to 5. ■ 0x5 (FIQ_NUM_6): Number of FIQ sources is equals to 6. ■ 0x6 (FIQ_NUM_7): Number of FIQ sources is equals to 7. ■ 0x7 (FIQ_NUM_8): Number of FIQ sources is equals to 8. **Value After Reset:** ICT_FIQ_NUM-1 **Exists:** Always
23:y	RSVD_ICT_FIQ_DFLT_EN	R	RSVD_ICT_FIQ_DFLT_EN Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** (ICT_FIQ_NUM<8) ? 1 : 0 **Range Variable[y]:** ICT_FIQ_NUM + 16
x:16	ICT_FIQ_DFLT_EN	R	The value of this register is derived from the ICT_FIQ_DFLT_EN coreConsultant parameter. **Values:** ■ 0x0 (ENABLES): Individual fiq enables on reset **Value After Reset:** (`ICT_FIQ_NUM == 8) ? `ICT_FIQ_DFLT_EN : {{(8-`ICT_FIQ_NUM){1'b0}}, `ICT_FIQ_DFLT_EN}) **Exists:** Always **Range Variable[x]:** ICT_FIQ_NUM + 15

Table 5-32 Fields for Register: ICTL_COMP_PARAMS_1 (Continued)

Bits	Name	Memory Access	Description
15	ICT_HAS_VECTOR	R	The value of this register is derived from the ICT_HAS_VECTOR parameter. **Values:** ■ 0x0 (FALSE): Interrupt vector generation circuitry is not instantiated. ■ 0x1 (TRUE): Interrupt vector generation circuitry is instantiated. **Value After Reset:** ICT_HAS_VECTOR **Exists:** Always
14	ICT_HAS_PFLT	R	The value of this register is derived from the ICT_HAS_PFLT parameter. **Values:** ■ 0x0 (FALSE): Does not allow interrupts that are assigned a priority level. ■ 0x1 (TRUE): Allows interrupts that are assigned a priority level to be compared against a system-level priority level. **Value After Reset:** ICT_HAS_PFLT **Exists:** Always
13:8	ENCODED_ICT_IRQ_NUM	R	This value is an encoded representation of the ICT_IRQ_NUM parameter. 0x00 = 2 to 0x3D = 64 0x3E to 0x3F = reserved **Value After Reset:** ICT_IRQ_NUM-2 **Exists:** Always
7:4	ICT_IRQ_PLEVEL	R	The value of this register is derived from the ICT_IRQ_PLEVEL parameter. **Value After Reset:** ICT_IRQ_PLEVEL **Exists:** Always
3	ICT_HAS_FIQ	R	The value of this register is derived from the ICT_HAS_FIQ parameter. **Values:** ■ 0x0 (FALSE): Default system priority controller filter level is not defined. ■ 0x1 (TRUE): Defines the default system priority controller filter level. **Value After Reset:** ICT_HAS_FIQ **Exists:** Always

Table 5-32 Fields for Register: ICTL_COMP_PARAMS_1 (Continued)

Bits	Name	Memory Access	Description
2	ICT_FORCEREG_ACTIVE_HIGH	R	The value of this register is derived from the ICT_FORCEREG_ACTIVE_HIGH parameter. **Values:** ■ 0x0 (FALSE): The irq_intforce and fiq_intforce register become active low. ■ 0x1 (TRUE): The irq_intforce and fiq_intforce register become active high. **Value After Reset:** ICT_FORCEREG_ACTIVE_HIGH **Exists:** Always
1:0	ENCODED_AHB_DATA_WIDTH	R	This value is an encoded representation of the AHB_DATA_WIDTH parameter. **Values:** ■ 0x0 (AHB_DATA_WIDTH_32): AHB data width is 32 bit ■ 0x1 (AHB_DATA_WIDTH_64): AHB data width is 64 bit ■ 0x2 (AHB_DATA_WIDTH_128): AHB data width 128 bit ■ 0x3 (AHB_DATA_WIDTH_256): AHB data width 256 bit **Value After Reset:** {(Encoded Value of AHB_DATA_WIDTH; AHB_DATA_WIDTH=32, Reset Value=0; AHB_DATA_WIDTH=64, Reset Value=1; AHB_DATA_WIDTH=128, Reset Value=2; AHB_DATA_WIDTH=256, Reset Value=3)} **Exists:** Always

5.1.28 AHB_ICTL_COMP_VERSION

- **Name:** Component Version Register
- **Description:** This register specifies the version of the Component.
- **Size:** 32 bits
- **Offset:** 0x3f8
- **Exists:** Always

Table 5-33 Fields for Register: AHB_ICTL_COMP_VERSION

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

5.1.29 ICTL_COMP_TYPE

- **Name:** Component Type Register
- **Description:** This register specifies the Component Type.
- **Size:** 32 bits
- **Offset:** 0x3fc
- **Exists:** Always

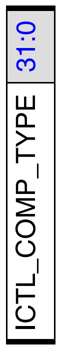


Table 5-34 Fields for Register: ICTL_COMP_TYPE

Bits	Name	Memory Access	Description
31:0	ICTL_COMP_TYPE	R	These bits specify the DesignWare Component Type number (0x44_57_11_20). This assigned unique hexadecimal value is constant and is derived from the two ASCII letters 'DW' followed by a 16-bit unsigned number. **Value After Reset:** 0x44571120 **Exists:** Always

6 Programming the DW_ahb_ictl

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

6.1 Programming Considerations

The APB data bus width is independently configurable from the width of registers internal to the DW_ahb_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_ahb_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.

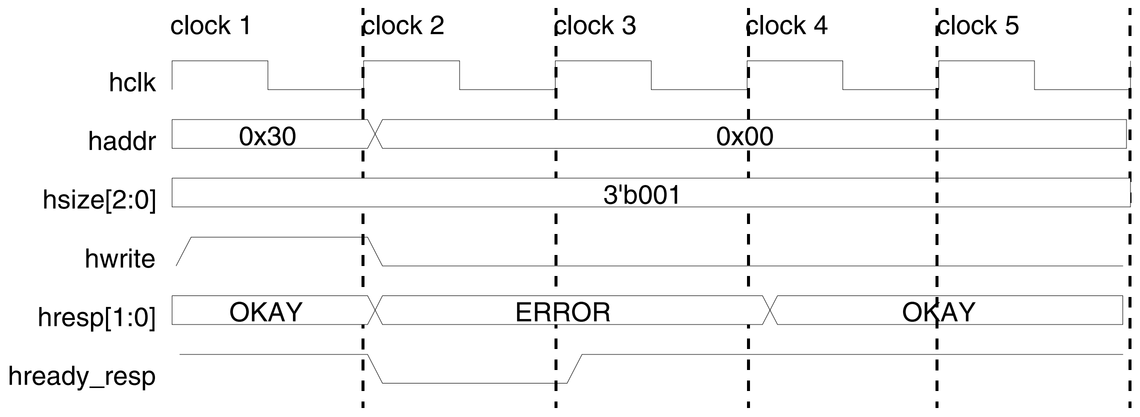
6.5 Illegal Accesses

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

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

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

Figure 6-1 AHB Error Response



6.6 Register Access

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

When performing a transfer on the AHB bus, the address must be aligned with the value of hsize. For example, if hsize = 32, then the address must be aligned to 32 bits, such as the two least significant bits equal 'b0. This condition is not checked in the slave interface but is a requirement of the AMBA specification. Also, hsize must not specify a transfer width greater than the slave's AHB data width; this is also a protocol violation.

