DesignWare Cores DesignWare DW_axi_dmac Databook

DW_axi_dmac – A415-0

Version 1.02a
February 2018

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Third-Party Links

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

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

www.synopsys.com

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.00a-ea01 onward.

Date	Version	Description
February 2018	1.02a	**Added:** ■ “AXI Unaligned Transfer Support” on page 69 ■ “Context Sensitive Low Power Option” on page 94 ■ “Asynchronous Hardware Handshake Support” on page 59 ■ “Single Arbiter Scheme” on page 30 ■ “Multi-Arbiter Scheme” on page 32 **Updated:** ■ Release version and date ■ Synthesis Results in “Performance” on page 477 ■ “Signal Descriptions” on page 105, “Parameter Descriptions” on page 99, and “Register Descriptions” on page 159 auto-extracted from the RTL with change bars **Removed:** ■ Removed Chapter 2 Building and Verifying a Component or Subsystem from the databook and added the contents in the newly created user guide ■ Running Leda on Generated Code with coreConsultant section ■ Removed all instances of Leda

Date	Version	Description
March 2016	1.01a	■ Revision version change for 2016.03a release ■ Added “Running SpyGlass® Lint and SpyGlass® CDC” on page 28 ■ Added “Running SpyGlass on Generated Code with coreAssembler” on page 35 ■ “Signal Descriptions” on page 105, “Parameter Descriptions” on page 99, and “Register Descriptions” on page 159 auto-extracted from the RTL ■ Added “Internal Parameter Descriptions” on page 465 ■ Removed “Narrow transfers” from “Unsupported Features” on page 19 ■ Modified gate count in Table 9-2 Area Results for DW_axi_dmac with tsmc28nm and DMAX_CHx_MULTI_BLK_TYPE set to 1 ■ Added Appendix A, “Synchronizer Methods” ■ Updated area and power numbers in “Performance” on page 477 ■ Added “Programming Flow for Single Block Transfer” on page 470
October 2014	1.00a	■ 2014.10a GA release ■ Included these sections in the “Verification” chapter: - “Overview of SV-UVM Tests” - “Overview of DW_axi_dmac Testbench”
August 2014	1.00a-ea04	Added the Clock Parameters block in “Signal Descriptions” on page 105.
July 2014	1.00a-ea03	■ Modified the Integration Considerations chapter ■ Modified description for the DMAX_LLI_ENDIAN_SELECTION_PIN_EN parameter
June 2014	1.00a-ea02	■ Modified address range for the Undefined Register Space in “Parameter Descriptions” on page 99. ■ Added two configuration parameters: - DMAX_MSTIF1_OSR_LMT - DMAX_MSTIF2_OSR_LMT ■ Modified reset values for the following registers: - DMAC_CommonReg_IntSignal_EnableReg - DMAC_CommonReg_IntStatus_EnableReg - CHx_IntStatus_EnableReg - CHx_IntSignal_EnableReg
April 2014	1.00a-ea01	■ Changing the Behaviour of Abnormal Channel Abort ■ Channel Abort feature now supported, removing all the Notes related to that. ■ Added a new section in Debug Interface ■ Added Description in section CHx_IntStatusReg ■ Removing the Note from Software Handshaking chapter
March 2014	1.00a-ea00	Initial release

Preface

This databook provides information that you need to interface the DesignWare AXI Central Direct Memory Access (DMA) Controller, referred to as DW_axi_dmac throughout the remainder of this databook. DW_axi_dmac is a highly configurable, highly programmable, high-performance multi-master multi-channel DMA Controller with AXI as the bus interface for data transfer. This component conforms to the AMBA Specification, Revision 2.0 and AMBA AXI Protocol Specification, Version 3.0, and 4.0 from ARM.

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

Organization

The chapters of this databook are organized as follows:

- Chapter 1, “Product Overview” provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, “Functional Description” describes the functional operation of the DW_axi_dmac.
- Chapter 3, “Parameter Descriptions” identifies the configurable parameters supported by the DW_axi_dmac.
- Chapter 4, “Signal Descriptions” provides a list and description of the DW_axi_dmac signals.
- Chapter 5, “Register Descriptions” describes the programmable registers of the DW_axi_dmac.
- Chapter 6, “Internal Parameter Descriptions” provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals, Registers, and Parameters chapters.
- Chapter 7, “Programming the DW_axi_dmac” provides information needed to program the configured DW_axi_dmac.
- Chapter 8, “Verification” provides information on verifying the configured DW_axi_dmac.
- Chapter 9, “Integration Considerations” includes information you need to integrate the configured DW_axi_dmac into your design.
- Appendix A, “Synchronizer Methods” documents the synchronizer methods (blocks of synchronizer functionality) used in DW_axi_dmac to cross clock boundaries.
- Appendix B, “Glossary” provides a glossary of general terms.

Related Documentation

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

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

Web Resources

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

Customer Support

To obtain support for your product:

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

    File > Build Debug Tar-file

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

    Go to http://solvnet.synopsys.com/EnterACall and click **Open A Support Case** to enter a call. Provide the requested information, including:
    - **Product:** DesignWare Library IP
    - **Sub Product 1:** AMBA
    - **Product Version:** *<product version number>*
    - **Problem Type:**

- **Issue Severity:**
- **Problem Title:** Provide short title of the problem
- **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

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

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

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

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

1 Product Overview

This chapter provides a basic overview of the DW_axi_dmac, which is a highly configurable, highly programmable, high performance, multimaster multichannel DMA Controller with AXI as the bus interface for data transfer.

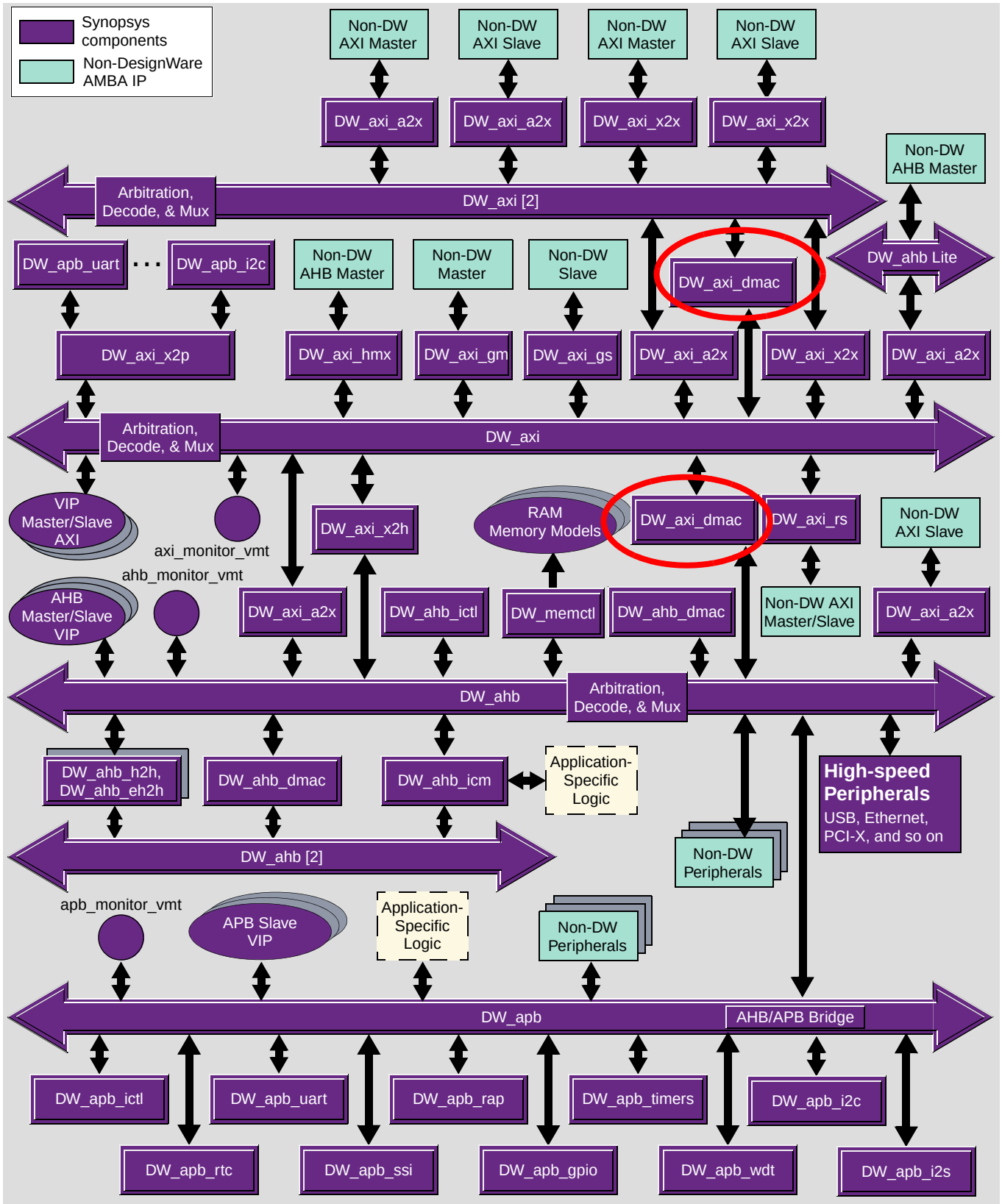
1.1 DesignWare System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA version 3.0-compliant and 4.0-compliant AXI (Advanced eXtensible Interface) components.

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

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

Figure 1-1 Example of DW_axi_dmac in a Complete System



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

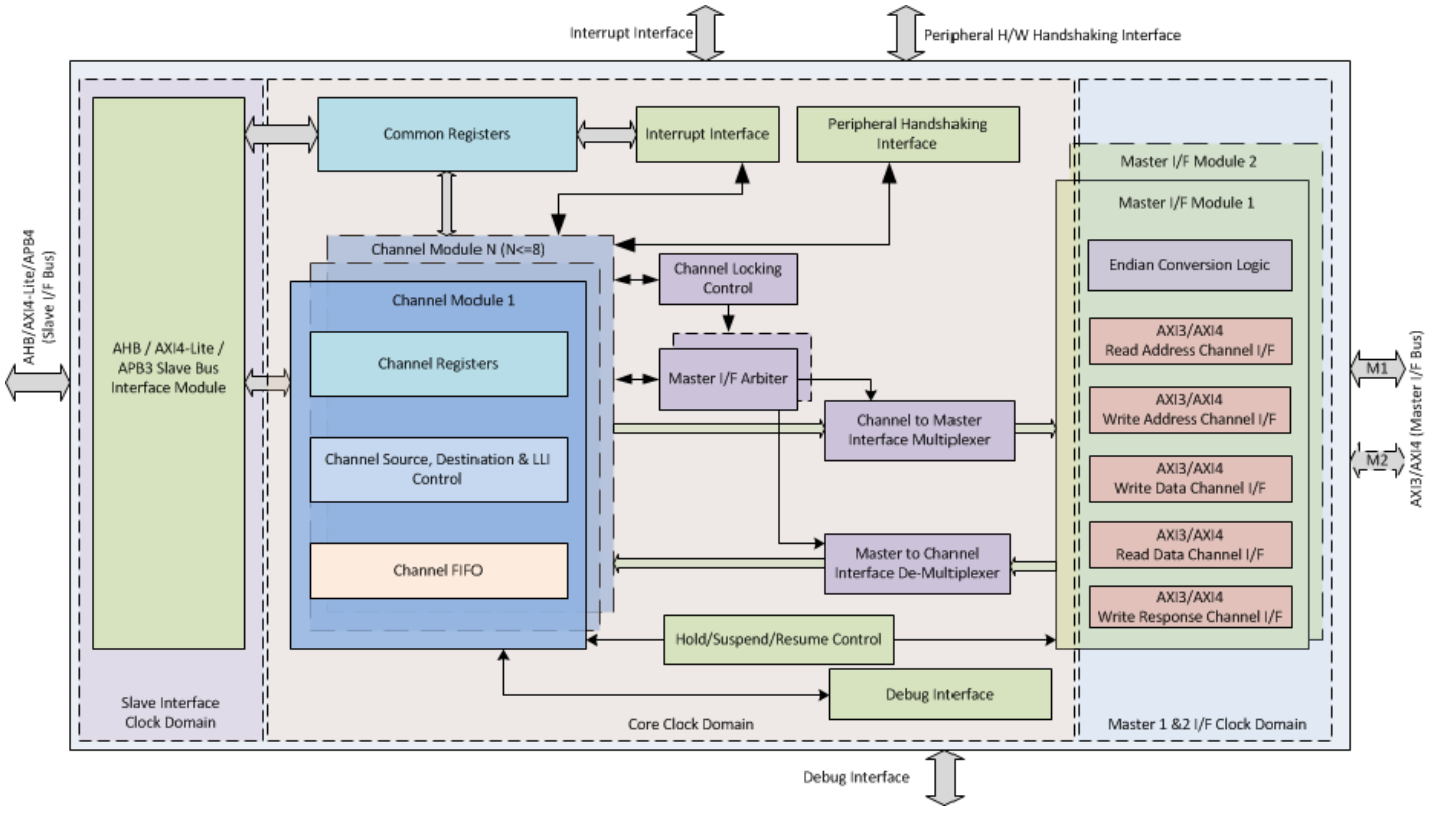
1.2 General Product Description

The Synopsys DW_axi_dmac conforms to the AMBA Specification, Revision 2.0 and the AMBA AXI Protocol Specification, Version 2.0 from ARM.