7 Verification

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

Note The DW_ahb_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 Installation Guide.*

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

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

This chapter discusses the following sections:

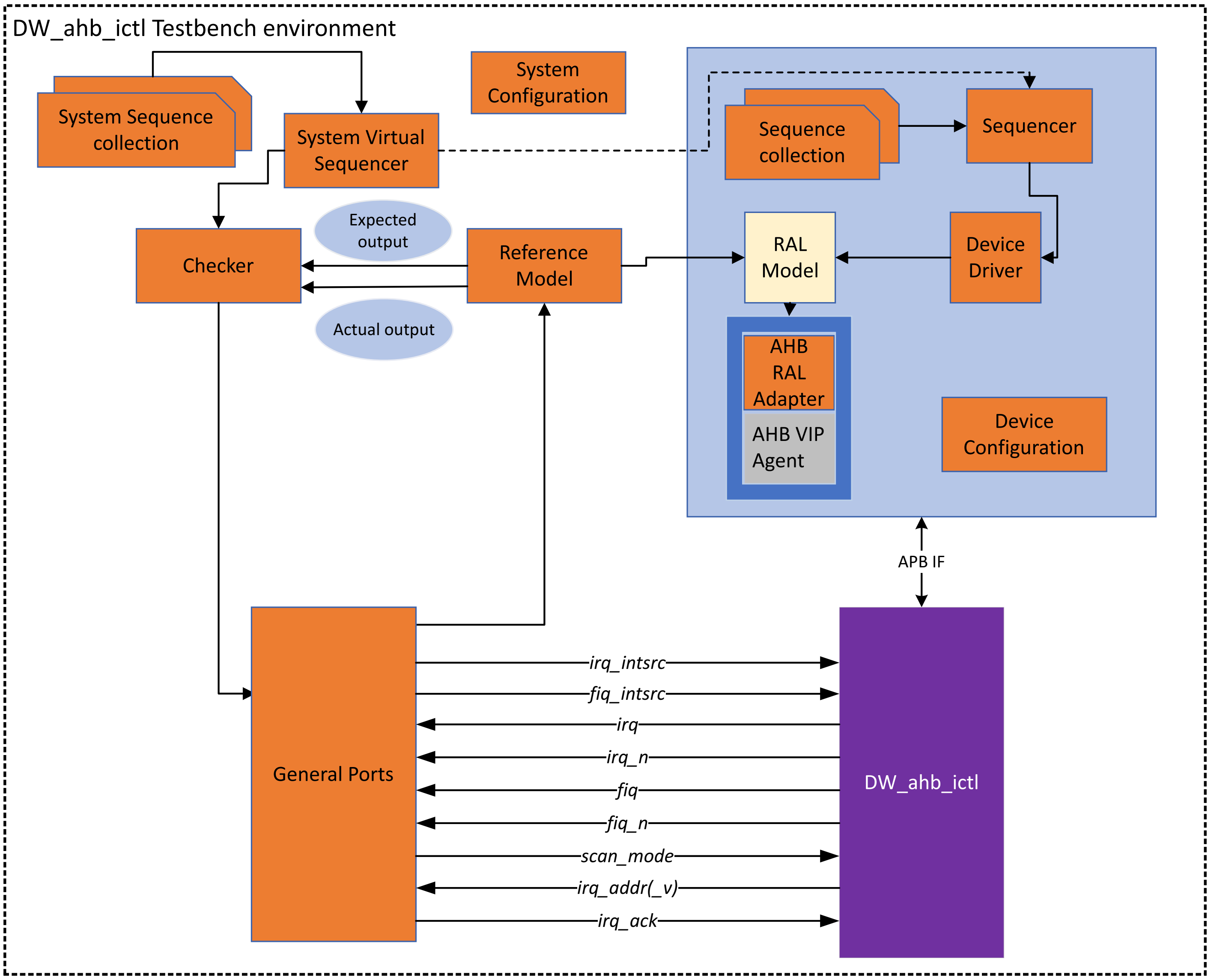
- "Verification Environment" on page 102
- "Testbench Directories and Files" on page 104
- "Packaged Testcases" on page 105

7.1 Verification Environment

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

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

Figure 7-1 DW_ahb_ictl UVM Verification Environment



The testbench consists of below elements:

- Testbench uses the standard SVT VIP for the protocol interfaces:
  - AHB VIP Interface for connecting master and slave ports of DUT.

- AHB VIP system environment (svt_ahb_system_env): AHB master agent and AHB slave agent for providing the VIP supported randomized transactions with bus instantiation on either side of the DUT.

- Testbench uses the custom components:
    - Device Agent

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

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

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

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

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

      Inside Device Agent we will have RAL model. RAL model is used for two purposes: first is to show (shadow) the values of registers inside DW_ahb_ictl and the second is checking whether the values of registers are correct upon reading. In order to facilitate that RAL model will have a two-way connection with the Reference model. Reference model will get the values from RAL to check whether behavior on General Ports is correct and will also be responsible for updates of the RAL, since Device agent is unable to see the activity that is happening on General Ports, where transaction can also change the values of registers in DW_apb_ictl (mostly status registers).

      The reason we use the RAL model is flexibility and reuse since if we change the adapter and VIP agent (interface we use) we can keep all the other logic.
    - AHB RAL Adapter

      AHB RAL Adapter converts higher level RAL Model Reads and Writes to AHB transactions, these transactions will be driven on the bus by AHB VIP Agent. By replacing this part with the different adapter we make this environment capable of being used by other interfaces.
    - AHB VIP Agent

      AHB VIP Agent drives transactions on AHB interface. By replacing AHB VIP Agent and AHB RAL Adapter with different components (such as AHB components) we can keep the other logic in Device Agent, and use it for that other interface.
    - General Ports

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

Write to irq_intsrc and fiq_intsrc inputs

Read irq, irq_n, firq, and firq_n

- ICT_HAS_FIQ = false:

  Write to irq_intsrc input

  Read irq and irq_n
- Scan Mode:

  Write to scan_mode input

General Ports also collects all the signals from the ports and sends them to the Reference Model.

- Vector Port

  Vector port potentially decreases IRQ service latency, as the processor does not need to initiate an AHB bus transaction in order to discover the vector address associated with the current highest-priority interrupt. Vector port exists only if ICT_ADD_VECTOR_PORT parameter is true.

  The Vector Port module is responsible to Read and Write the DUTs signals as follows:

  - ICT_ADD_VECTOR_PORT = true:

    Write to irq_ack input

    Read irq_addr, irq_n and irq_addr_v outputs
- System Virtual Sequencer

  The main responsibility of System Virtual Sequencer is to synchronize General Ports sequencer and RAL sequencer.
- Reference Model / Checker

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

  Based on the transactions received from the agent monitors (General Ports monitors) and values of registers in RAL, the Reference Model also calculates the expected values of DUT output signals. The actual DUT outputs, as well as the expected values, are then sent to the Checker. In the Checker comparing the expected values of DUT output signals against the values actually driven by the DUT is done.

7.2 Testbench Directories and Files

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

Table 7-1 shows the various directories and associated files:

Table 7-1 DW_ahb_ictl Testbench Directory Structure

Directory	Description
<configured workspace>/sim/testbench	Top level testbench module (test_top.sv) and the DUT to the testbench wrapper (dut_sv_wrapper.sv) exist in this folder
<configured workspace>/sim/testbench/env	Contains testbench files. For example scoreboard, sequences, VIP, environment, sequencers, and agents.

Table 7-1 DW_ahb_ictl Testbench Directory Structure

Directory	Description
<configured workspace>/sim/	Primarily contains the supporting files to compile and run the simulation. After the completion of the simulation, the log files are present here
<configured workspace>/sim/test_*	Contains individual test cases. After the completion of the simulation, the test specific log files and if applicable the waveform files are stored here

7.3 Packaged Testcases

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

The associated test cases shipped and their description is explained in the following Table 7-2:

Table 7-2 DW_ahb_ictl Test Description

Test Name	Test Description
test_100_reg_reset_bit_bash	This test builds the sequence `dw_ahb_ictl_reg_reset_bit_bash_virtual_sequence`, which in turn initiates `dw_ahb_ictl_reg_reset_bit_bash_sequence`. ■ Validates the register values after reset. ■ Bit bash for the bits of all the registers.
test_101_multi_sw_pol	This test builds the sequence `dw_ahb_ictl_multi_sw_polarity_virtual_sequence`. ■ Switches the interrupt polarity intermediately when the interrupt source of the respective interrupt is enabled. ■ This test is performed on multiple interrupts over various iterations.
test_102_multi_fiq_sw_pol	This test builds the sequence `dw_ahb_ictl_fiq_sw_polarity_virtual_sequence`. ■ Condition ICT_HAS_FIQ= 1. ■ Switches the interrupt polarity intermediately when the interrupt source of the respective interrupt is enabled. ■ This test is performed on multiple interrupts over various iterations.

Table 7-2 DW_ahb_ictl Test Description

Test Name	Test Description
test_103_multi_fiq_polarity	This test builds the sequence `dw_ahb_ictl_multi_fiq_polarity_virtual_sequence`. ■ Condition ICT_HAS_FIQ= 1. ■ This test perform enabling multiple interrupts over number of iterations and checking their status by reading FIQ_RAWSTATUS. ■ Follow the steps for every single interrupt: 1. Randomize which interrupt source to assert. 2. Start APB sequence to enable that interrupt source. 3. Start ICTL sequence which will assert that interrupt source. 4. Read the FIQ_RAWSTATUS to monitor the interrupt. 5. Further disable the interrupt source and again read the FIQ_RAWSTATUS.
test_104_multi_interrupt_vector	This test builds the sequence `dw_ahb_ictl_multi_interrupt_vector_virtual_sequence`. ■ Condition ICT_HAS_VECTOR =1. ■ Checks the functionality of interrupt vector. ■ Checks for multiple interrupts over various iterations simultaneously.
test_105_multi_vector_port	This test builds the sequence `dw_ahb_ictl_multi_vector_port_virtual_sequence`. ■ Condition ICT_HAS_VECTOR =1 and ICT_ADD_VECTOR_PORT =1. ■ Validates the vector port with respect to multiple interrupts simultaneously across various iterations.
test_106_multi_en_mask	This test builds the sequence `dw_ahb_ictl_multi_enable_maskng_virtual_sequence`. ■ Checks all interrupt handling registers. ■ Tests for multiple interrupts over various iterations, follow the steps for all selected interrupts. 1. Enable the selected interrupt source. 2. Assert the interrupt. 3. Read all the interrupt related registers. 4. Mask the interrupt by writing FIQ_MASKSTATUS register. 5. Read the FIQ_STATUS to check that interrupt is masked. 6. Read the FIQ_RAWSTATUS to check interrupt is active 7. Unmask the interrupt and repeat all the steps.

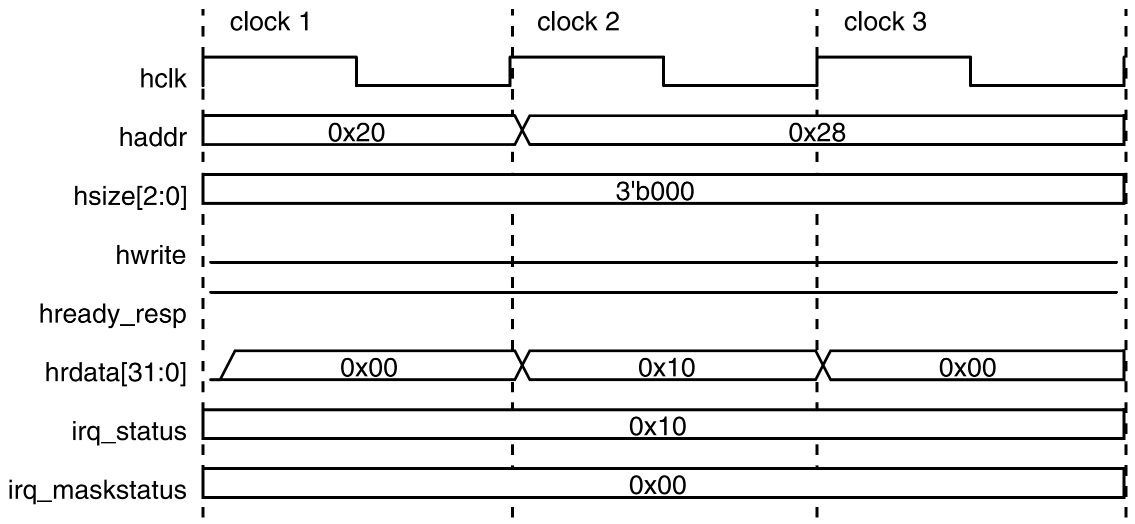
8 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment.