1.2.1 DW_axi_dmac Block Diagram

shows the top-level architecture of the DW_axi_dmac.

Figure 1-2 Top-Level Architecture of the DW_axi_dmac



1.3 Features

The DW_axi_dmac supports the following features:

1.3.1 General Features

- Independent core, slave interface and master interface clocks
- Shutting down the slave interface clock
  - Master can shut down the slave interface clock when the slvif_busy output is de-asserted.
  - Master must restart the clock before trying to access the slave interface again.
- Individually shutting down the master interface clocks when no peripheral is active
- Up to eight channels, one per source and destination pair, configurable in coreConsultant
- Data transfers in one direction only (each channel is unidirectional)
- Up to two AXI master interfaces, configurable in coreConsultant
  - Two master interfaces for multilayer support
  - Multiple AXI masters increase bus performance by allowing direct connection of peripherals on different AXI interconnects
  - Support for different ACLK on different AMBA layers
- Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral DMA transfers
- AMBA 3 AXI / AMBA 4 AXI-compliant master interface
- AHB / AXI4-Lite / APB3 slave interface for programming the DMA controller
  - Only AHB supported in this version
  - AHB slave interface supports only SINGLE transfers (hburst = 3'b000).
- AXI master data bus width up to 512 bits (for both AXI master interfaces), configurable in coreConsultant
- Endian mode can be selected statically or dynamically for AXI master interfaces, configurable in coreConsultant
- Input pin to dynamically select endian scheme
- Independent control for endian scheme of linked list access on master interfaces, configurable in coreConsultant
- Optional identification register, configurable in coreConsultant
- Channel locking support

  Supports locking of the internal channel arbitration for the master bus interface at different transfer hierarchy
- DMAC status indication outputs

  Idle / busy indication

- DMA hold function
- Output pin indicates the last write transfer at DMA transaction level
- Multiple levels of DMA transfer hierarchy

  DMA transfer split into transaction, block, and complete DMA transfer levels
- Support for AXI unaligned transfers
- Support for Context Sensitive Low Power Option

1.3.2 Channel Buffering

- Single FIFO per channel
- FIFO depth, configurable in coreConsultant
- Automatic packing/unpacking of data to fit FIFO width

1.3.3 Channel Control

- Programmable transfer type for each channel (memory-to-memory, memory-to-peripheral, peripheral-to-memory and peripheral-to-peripheral)
- Single or multiple DMA transactions
- Programmable multiple transaction size for each channel
- Programmable maximum AMBA burst transfer size for each channel, configurable in coreConsultant
- Channel disabling without data loss
- Channel suspend and resume
- Programmable channel priority
- Locking of internal channel arbitration for master bus interface at different transfer hierarchy
- Programmable multiblock transfer using linked list, contiguous address, auto reload, and shadow register methods
- Dynamic extension of linked list
- Independent configuration of SRC/DST multiblock transfer type
- Multiple state machines, one for each channel SRC and DST
- Separate state machines for data and LLI access
- Control signals, such as cache and protection, programmable per DMA block
- Programmable transfer length (block length)
- Error status register to ease debugging during error events

1.3.4 Flow Control

- Programmable flow control at DMA transfer level
  - If the size of the block transfer is known prior to DMA initialization, the DMA controller is a flow controller at the DMA block transfer level.
  - If the size of the DMA block transfer is unknown prior to the DMA initialization Peripheral, either source or destination is the flow controller for undefined length (demand mode) DMA block transfers.

1.3.5 Handshaking Interface

- Programmable software and hardware handshaking interfaces for non-memory peripherals
- Up to 16 hardware handshaking interfaces / peripherals, configurable in coreConsultant
- Enabling / disabling of individual handshake interfaces
- Programmable mapping between peripherals and channels; many-to-one mapping with only one peripheral active at a time
- Memory mapped registers to control DMA transfer in software handshaking mode

1.3.6 Interrupt Outputs

- Combined and separate interrupt outputs
- Interrupt generation on
  - DMA transfer completion
  - Block transfer completion
  - Single or multiple transaction completion
  - Error condition
  - Channel suspend or disable
- Interrupt enabling and masking

1.3.7 Bus Interface

- AMBA 3 AXI and AMBA 4 AXI protocols for master interface and AHB, AXI4-Lite, and APB 3 protocols for slave interface
- Data bus width up to 512 bits for master interface, configurable in coreConsultant
- Outstanding transactions on master interface
- Setting outstanding transaction limit per channel on the master interface
- Configurable AXI transfer width
- Out-of-order transaction support for different channels connected on same master interface

  Transactions of a particular channel are always initiated in order.

- Increment and fixed address transfers on master interface
- Source and destination data transfer addresses; must be aligned to respective transfer widths
- Data bus width of 32 / 64 bits for slave interface, configurable in coreConsultant
- Transfer size (width) used for slave interface; must be same as data bus width

1.3.8 Unsupported Features

The following features are not supported in this release:

- AXI 3 locked transfers (bus locking)
- Bus locking on master interface in AXI 3 mode
- Wrap address transfers

1.4 Terminology

The following terms are concise definitions of the DMA concepts used throughout this document.

- **Source peripheral** – Device on an AXI layer from which the DW_axi_dmac reads data. The DW_axi_dmac then stores the data in the channel FIFO. The source peripheral teams up with a destination peripheral to form a channel.
- **Destination peripheral** – Device to which the DW_axi_dmac writes the stored data from the FIFO (data is previously read from the source peripheral).
- **Memory** – Source or destination that is always ready for a DMA transfer and does not require a handshaking interface to interact with the DW_axi_dmac.
- **Channel** – Read / write data path between a source peripheral on one configured AXI layer and a destination peripheral on the same or a different AXI layer that occurs through the channel FIFO. If the source peripheral is not memory, then a source handshaking interface is assigned to the channel. If the destination peripheral is not memory, then a destination handshaking interface is assigned to the channel. Source and destination handshaking interfaces can be assigned dynamically by programming the channel registers.
- **Master interface** – DW_axi_dmac is a master on the AXI bus, reading data from the source and writing it to the destination over the AXI bus. It is possible to have up to two master interfaces, so that up to two independent source and destination channels can operate simultaneously. Each channel has to arbitrate for the master interface. If the source and destination peripherals reside on different AXI layers, there must be multiple master interfaces.
- **Slave interface** – The AHB / AXI4-Lite / APB3 interface over which the DW_axi_dmac is programmed. The slave interface can be on the same layer as any of the master interfaces, or it can be on a separate layer.
- **Handshaking interface** – A set of signals or software registers that conform to a protocol to perform a handshake between the DW_axi_dmac and the source or destination peripheral. A handshake helps to control the transfer of a single or burst transaction between the DW_axi_dmac and the peripheral. This interface is used to request, acknowledge, and control a DW_axi_dmac transaction. A channel can receive a request through one of two types of handshaking interface: hardware or software.

- **Hardware handshaking interface** – Uses hardware signals to control the transfer of a single or burst transaction between the DW_axi_dmac and the source or destination peripheral. The simple use of the hardware handshaking interface is when the interrupt line from the peripheral is tied to the “dma_req” input of the hardware handshaking interface; other interface signals are ignored.
- **Software handshaking interface** – Uses software registers to control transferring a single or burst transaction between the DW_axi_dmac and the source or destination peripheral. In this mode, no special DW_axi_dmac handshaking signals are needed on the I/O of the peripheral. This mode is useful for interfacing an existing peripheral to the DW_axi_dmac without modifying it.

- **Flow controller** – Device (either the DW_axi_dmac, or a source or destination peripheral) that determines the length of a DMA block transfer and terminates it. If the length of a block is known before enabling the channel, then the DW_axi_dmac should be programmed as the flow controller. If the length of a block is not known prior to enabling the channel, the source or destination peripheral should terminate the block transfer. In this mode, the peripheral (source/destination) is the flow controller.
- **Transfer hierarchy** – Transfers are split into a maximum of four levels: DMA transfer level, block transfer level, transaction level, and AXI transfer level. This is done to minimize the effect of the scenario where the channel is granted to a particular set of peripherals, but a peripheral does not have enough data to transfer continuously. In such a scenario, the channel can not be given to any other peripheral, which reduces the performance.

  Figure 1-3 illustrates the hierarchy between DMA transfers, block transfers, transactions (single or burst), and AMBA AXI transfers (single or burst) for non-memory peripherals.

  Figure 1-4 illustrates the hierarchy between DMA transfers, block transfers, and AMBA AXI transfers (single or burst) for memory peripherals.

  Note There is no transaction level for memory peripherals, as a memory is assumed to be always ready for data transfer.

Figure 1-3 DMA Transfer Hierarchy for Non-Memory Peripherals

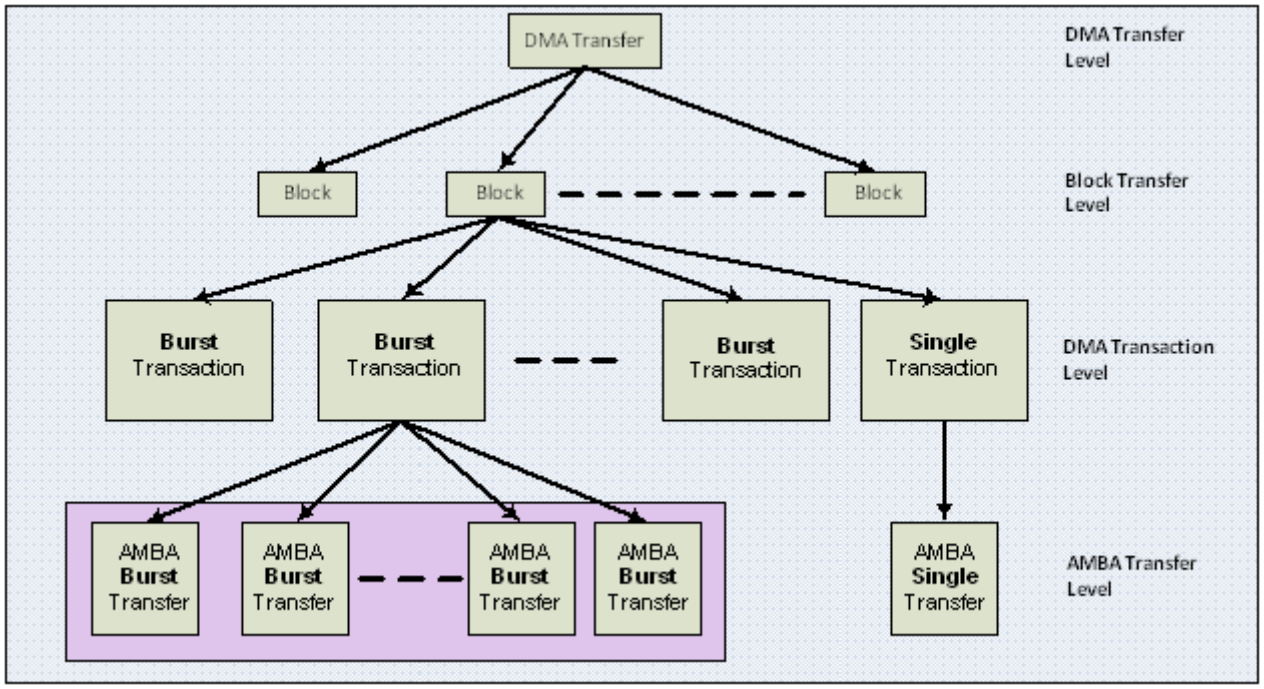
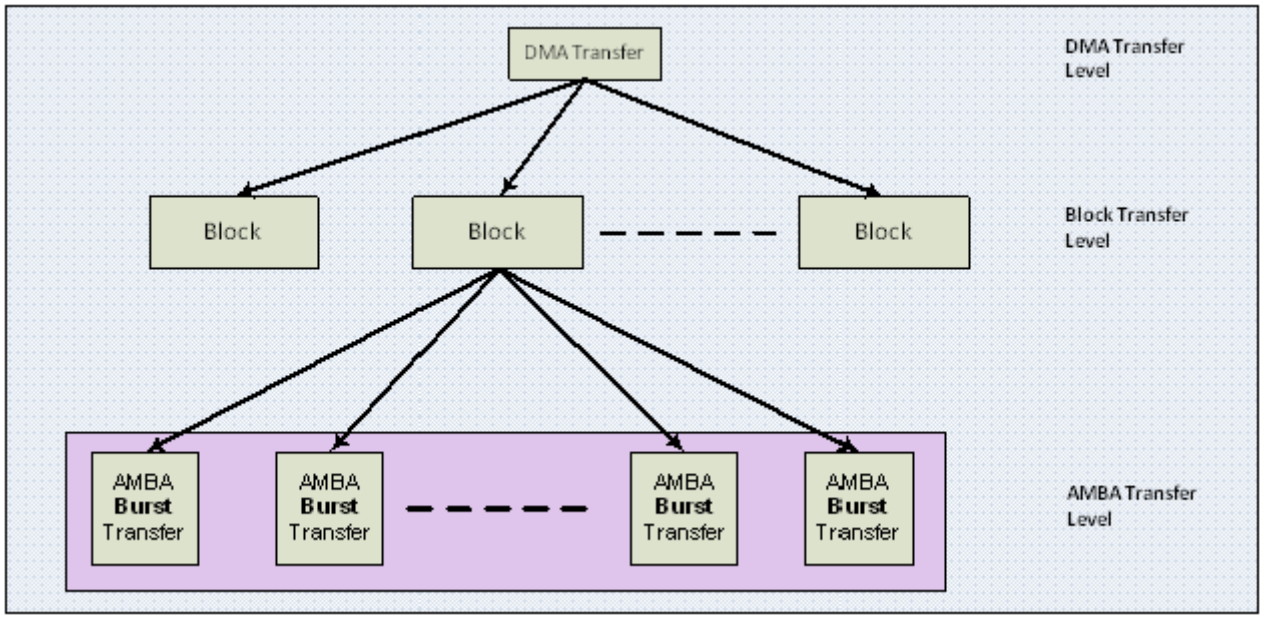


Figure 1-4 DMA Transfer Hierarchy for Memory Peripherals



- **Transaction** – Basic unit of a DW_axi_dmac transfer, as determined by either the hardware or the software handshaking interface. A transaction is relevant only for transfers between the DW_axi_dmac and a source or destination peripheral if the peripheral is a non-memory device. There are two types of transactions:
  - **Single transaction** – Length of a single transaction is always 1 and it is converted to an INCR AXI transfer of burst length 1.
  - **Burst transaction** – Length of a burst transaction is programmed into the DW_axi_dmac. A burst transaction is converted into a sequence of AXI burst transfers. The burst transaction length is under program control and normally bears some relationship to the FIFO sizes in the DW_axi_dmac and in the source and destination peripherals.
- **Block** – A block of DW_axi_dmac data, the amount of which is the block length and is determined by the flow controller. For transfers between the DW_axi_dmac and memory, a block is broken directly into a sequence of burst transfers. For transfers between the DW_axi_dmac and a non-memory peripheral, a block is broken into a sequence of DW_axi_dmac transactions. These are in turn broken into a sequence of AXI transfers.
- **DMA transfer** – Software controls the number of blocks in a DW_axi_dmac transfer. Once the DMA transfer has completed, the hardware within the DW_axi_dmac disables the channel and can generate an interrupt to signal the DMA transfer completion. The channel can then be reprogrammed for a new DMA transfer.
  - **Single-block DMA transfer** – Consists of a single block.
  - **Multi-block DMA transfer** – A DMA transfer may consist of multiple DMA blocks. Multi-block DMA transfers are supported through block chaining (linked list pointers), auto-reloading of channel registers, shadow registers, and contiguous blocks. The source and destination can independently select which method to use.
    - **Linked lists (block chaining)** – A linked list pointer (LLP) points to the location in system memory where the next linked list item (LLI) exists. The LLI is a set of registers that describes the next block (block descriptor) and an LLP register. The DW_axi_dmac fetches the LLI at the beginning of every block when block chaining is enabled. LLI access always uses the burst size (arsize / awsize) that is the same as the data bus width and cannot be changed or programmed to anything other than this. Burst length (awlen / arlen) is chosen based on the data bus width so that the access does not cross one complete LLI structure of 64 bytes. DW_axi_dmac fetches the entire LLI (40 bytes) in one AXI burst, if the burst length is not limited by other settings.
    - **Auto-reloading** – DW_axi_dmac automatically reloads the channel registers at the end of each block to the value that was set when the channel was first enabled.
    - **Contiguous blocks** – Address between successive blocks is selected to be a continuation from the end of the previous block.
    - **Shadow register** - DW_axi_dmac automatically loads the channel registers from the contents of shadow register set at the end of each block. Software can program the shadow registers with the values corresponding to the next block transfer when the current block transfer is in progress.
- **Channel locking** – Software can program a channel to keep the AXI master interface by locking arbitration of the master bus interface for the duration of a DMA transfer, block transfer, or transaction (single or burst).

1.5 Standards Compliance

The Synopsys DW_axi_dmac conforms to the AMBA Specification, *AMBA Specification, Revision 2.0* from ARM. Readers are assumed to be familiar with this specification.

1.6 Verification Environment Overview

The DW_axi_dmac must support an extensive verification environment, which sets up and invokes the selected simulation tool to execute tests that verify the functionality of the configured component and allows you to analyze the results of the simulation.

"Verification" on page 473 discusses the specific procedures for verifying the DW_axi_dmac.

1.7 Licenses

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

1.8 Where To Go From Here

At this point, you may want to get started working with the DW_axi_dmac 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_axi_dmac component, see "Overview of the coreConsultant Configuration and Integration Process" on page 26.

For more information about implementing your DW_axi_dmac component within a DesignWare subsystem using coreAssembler, see "Overview of the coreAssembler Configuration and Integration Process" on page 32.

2 Functional Description

This chapter describes the functional details of the DW_axi_dmac component. DW_axi_dmac is a highly configurable, highly programmable, high-performance multi-master multi-channel DMA Controller with AXI as the bus interface for data transfer.

This chapter included the following topics:

2.1 Data Flow

Figure 2-1 shows the flow of data in the DW_axi_dmac in a scenario when the source and destination peripherals are on the same AXI layer, while Figure 2-2 shows the flow of data when the source and destination peripherals are on different AXI layers. In both scenarios, the source peripheral uses a hardware handshaking interface and the destination peripheral uses a software handshaking interface. If these peripherals are memory, handshaking interfaces are not used.

Figure 2-1 DW_axi_dmac Flow Diagram when Source and Destination Peripherals on Same AXI Layer

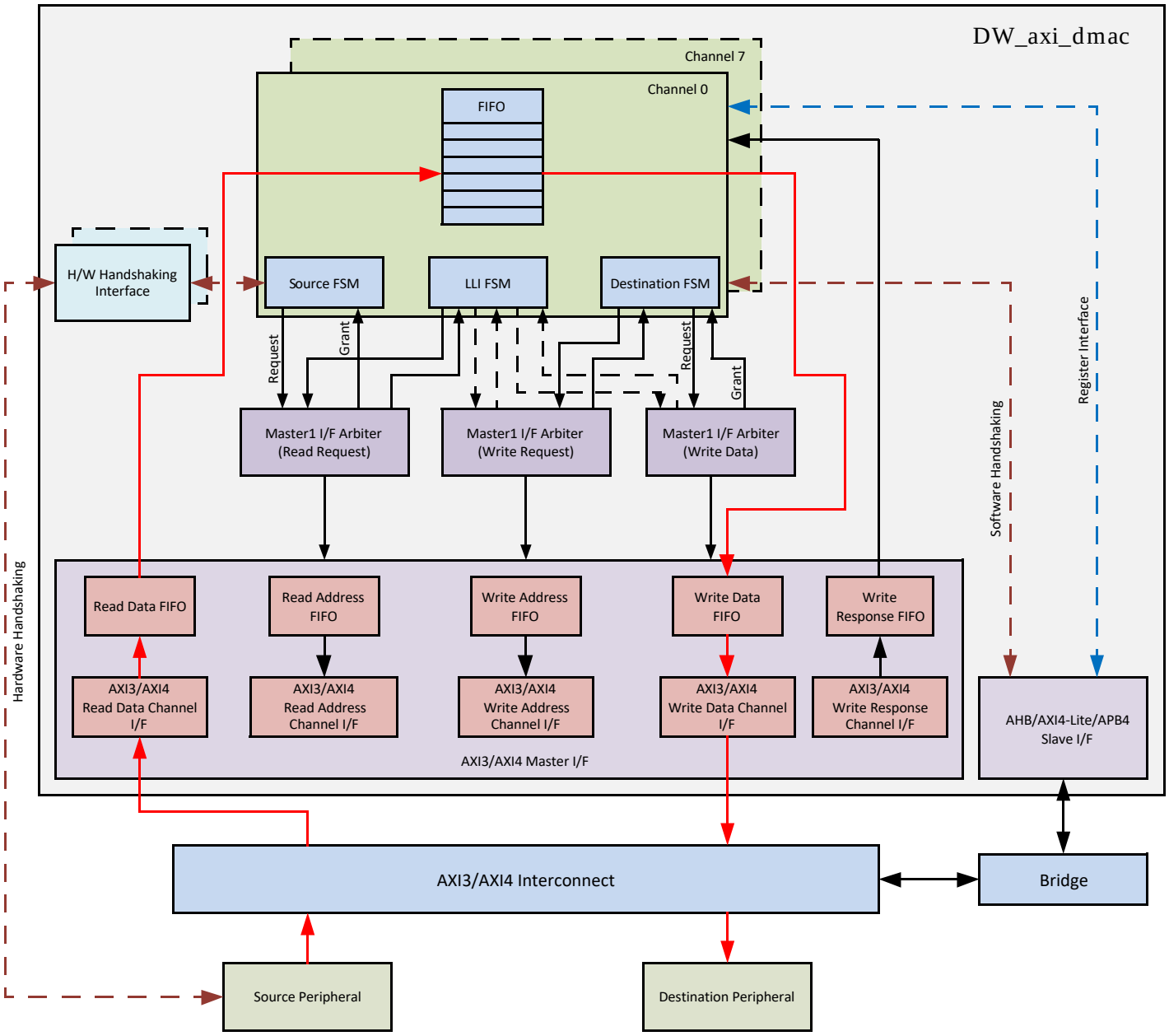
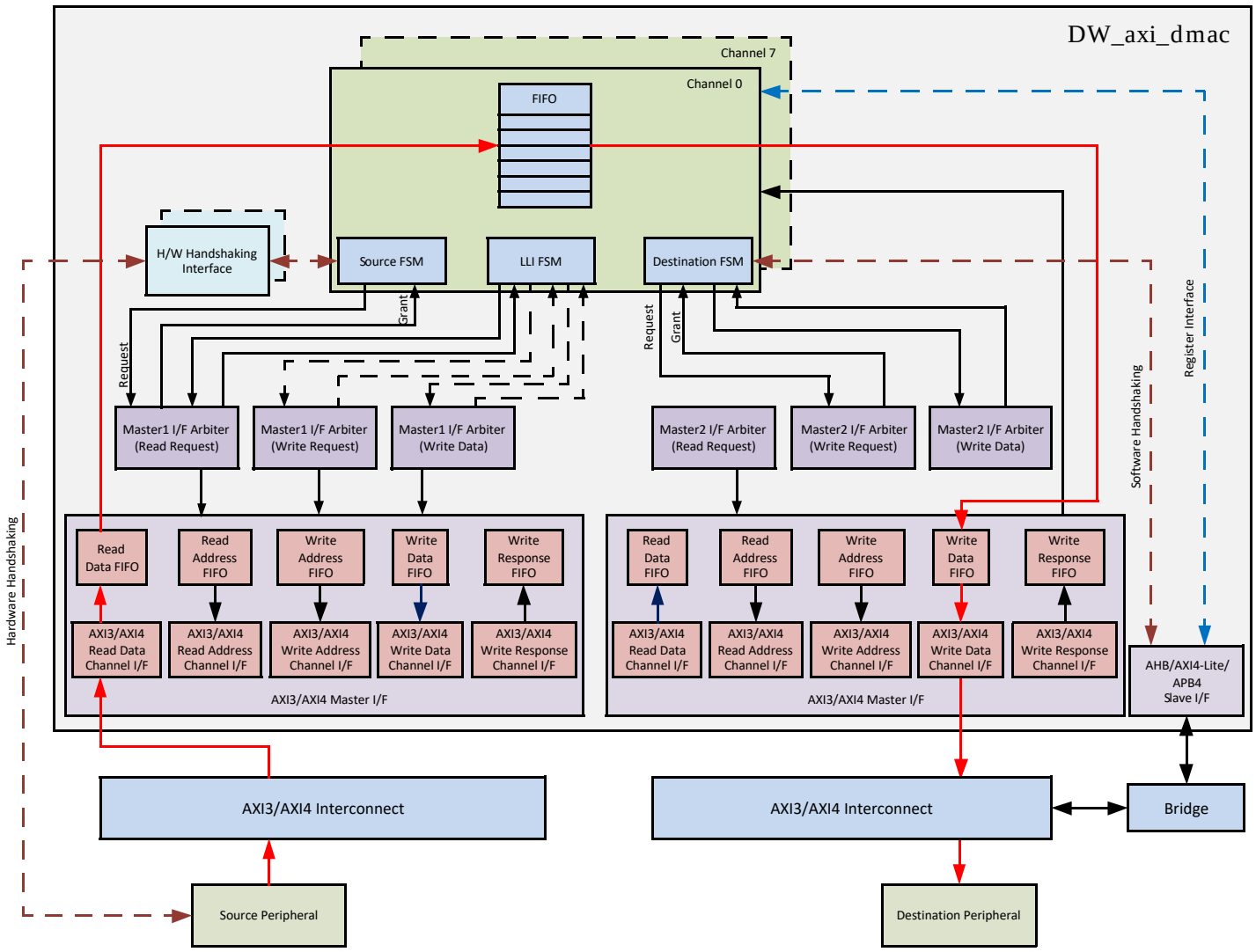