8.1 Read Accesses

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

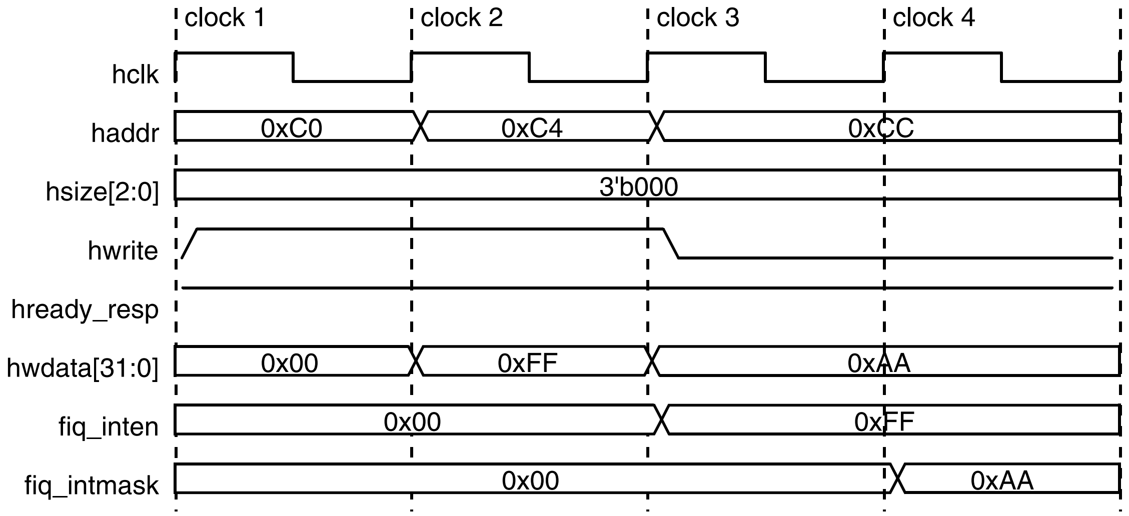
Figure 8-1 AHB Read



8.2 Write Accesses

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

Figure 8-2 AHB Write

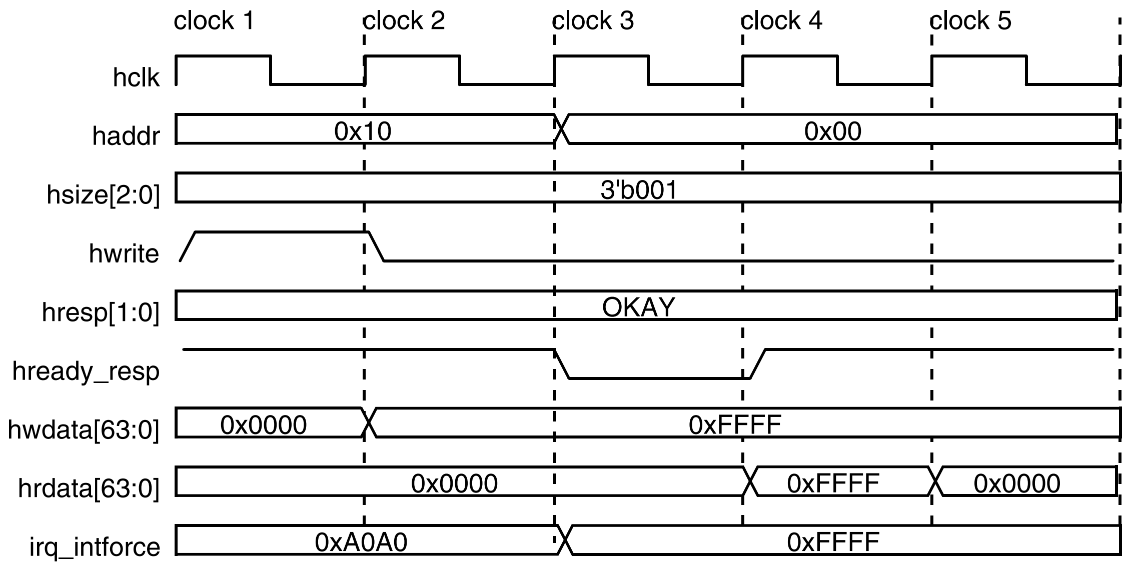


The operation of the AHB bus is pipelined, so while the write data for the first write is present on the bus for the slave to sample, the control for the next write is present on the bus for the slave to sample.

8.3 Consecutive Write-Read

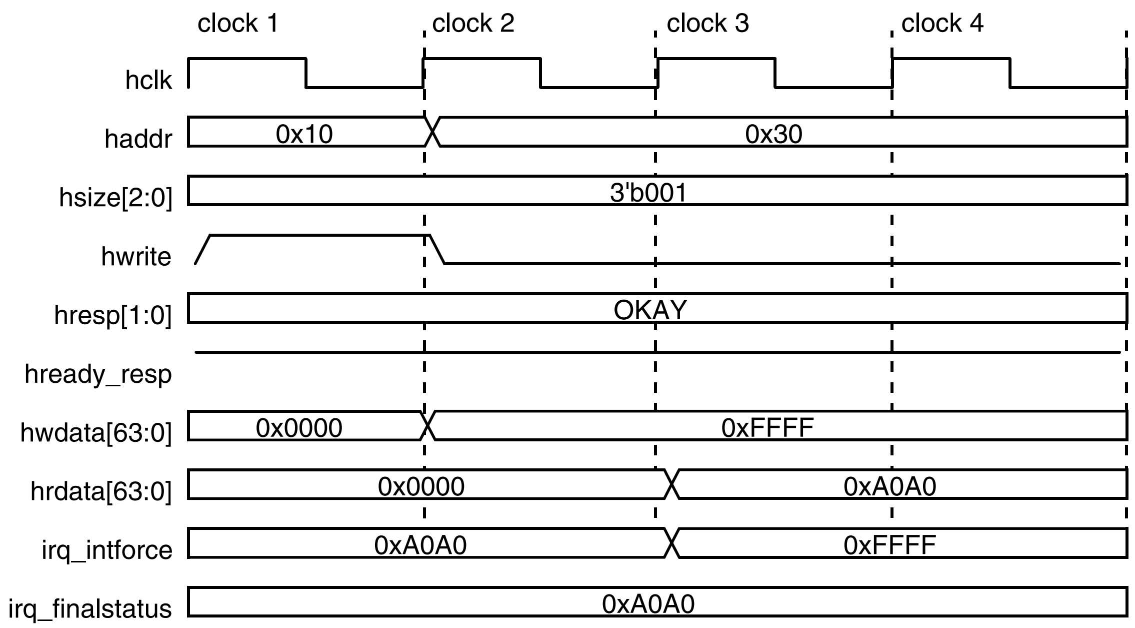
This is a specific case for the AHB slave interface. The AMBA specification says that for a read after a write to the same address, the newly written data must be read back, not the old data. To comply with this, the slave interface in the DW_ahb_ictl inserts a "wait state" when it detects a read immediately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

Figure 8-3 AHB Wait State Read/Write



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

Figure 8-4 AHB Consecutive Read/Write



8.4 Accessing Top-level Constraints

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

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

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

```
autocomplete_activity Synthesize
```

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

```
write_sdc <filename>
```

8.5 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.