Figure 2-2 DW_axi_dmac Flow Diagram when Source and Destination Peripherals on Different AXI Layers



2.2 Clocks and Resets

DW_axi_dmac design supports different modes of clocking based on the value of DMAX_SLVIF_CLOCK_MODE, DMAX_MSTIF1_CLOCK_MODE, and DMAX_MSTIF2_CLOCK_MODE coreConsultant parameters. There can be a maximum of four clocks in the design; DW_axi_dmac internally takes care of the clock domain crossing requirements.

There can be one to four reset inputs to DW_axi_dmac:

- Slave interface – one reset
- Master 1 and 2 interfaces – one reset each
- One core reset input based on the values of the following coreConsultant parameters:
  - DMAX_SLVIF_CLOCK_MODE,
  - DMAX_MSTIF1_CLOCK_MODE, AND
  - DMAX_MSTIF2_CLOCK_MODE

It is assumed that all reset inputs are asserted and de-asserted at the same time with de-assertion synchronous to the corresponding clocks. De-assertion may not be exactly at the same time as there may be delays associated with respect to the synchronization to each clock domain. All resets can be derived from a single reset input.

Note DW_axi_dmac does not support asserting only of a few resets (not all) or asserting resets at different times (with a delay between each); doing so results in unpredictable behavior.

DW_axi_dmac supports a soft reset, which is set using the DMAC_RST field in the DMAC_ResetReg register. The soft reset procedure is as follows.

1. Software writes 1 to the DMAC_ResetReg.DMAC_RST bit to reset the DW_axi_dmac

   Software polls the DMAC_ResetReg.DMAC_RST bit until it is seen as 0, which confirms the reset.
2. DW_axi_dmac resets all the modules in different clock domains except for the slave bus interface module.

   The Slave bus interface module is not reset because software is polling the DMAC_ResetReg.DMAC_RST field.
3. DW_axi_dmac clears the DMAC_ResetReg.DMAC_RST bit to 0.

Note Software is not allowed to write 0 to the DMAC_ResetReg.DMAC_RST field. A reset does not guarantee the completion of the data transfer for ongoing or posted requests. If a reset is asserted in between transfers, it might result in AXI protocol violations.

2.3 Slave Bus Interface

Slave Bus Interface Module implements the logic to access the internal registers of DW_axi_dmac by an external AHB / AXI4-Lite / APB3 Master. This module supports only the little-endian scheme for data transfer. The Slave Bus Interface module supports a 32- or 64-bit data bus width (the APB3 interface supports only 32 bits).

The Slave Bus Interface can be configured to operate on a different clock than the DW_axi_dmac core clock (dmac_core_clock), which can be configured by setting of the DMAX_SLVIF_CLOCK_MODE parameter in coreConsultant. If DMAX_SLVIF_CLOCK_MODE is set to 1, the Slave Bus Interface module operates on either hclk, aclk, or pclk depending on the protocol used for this interface. The DW_axi_dmac design takes care of the clock domain crossing in this situation.

For more details about configuration and programming of this module, see Chapter 3, “Parameter Descriptions”, Chapter 4, “Signal Descriptions”, and Chapter 5, “Register Descriptions”.

Note The current version of DW_axi_dmac supports only the AHB protocol for the slave bus interface; AXI4-Lite and APB3 support is deferred to future versions.

2.4 Master Interface

The master interface implements the transfer of data on the AXI bus. The AXI protocol can be either AXI3 or AXI4, which is configurable in coreConsultant. DW_axi_dmac supports a maximum of two master interfaces, which operate independently to transfer data between peripherals and memories. Both master interfaces must use the same AXI protocol. Address and Data bus widths of master interfaces are configurable in coreConsultant, but both master interfaces use the same configuration.

The master interface implements all five channels specified in the AXI protocol:

- Write address
- Write data
- Write response
- Read address
- Read data

The master interface implements different FIFOs for temporary storage of data transferred between external memories or peripherals through different channels in DW_axi_dmac core.

The Master interface can be configured to operate on a different clock than the DW_axi_dmac core clock (dmac_core_clock), which can be configured by setting DMAX_MSTIFN_CLOCK_MODE parameter in coreConsultant. If DMAX_MSTIFN_CLOCK_MODE is set to 1, the Master Interface module operates on aclk_m*N*. The DW_axi_dmac design takes care of the clock domain crossing in this case.

The master interface implements logic to convert between big-endian and little-endian format. The endian scheme used for the big-endian format is the Byte Invariant (BE-8) method, which is the big-endian format supported by the AXI protocol. DW_axi_dmac enables you to use the little-endian scheme for LLI fetch and LLI write-back, irrespective of the endian format used for data transfer on that particular master interface.

For more information about configuring or programming the Master interface, see Chapter 3, “Parameter Descriptions”, Chapter 4, “Signal Descriptions”, and Chapter 5, “Register Descriptions”.

DW_axi_dmac supports a maximum of eight channels, which can be assigned to either of the master interfaces, based on a well-defined, programmable arbitration scheme for accessing the master interface. Apart from channel source and destination peripherals, LLI fetch and LLI write-back operations also need access to the master interface. DW_axi_dmac implements a programmable arbitration scheme which is explained in the following section.

2.5 Arbitration Scheme

DMA transfers can be split into the following transfer hierarchy levels:

- Transaction level (only applicable for non-memory peripherals)
- Block level
- Complete DMA transfer level
- AXI transfer level

DW_axi_dmac implements a dynamic priority and fair-among-equals arbitration scheme, with options for channel locking at different DMA transfer hierarchy levels.

The number of priority levels available is the same as the number of enabled channels and the priority value is programmable for each channel in the channel configuration register, CHx_CFG. Multiple channels can be given the same priority level, however, some priority levels remain unused. A priority of 7 is the highest priority, and 0 is the lowest.

Dynamic priority and fair-among-equals arbitration is a two-tier arbitration scheme which works as follows:

- The first-tier arbitration uses the priority inputs of the requests and decides which one of the requesting clients is issued the grant signal. The request with the highest priority is granted access to the AXI channel on the master interface.
- If two or more requests to the arbiter have the same programmed priority value, the second-tier arbitration, based on the fair-among-equals scheme, is used. In this scheme, the grant is issued fairly among actively requesting clients with the same priority level.

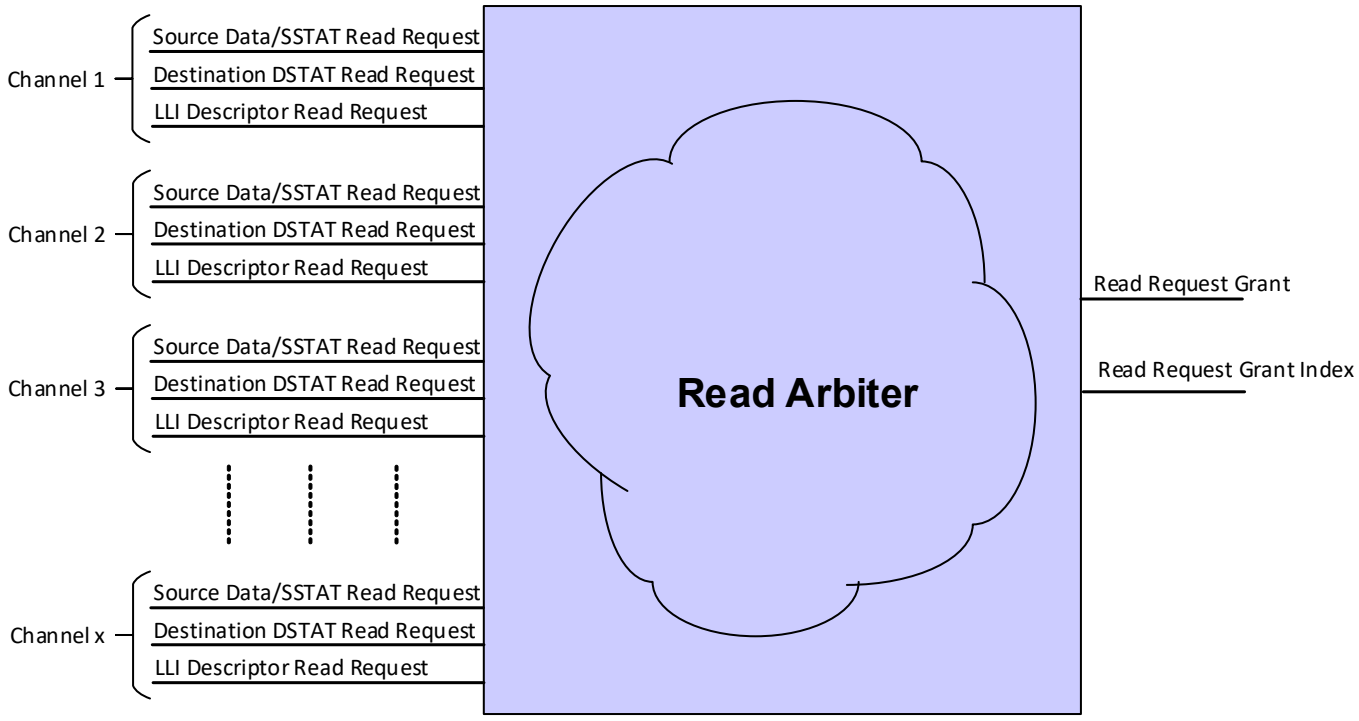
2.5.1 Single Arbiter Scheme

In Single Arbiter scheme, the arbitration of the read requests from source, destination, and LLI state machine is performed by using the Single Arbiter. The read requests from a channel to the arbiter consists of the following requests:

- Source Data and SSTAT Read Request
- Destination DSTAT Read Request
- LLI Descriptor Read Request (Optional, available only when (DMAX_HAS_LLI_PARAM=1))

The Single arbiter scheme performs the arbitration between these read requests from all channels based on the Dynamic priority (first-tier arbitration) and fair-among-equals (second-tier arbitration) scheme described in “Arbitration Scheme” on page 30. The granted read request gets the access to the AXI Master Interface. The block diagram of the Single Arbiter Scheme - Read Arbiter is as shown in Figure 2-3.

Figure 2-3 Single Arbiter Scheme - Read Arbiter



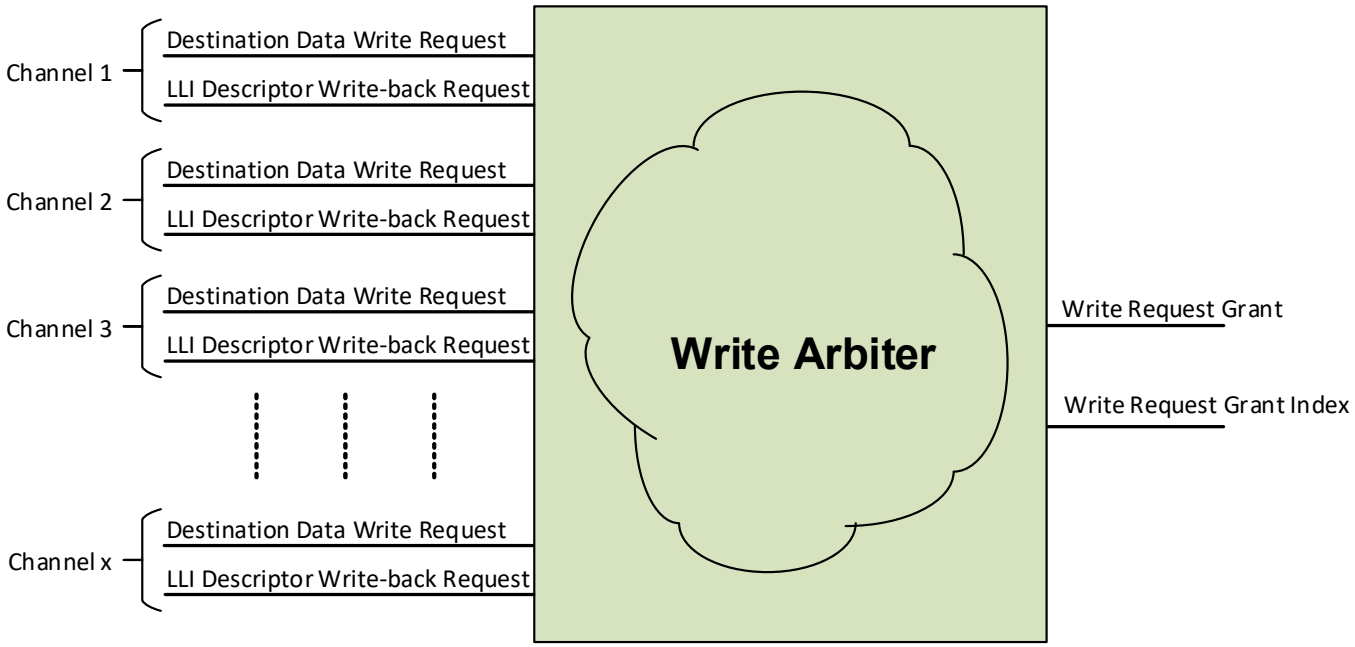
Where, x = DMAX_NUM_CHANNELS that is configured number of Channels.

Similarly, the arbitration of write requests from destination and LLI state machine is performed using the Single Arbiter. The write requests from a channel to arbiter consists of the following:

- Destination Data Write Request
- LLI Descriptor Write-Back Request (Optional, available only when (DMAX_HAS_LLI_PARAM=1) and (DMAX_CHx_LLI_WB_EN=1))

The Single arbiter scheme performs the arbitration between these write requests from all channels based on the Dynamic priority (first-tier arbitration) and fair-among-equals (second-tier arbitration) scheme. The granted write request gets the access to the AXI Master Interface. The block diagram of the Single Arbiter Scheme - Write Arbiter is as shown in Figure 2-4.

Figure 2-4 Single Arbiter Scheme - Write Arbiter



Where, x = DMAX_NUM_CHANNELS that is configured number of channels.

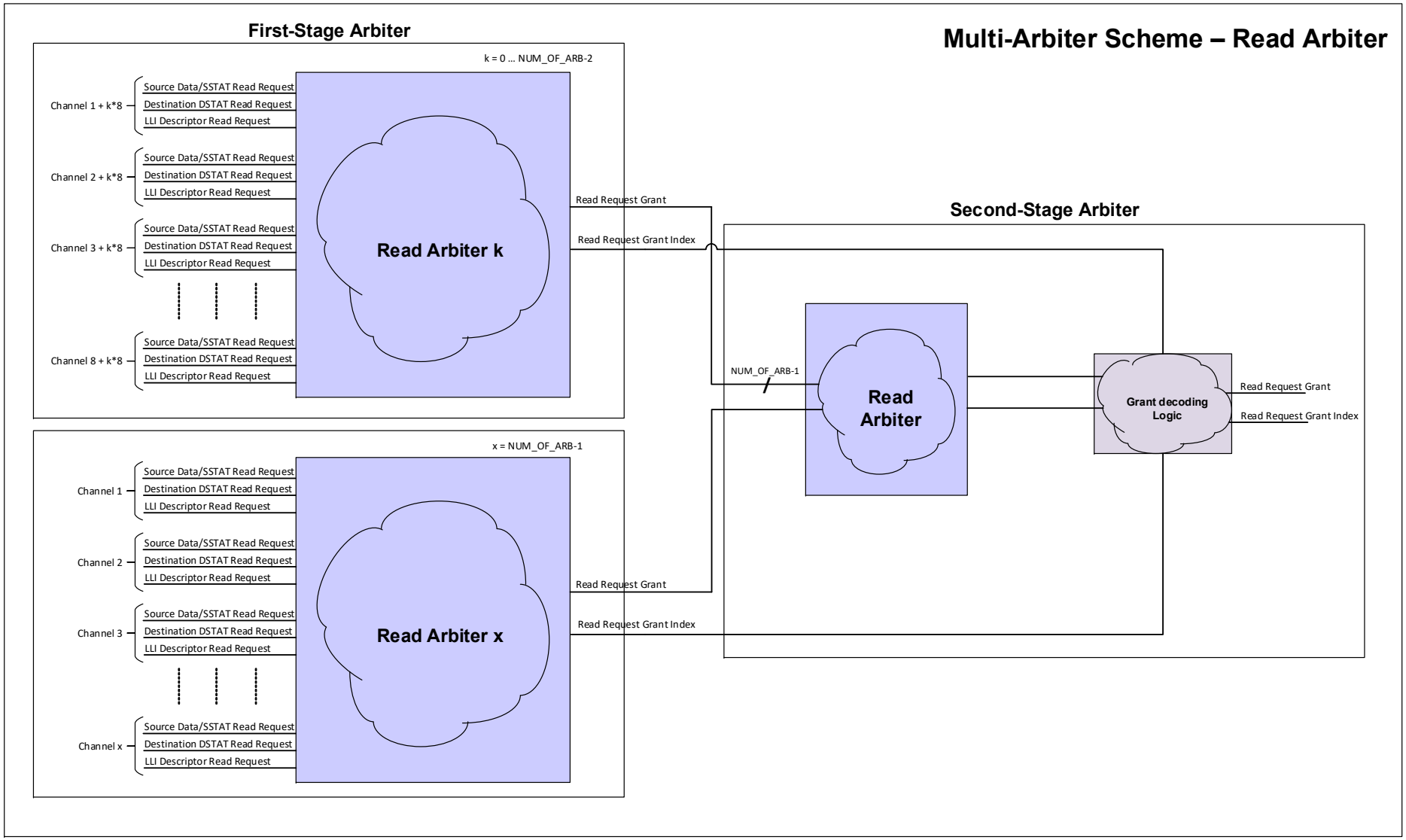
If the number of channels that is DMAX_NUM_CHANNELS are less than or equal to 8, then DW_axi_dmac is configured with Single Arbiter Scheme that is DMAX_MULT_ARB_EN is always hardcoded to 0. If the number channels are greater than 8, then DW_axi_dmac can be configured with Single Arbiter Scheme (DMAX_MULT_ARB_EN = 0) or Multi-Arbiter Scheme (DMAX_MULT_ARB_EN = 1).

2.5.2 Multi-Arbiter Scheme

When the number of channels are greater than 8, to meet higher QoR (timing) requirement (if required) - the Multi-Arbiter Scheme is introduced, which is a configurable feature through the DMAX_MULT_ARB_EN parameter. This Multi-Arbiter scheme uses the two-stage arbitration architecture. The first-stage has multiple arbiters with each having up to 8 Channels related to read or write requests with registered grant outputs. For the definition of read or write requests for a channel, see “Single Arbiter Scheme” on page 30. The second-stage has single arbiter with grant outputs from each of the first-stage arbiter as request input to the second-stage arbiter. The grant output of the second-stage arbiter is not registered. The Multi-Arbiter scheme splits the single arbiter into multiple smaller arbiters as shown in Figure 2-5 and Figure 2-6. This helps in improving the overall QoR of the DW_axi_dmac.

The block diagram of Multi-Arbiter Scheme - Read Arbiter is as shown in Figure 2-5. Each of the Read Arbiter arbitrates using Dynamic priority (first-tier arbitration) and fair-among-equals (second-tier arbitration) scheme at the individual arbiter level.

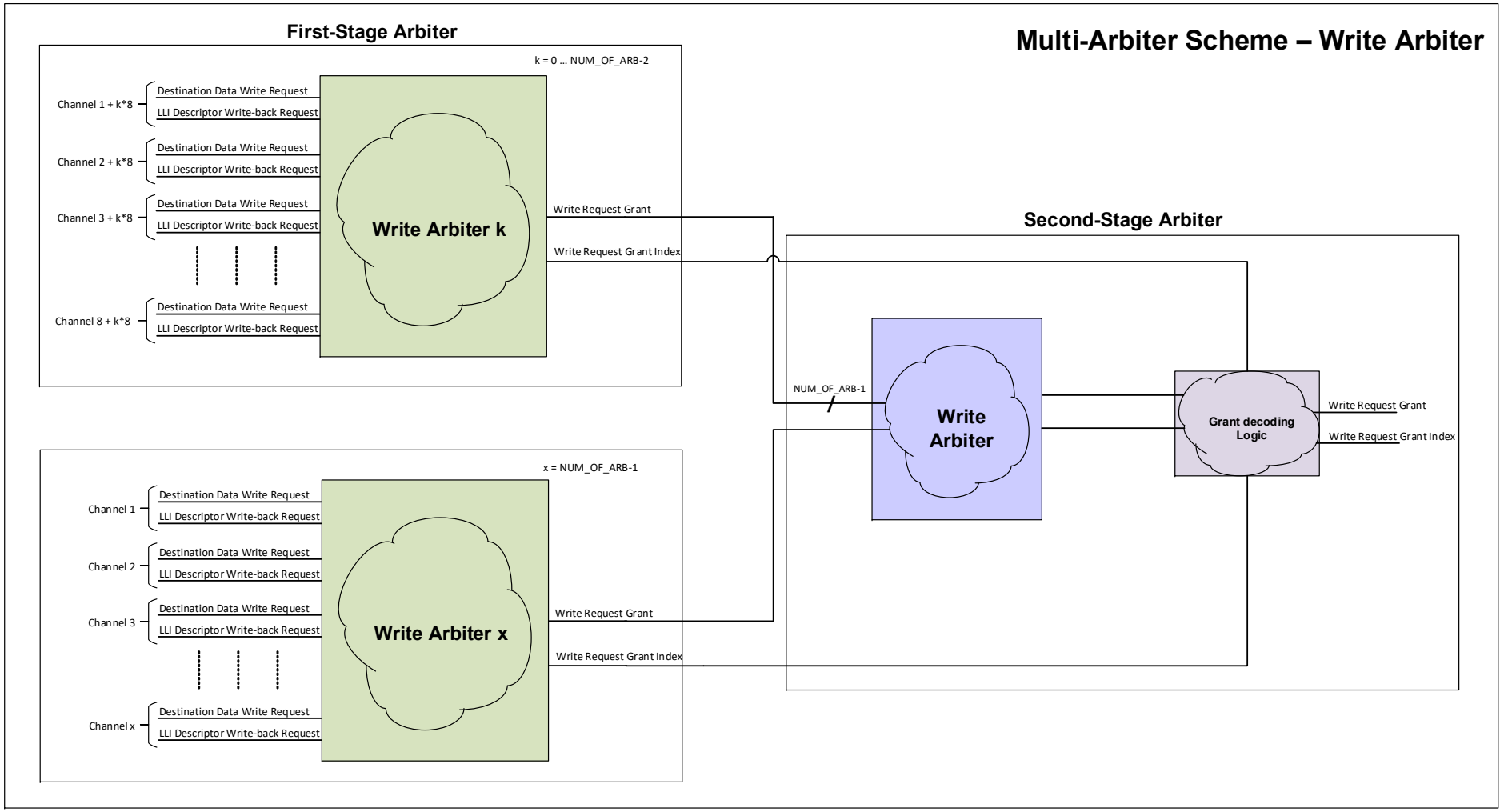
Figure 2-5 Multi-Arbiter Scheme - Read Arbiter



Where, NUM_OF_ARB = DMAX_NUM_CHANNELS/8 + DMAX_NUM_CHANNELS%8

The block diagram of Multi-Arbiter Scheme - Write Arbiter is as shown in Figure 2-6. Each of the Write Arbiter arbitrates using Dynamic priority (first-tier arbitration) and fair-among-equals (second-tier arbitration) scheme at the individual arbiter level.