8.5.1 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

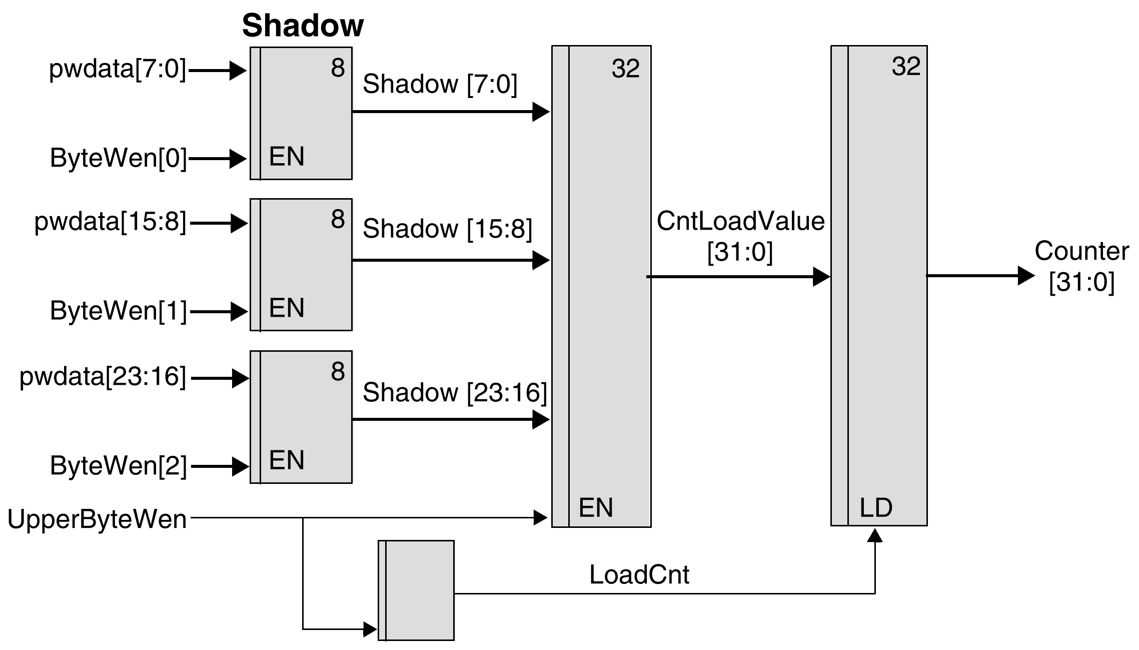
There are three relationship cases to be considered for the processor and peripheral clocks:

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

8.5.1.1 Identical Clocks

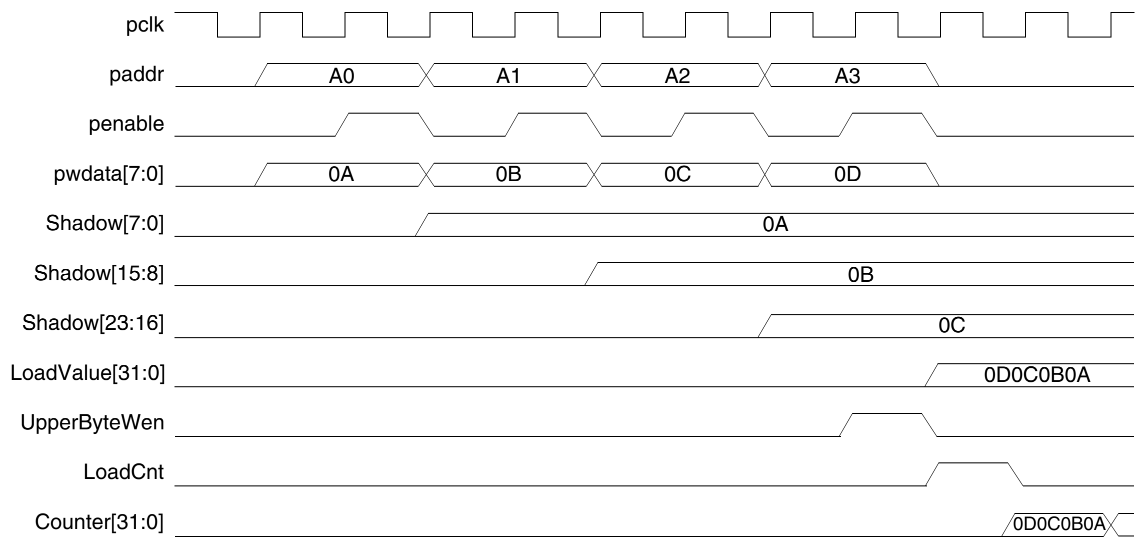
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

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.

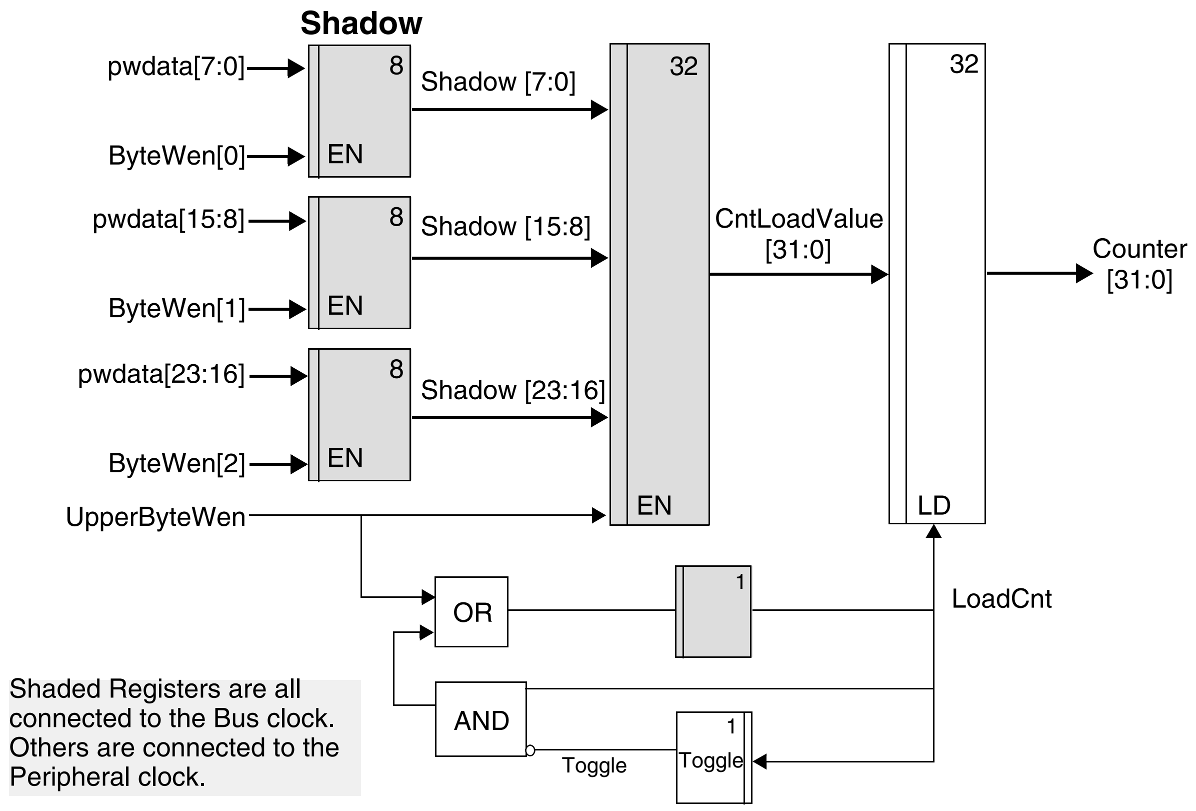
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this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

8.5.1.2 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.