Figure 2-6 Multi-Arbiter Scheme - Write Arbiter



Where, NUM_OF_ARB = DMAX_NUM_CHANNELS/8 + DMAX_NUM_CHANNELS%8

The following table shows few example of Multi-Arbiter scheme organization with different number of channels configured.

Table 2-1 Multi-Arbiter Scheme - Arbiter Organization

DMAX_NUM_CHANNELS	Number of First-Stage Arbiters (NUM_OF_ARB)	Number of read/write requests per First-Stage Arbiter	Number of requests for Second-Stage Arbiter
32	4	All 4 Arbiters receives request from 8 Channels	4
27	4	First 3 Arbiters receives request from 8 Channels Last Arbiter receives request from 3 Channels	4
21	3	First 2 Arbiters receives request from 8 Channels Last Arbiter receives request from 5 Channels	3
12	2	First Arbiter receives request from 8 Channels Second or last Arbiter receives request from 4 Channels	2

2.5.3 Locks and Grants

The lock input of the arbiter enables a request, despite other requests, to be given an exclusive grant for the duration of the corresponding lock input. After a client receives the grant, it can lock out other clients from the arbitration process by setting the corresponding lock input.

DW_axi_dmac design allows a channel to be locked to the arbiter at the transaction (only applicable for non-memory peripherals), block, or complete DMA transfer level. By default, DW_axi_dmac locks the arbiter for one complete AXI transfer. If channel locking is enabled at a higher level—for a transaction, block, or complete DMA transfer—the channel that locked the arbiter gets access to the master interface until the end of locking period, and requests from other channels during this time are ignored.

2.5.4 Priority for Different Accesses

If the source, destination, and linked list peripherals of different channels with the same priority value are connected on the same master interface, the priority for different accesses are as follows:

- For AXI Read Channel

  Linked List fetch > Source Data Transfer / Source Status fetch / Destination Status fetch

- For AXI Write Channel

  Destination Data Transfer > Linked List Write-back (CHx_LLP_STATUS, Source and Destination Status)

2.5.5 Same Channel Transfer, Fetch, and Access

Source data transfer, destination data transfer, linked list fetch, and linked list status write back access for a particular channel generally happens sequentially as follows.

Linked list fetch > Source data transfer and destination data transfer > Source status fetch and destination status fetch > linked list write back

There is no need for arbitration between different accesses in this case.

2.6 Channel Locking

DMA transfers can be split into the following transfer hierarchy levels:

- Complete transfer level
- Block level
- Transaction level (only applicable for non-memory peripherals)
- AXI transfer level

DW_axi_dmac allows a channel to be locked to the arbiter during memory-to-memory transfers at block and complete DMA transfer levels. If channel locking is enabled on a block or at the complete DMA transfer level, the channel that locked the arbiter gets access to the master interface until the end of the locking period, and requests from other channels during this time are ignored.

2.6.1 Enabling Channel Locking

Channel locking can be implemented in the following scenarios.

- If channel locking is enabled at a particular transfer hierarchy and it is a memory-to-memory transfer, the locking of the corresponding channel to the AXI channels of the master interface is established.
- Transfer is a memory-to-memory transfer.

Hardware does not check for the validity of the channel locking setting, therefore, the software must enable the channel locking only for memory-to-memory transfers at block or DMA transfer levels. Illegal programming of channel locking might result in unpredictable behavior.

2.6.2 Clearing Channel Locking

Channel locking is cleared when DW_axi_dmac suspends, disables, or aborts the channel upon request from software, or DW_axi_dmac disables the channel upon receiving an error response on the master interface.

The ChLock_Cleared_IntStat bit in the CHx_IntStatusReg register is set to 1 if DW_axi_dmac clears the channel locking due to an error response received on the master interface, or if software suspends, disables, or aborts the channel. Additionally, the ChLock_Cleared_IntStat is cleared at the end of each block or DMA transfer if channel locking is enabled on DMA block level or transfer level, respectively.

2.6.3 Possible Deadlock Conditions

A deadlock condition can occur when multiple channels are enabled concurrently on a master interface and one of these channels has obtained a read lock on the master interface and the remaining channels are locked out from proceeding with a DMA transfer. While programming, you must ensure that sure deadlock do not occur. A deadlock can occur only for configurations where the number of masters are equal to 2 (DMAC_NUM_MASTER_IF = 2) and the number of channels are more than 1 (DMAC_NUM_CHANNELS > 1).

Figure 2-7 illustrates a scenario in which the read channel is locked out for subsequent read requests.

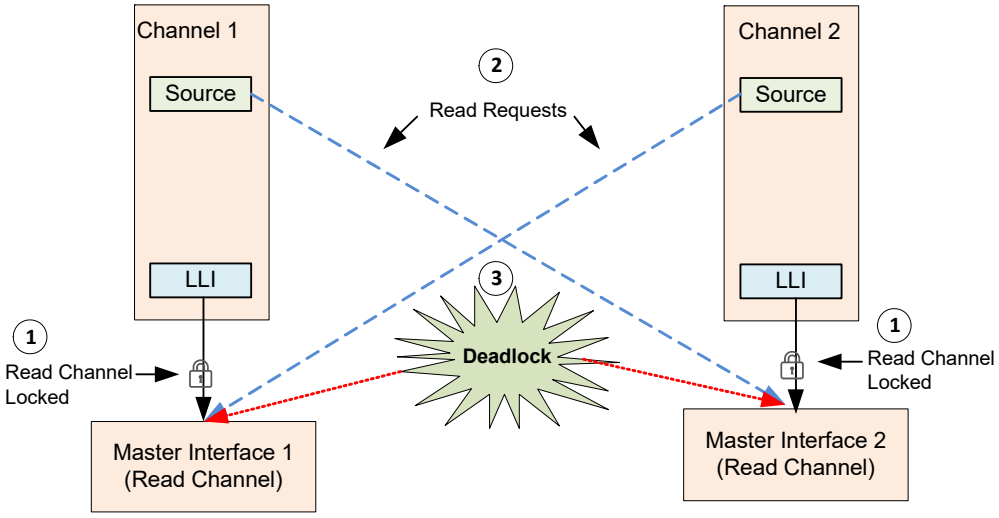
Figure 2-7 Scenario 1: Deadlock Condition

Channel 1 Programming Instructions

CH1_CTL.SMS= 1	Source on Master Interface 2
CH1_CTL.DMS=0	Destination on Master Interface 1
CH1_LLP.LMS=0	LLI on Master interface 1
CH1_CFG.LOCK_CH_L=01	Locking over complete DMA transfer
CH1_CFG.LOCK_CH=1	Locking enabled

Channel 2 Programming Instructions

CH2_CTL.SMS= 0	Source on Master Interface 1
CH2_CTL.DMS=0	Destination on Master interface 1
CH2_LLP.LMS=1	LLI on Master interface 2
CH2_CFG.LOCK_CH_L=01	Locking over complete DMA transfer
CH2_CFG.LOCK_CH=1	Locking enabled



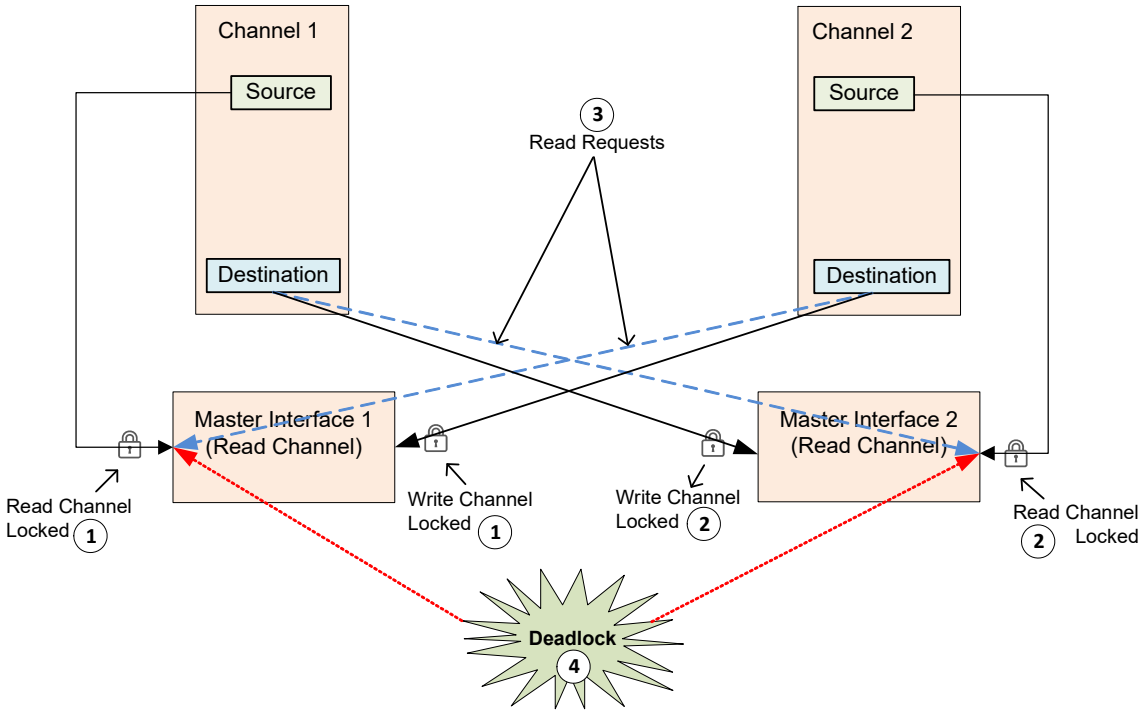
1. LL1 in both Channels 1 and 2 request lock for Master interface 1 and 2 read address channels, respectivley.
   Request granted for complete DMA transfer as no other channel is active.
2. After sometime, if:
   - Source of Channel 1 requests for Master 2 read channel
   - Source of Channel 2 requests for Master 1 read channel
3. Deadlock occurs as the master read address channels are locked with different channels.

Figure 2-8 illustrates another scenario in which the read channel is locked out for subsequent read requests.

Figure 2-8 Scenario 2: Deadlock Condition

Channel 1 Programming Instructions	
CH1_CTL.SMS= 0	Source on Master interface 1
CH1_CTL.DMS=1	Destination on Master interface 2
CH1_CTL.DST_STAT_EN=1	Destination on Status Fetch is enabled
CH1_LLP.LMS=0	LLI on Master interface 1
CH1_CFG.LOCK_CH_L=01	Locking over complete DMA transfer
CH1_CFG.LOCK_CH=1	Locking enabled

Channel 2 Programming Instructions	
CH2_CTL.SMS= 1	Source on Master interface 2
CH2_CTL.DMS=0	Destination on Master interface 1
CH2_CTL.DST_STAT_EN=1	Destination on Status Fetch is enabled
CH2_LLP.LMS=1	LLI on Master interface 2
CH2_CFG.LOCK_CH_L=01	Locking over complete DMA transfer
CH2_CFG.LOCK_CH=1	Locking enabled



1. Master interface 1 read and write address channels locked by Channel 1 and Channel 2, respectivley.
2. Master interface 2 read and write address channels locked by Channel 1 and Channel 2 respectively.
3. At the end of first block, if:
   - Destination of Channel 1 requests Master 2 read channel for status fetch
   - Destination of Channel 2 requests Master 1 read channel for status fetch
4. Deadlock occurs as both master read address channels are locked by different channels.

2.7 Endian Scheme

DW_axi_dmac supports little-endian and big-endian format for data access on the AXI master interface. The endian scheme used for the big-endian format is the Byte Invariant (BE-8) method, which is the format supported by the AXI protocol. If the AXI master interface is configured for the big-endian format, big-endian BE-8-to-little-endian conversion is done for AXI read data and little-endian-to-big-endian BE-8 conversion is done for AXI write data.