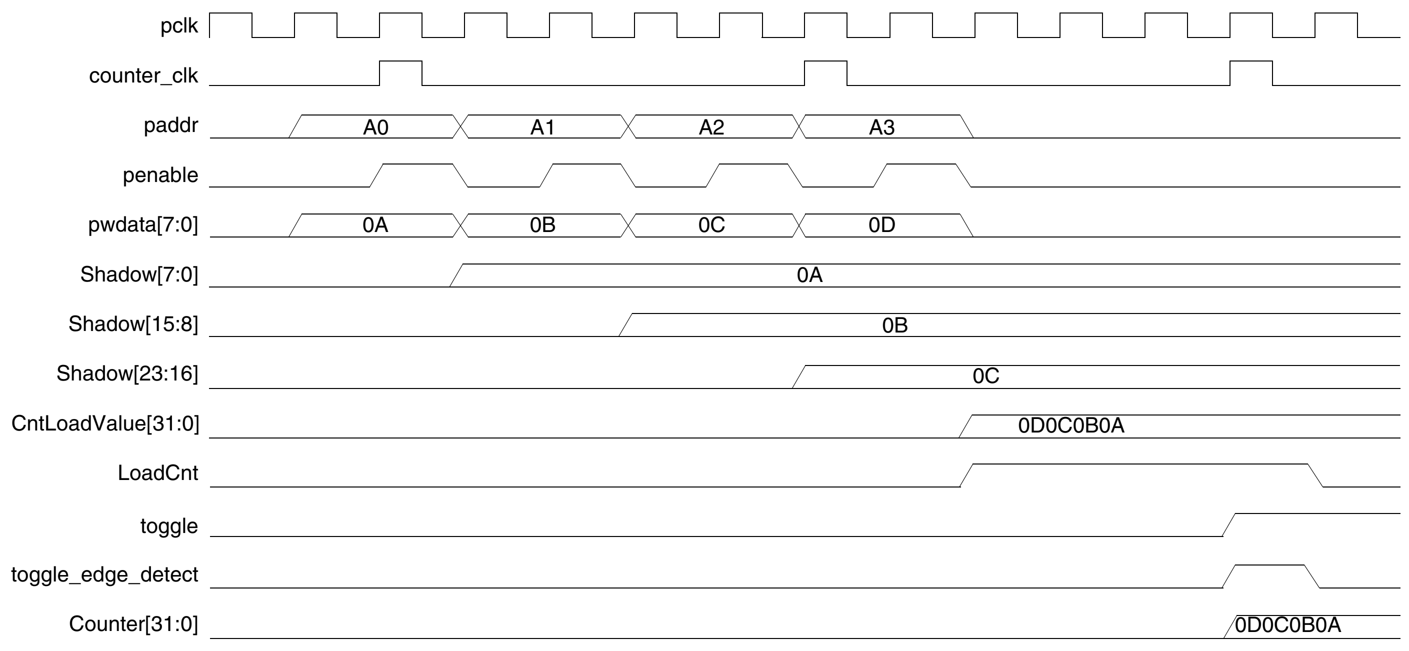
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

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.

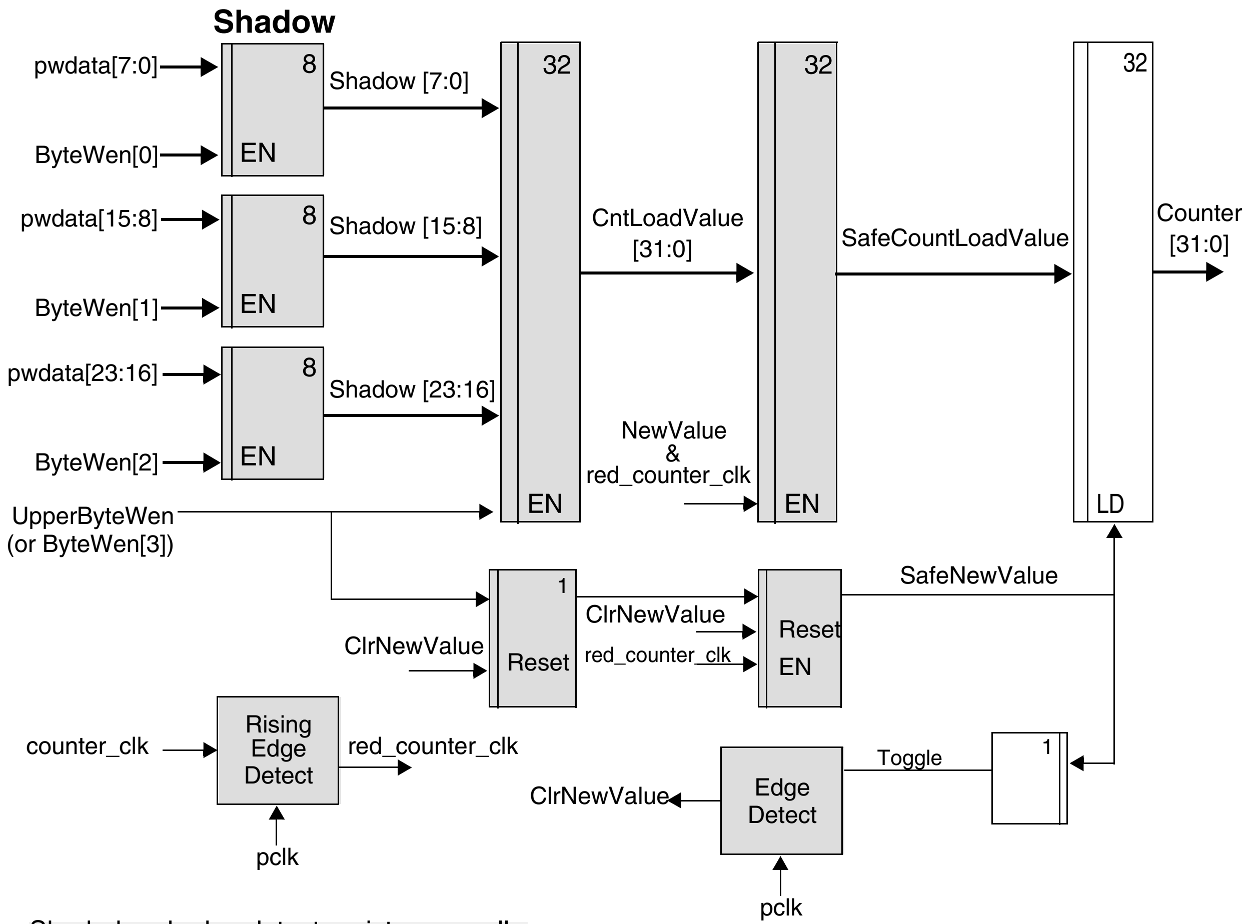
Figure 8-8 Coherent Loading – Synchronous Clocks