The following figures show the conversion process between big-endian BE-8 and little-endian data appearing on the AXI master bus for different data bus width and transfer size combinations.

Figure 2-9 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 32-Bit Data Bus (Byte Access)

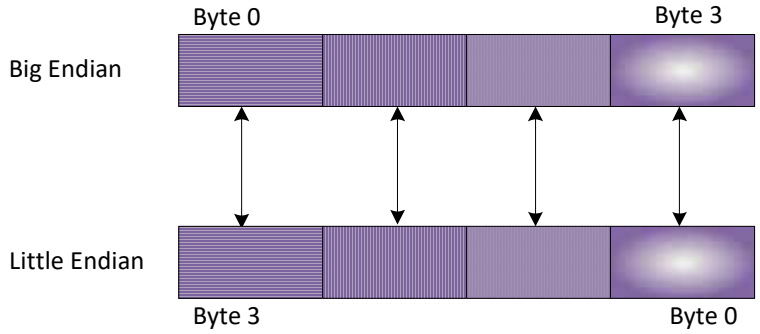


Figure 2-10 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 32-Bit Data Bus (Half Word Access)

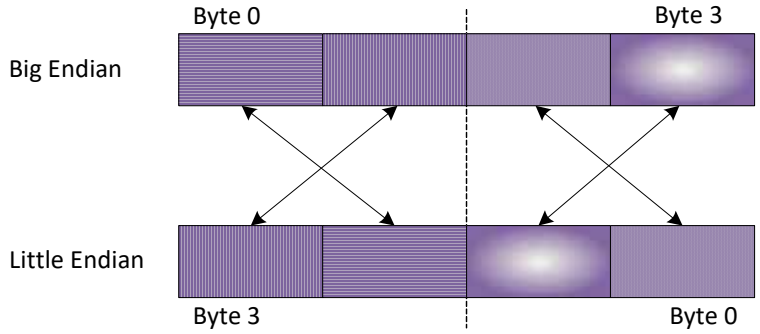


Figure 2-11 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 32-Bit Data Bus (Word Access)

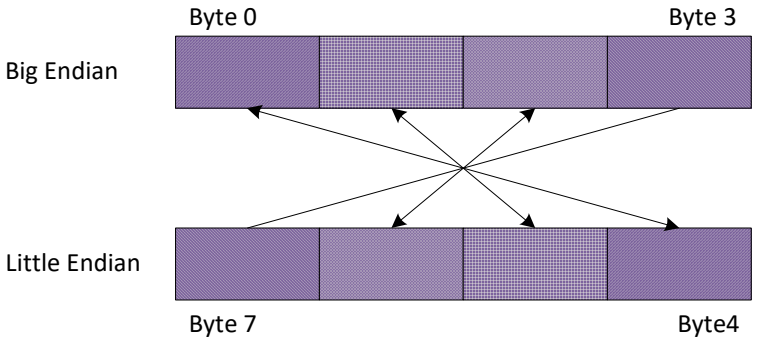


Figure 2-12 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Byte Access)

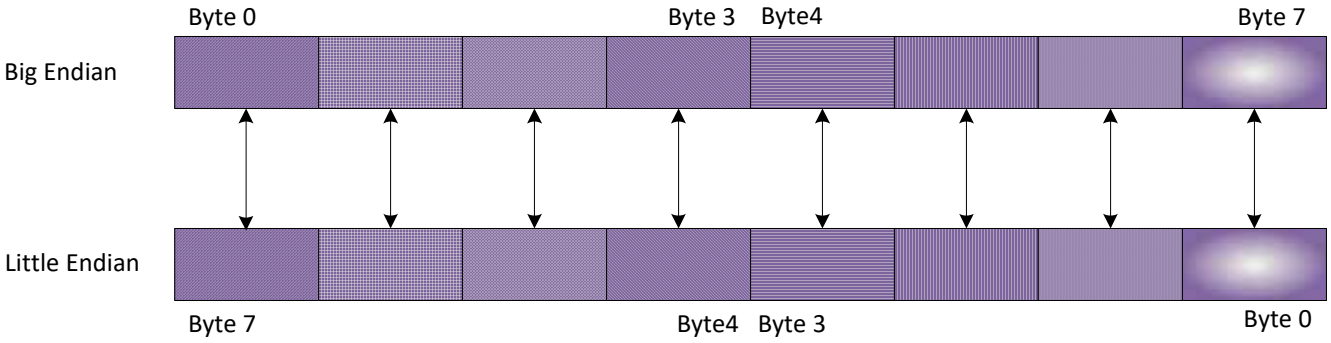


Figure 2-13 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Half Word Access)

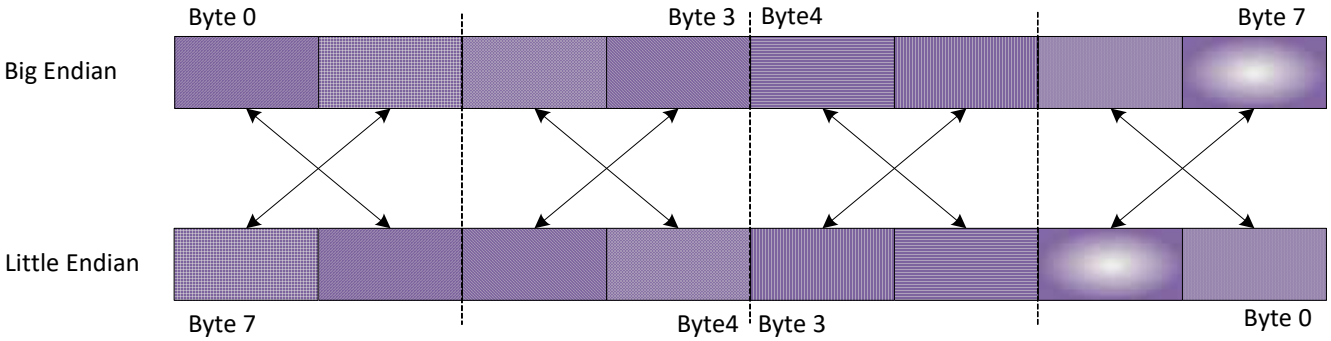


Figure 2-14 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Word Access)

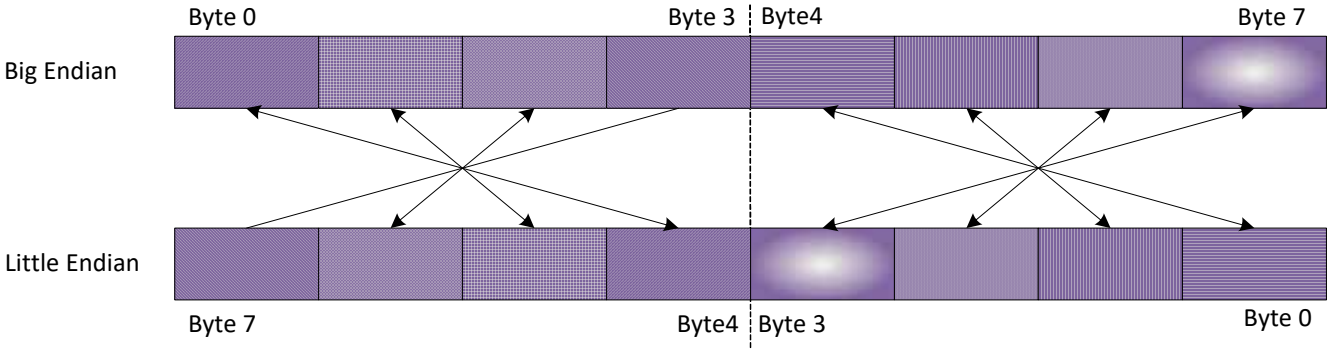


Figure 2-15 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Double Word Access)

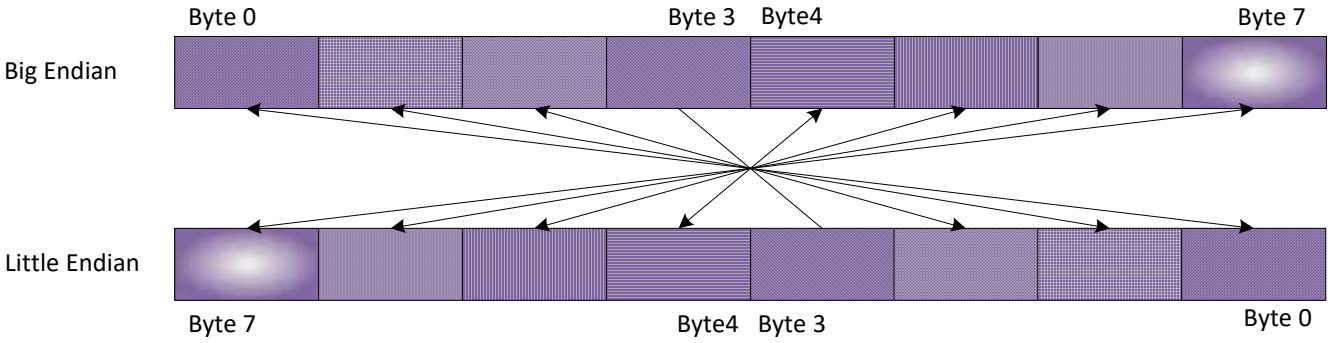


Figure 2-16 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Byte Access)

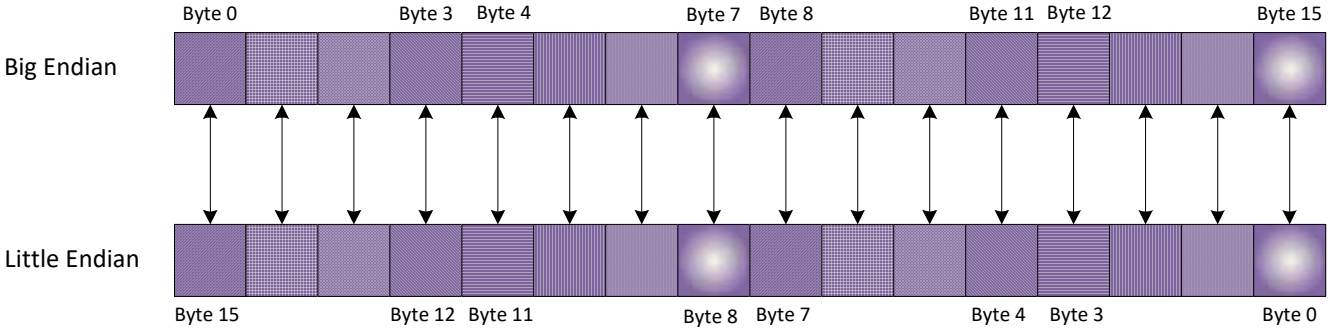


Figure 2-17 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Half Word Access)

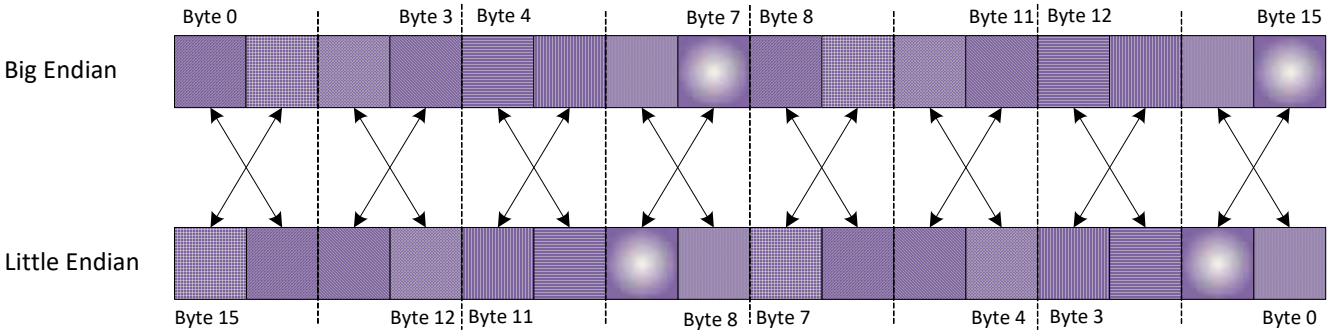


Figure 2-18 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Word Access)