8.5.1.3 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



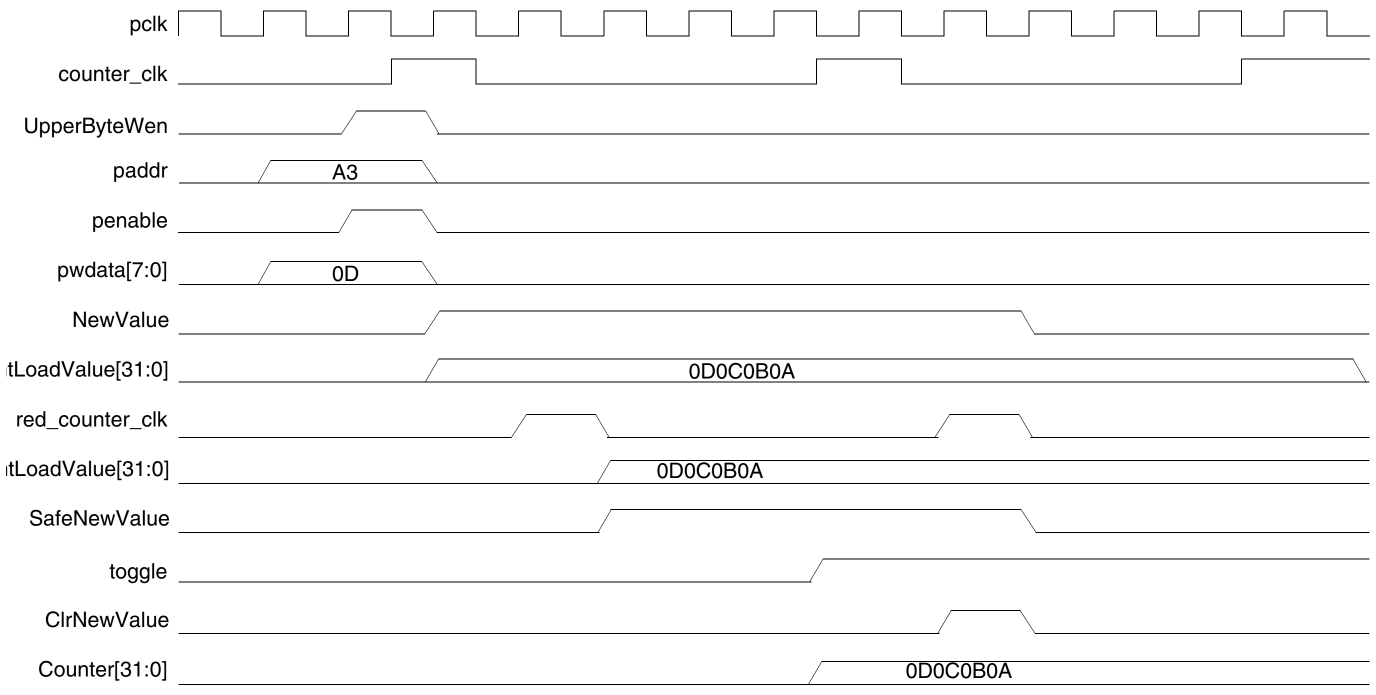
Shaded and edge detect registers are all connected to the Bus clock. Others are connected to the Peripheral clock.

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.

8.5.2 Reading Coherently

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

Table 8-2 Lower Byte Generation

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

Table 8-2 Lower Byte Generation

	Lower Byte Bus Width		
25 - 32	0	0	NCR

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

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

- Identical and/or synchronous
- Asynchronous

8.5.2.1 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

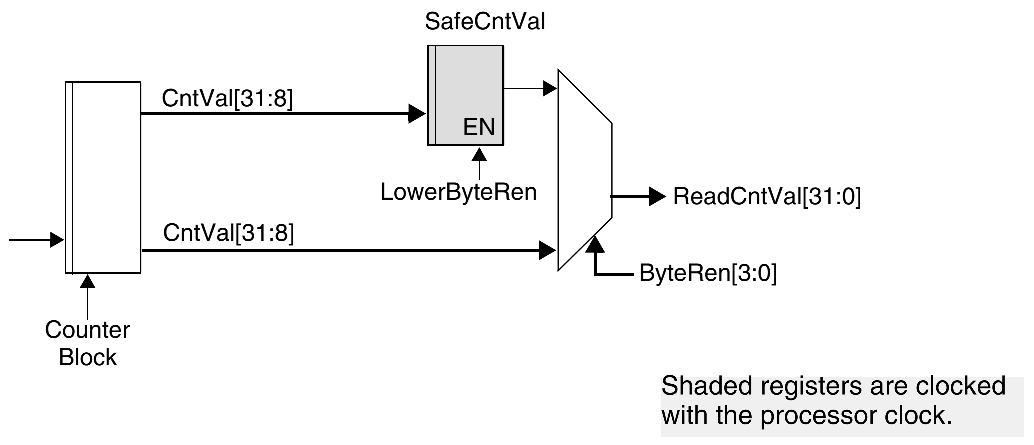
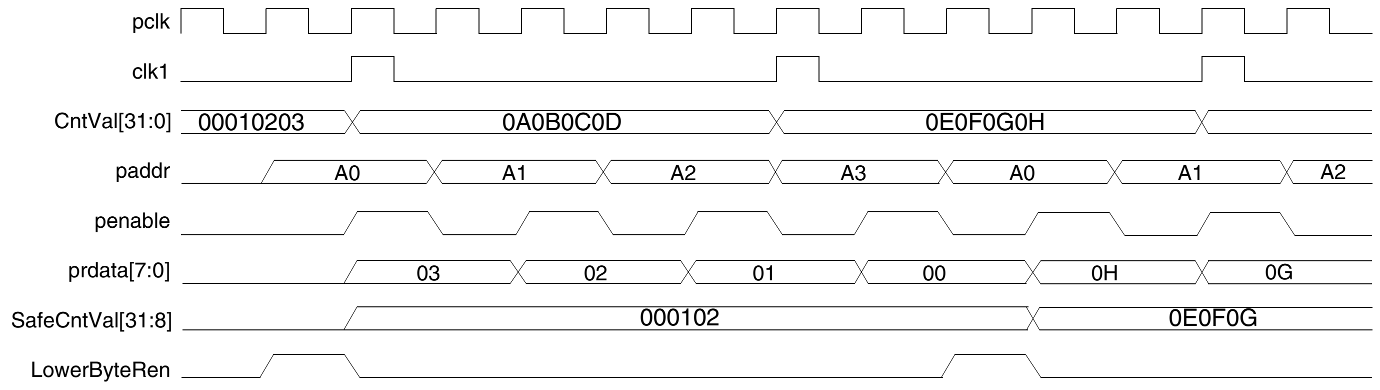


Figure 8-12 Coherent Registering – Synchronous Clocks



8.5.2.2 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.

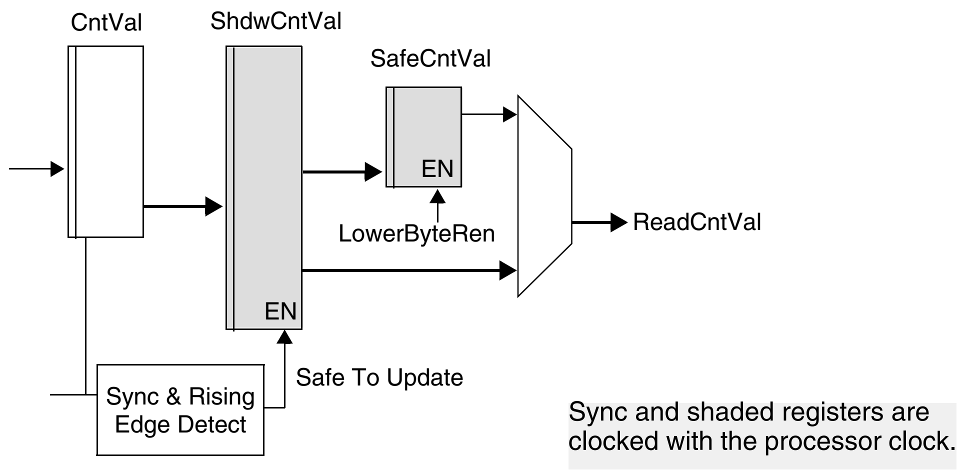
Note You must read LSB to MSB when the bus width is narrower than the counter width.

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


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

8.6 Performance

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

8.6.1 Power Consumption, Frequency, and Area and DFT Coverage

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

Table 8-3 Synthesis Results for DW_ahb_ictl

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	hclk = 300 MHz	2091	5 nW	0.030 mW	99.92	99.85	100
Typical Configuration 1 ICT_ADD_VECTOR_PORT = 1 ICT_HAS_PFLT = 1 ICT_HAS_VECTOR_USER = 1 ICT_HC_PRIORITIES = 0 ICT_IRQ_NUM = 64 ICT_READ_PRIORITY = 1	hclk = 300 MHz	15254	40 nW	0.188 mW	99.99	99.98	100
Typical Configuration 2 ICT_FIQ_NUM = 8 ICT_HAS_PFLT = 1 ICT_HAS_VECTOR_USER = 1 ICT_IRQ_NUM = 64 AHB_DATA_WIDTH = 256 ICT_IRQ_PLEVEL = 5	hclk = 300 MHz	12805	30 nW	0.177 mW	99.99	99.96	100

A
Internal Parameter Descriptions

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

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

Table A-1 Internal Parameters

Parameter Name	Equals To
ENCODED_AHB_DATA_WIDTH	{[~~function_of:~~ ::DW_ahb_ictl::encode_ahb_data_width AHB_DATA_WIDTH]}
ENCODED_ICT_FIQ_NUM	{[~~function_of:~~ ::DW_ahb_ictl::encode_ict_fiq_num ICT_FIQ_NUM]}
ENCODED_ICT_IRQ_NUM	{[~~function_of:~~ ::DW_ahb_ictl::encode_ict_irq_num ICT_IRQ_NUM]}
FALSE	1'b0
HRESP_WIDTH	(AHB_SCALAR_HRESP ==1 ? 1 : 2)
ICT_HAS_VECTOR	(((ICT_HAS_PFLT == 1)) ? ICT_HAS_VECTOR_USER : 0)
ICTL_VERSION_ID	32'~~h3231342a~~h3231352a
TRUE	1'b1

B
Basic Core Module (BCM) Library

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

This appendix contains the following sections:

- "BCM Library Components" on page 123
- "Synchronizer Methods" on page 123

B.1 BCM Library Components

Table B-1 describes the list of BCM library components used in DW_ahb_ictl.

Table B-1 BCM Library Components

BCM Module Name	BCM Description	DWBB Equivalent
DW_ahb_ictl_bcm01	Minimum/Maximum Value	DW_minmax
DW_ahb_ictl_bcm21	Single Clock Data Bus Synchronizer	DW_sync

B.2 Synchronizer Methods

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

This section contains the following sections:

- "Synchronizers Used in DW_ahb_ictl" on page 124
- "Synchronizer 1: Simple Double Register Synchronizer (DW_ahb_ictl)" on page 124

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

B.2.1 Synchronizers Used in DW_ahb_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_ahb_ictl are listed and cross referenced to the synchronizer type in Table B-2. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

Table B-2 Synchronizer used in DW_ahb_ictl

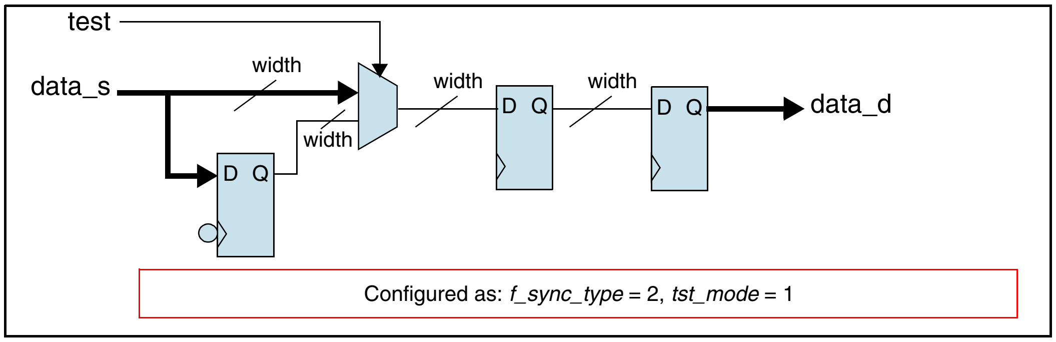
Synchronizer Module File	Synchronizer Type and Number
DW_ahb_ictl_bcm21.v	Synchronizer 1: Simple Multiple Register Synchronizer

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

B.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_ahb_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 B-1) both using positive edge of clock.

Figure B-1 Block Diagram of Synchronizer 1 With Two Stage Synchronization (Both Positive Edge)



C
Glossary

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

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

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