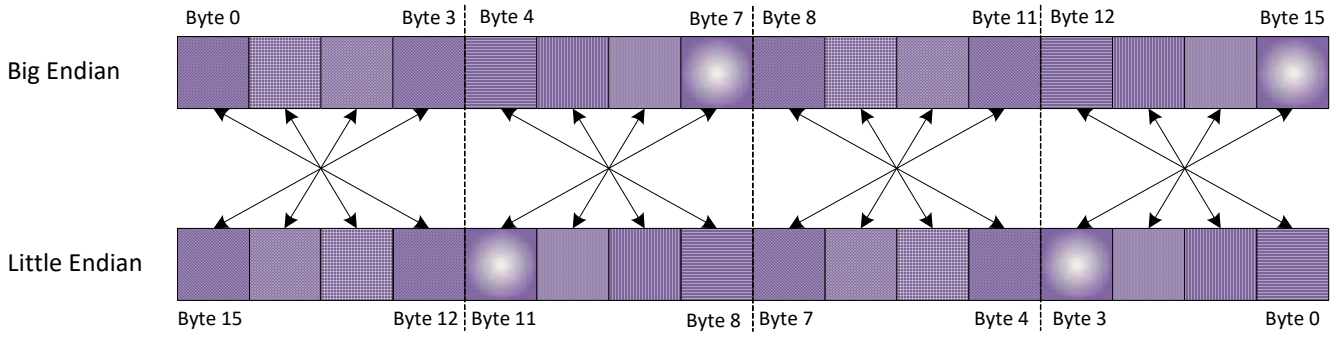


Figure 2-19 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Double Word Access)

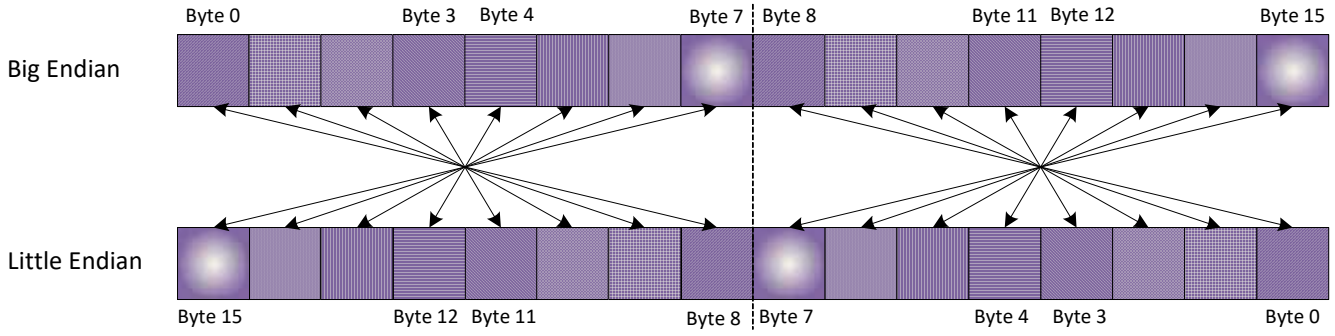
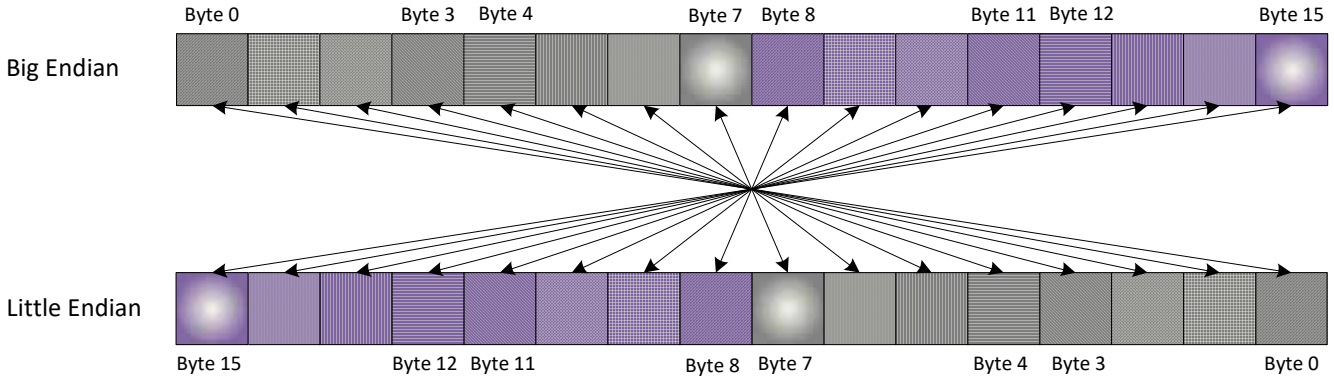


Figure 2-20 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (128-Bit Access)



Note A similar scheme is used for other transfer sizes as well. In general, the data bus is grouped in terms of transfer size and bytes within each group are swapped such that the lower byte becomes the higher byte, and the higher byte becomes the lower byte.

The endian scheme used for the source and destination data transfer, LLI fetch, and status and control write-back access on the master interfaces is decided by the value of the respective coreConsultant parameters and the I/O signals. Table 2-2 shows all possible combinations of the endian scheme on the master interface.

Table 2-2 Endian Formats Supported on Master Interface

DMAX_STATIC_ENDIAN_SELECT_MSTIF	DMAX_ENDIAN_FORMAT_MSTIF	dmac_endian_format_mstif1/2	Endian Scheme Used for SRC/DST Data Access on M1/M2 Master Interface	DMAX_LLI_ENDIAN_SELECTION_PIN_EN	dmac_le_select_lli_mstif1/2	Endian Scheme Used for LLI Fetch/ Status and Control Write Back Access on M1/M2 Master Interface
1	0	X	Little-Endian	X	X	Little-Endian
1	1	X	Big-Endian BE-8	0	X	Big-Endian BE-8
				1	0	Big-Endian BE-8
					1	Little-Endian
0	X	0	Little-Endian	X	X	Little-Endian
0	X	1	Big-Endian BE-8	0	X	Big-Endian BE-8
				1	0	Big-Endian BE-8
					1	Little-Endian

2.8 Interrupt Interface

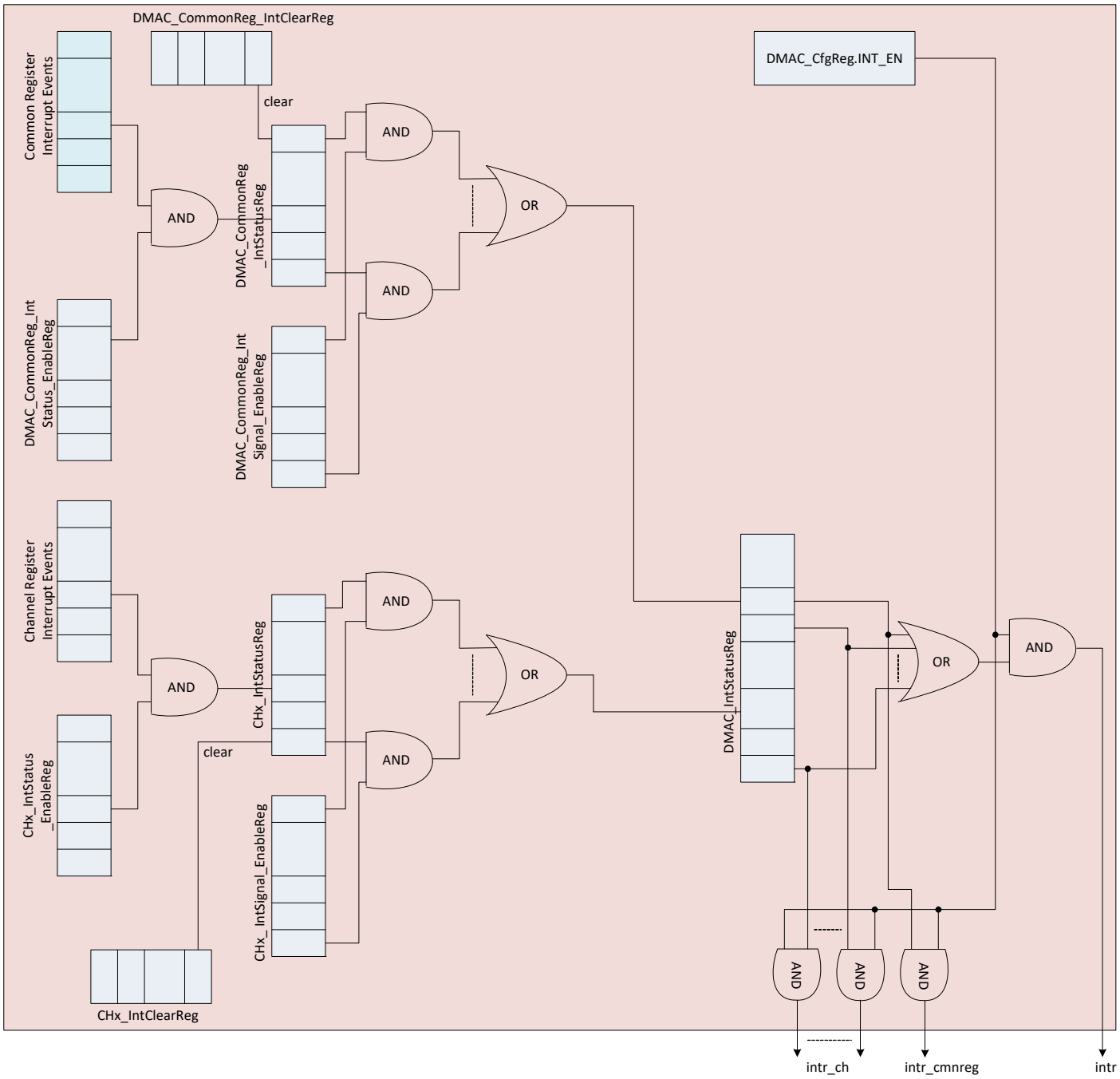
DW_axi_dmac supports combined and individual interrupt outputs configurable in coreConsultant. The polarity of the interrupt (Active High / Active Low) can be configured in coreConsultant. Interrupt outputs, by default, are synchronous to the core clock (dmac_core_clock), but they can be synchronized to the slave interface clock using the DMAX_INTR_SYNC2SLVCLK parameter in coreConsultant. If this parameter is set to 1, interrupt outputs are synchronous to hclk / aclk / pclk based on the protocol used for the slave interface. DW_axi_dmac manages the clock domain synchronization requirements in this configuration.

DW_axi_dmac has different registers to support the interrupt interface. Software can read these registers to understand the source of the interrupt and take appropriate actions.

Note DW_axi_dmac supports synchronizing the interrupt outputs to the Slave Interface clock domain. The synchronization mechanism is not captured in Figure 2-21.

Figure 2-21 shows the interrupt generation mechanism in DW_axi_dmac.

Figure 2-21 DW_axi_dmac Interrupt Generation



2.9 Single Transaction Region

In certain cases, a DMA block transfer cannot complete using only burst transactions. Typically this occurs when the block size is not a multiple of the burst transaction length. In these cases, the block transfer uses burst transactions up to the point where the amount of data left to complete the block is less than the amount of data in a burst transaction. At this point, the DW_axi_dmac samples the dma_single status flag and completes the block transfer using single transactions. The peripheral asserts a single status flag to indicate to the DW_axi_dmac that there is enough data or space to complete a single transaction from or to the source or destination peripheral.

For hardware handshaking, the single status flag is a signal on the hardware handshaking interface. For software handshaking, the single status flag is one bit in the software handshaking interface register.

The single transaction region is the time interval where the DW_axi_dmac uses single transactions to complete the block transfer; burst transactions are used only outside this region.

The following terms are used for explaining single transaction region.

- Source single transaction size in bytes:

  src_single_size_bytes = CHx_CTL.SRC_TR_WIDTH / 8

- Source burst transaction size in bytes:

  src_burst_size_bytes = CHx_CTL.SRC_MSIZE * src_single_size_bytes

- Destination single transaction size in bytes:

  dst_single_size_bytes = CHx_CTL.DST_TR_WIDTH / 8

- Destination burst transaction size in bytes:

  dst_burst_size_bytes = CHx_CTL.DST_MSIZE * dst_single_size_bytes

- Block size in bytes:

  - If DW_axi_dmac is flow controller – With the DW_axi_dmac as the flow controller, the processor programs the DW_axi_dmac with the number of data items (block size) of source transfer width (CHx_CTL.SRC_TR_WIDTH) to be transferred by the DW_axi_dmac in a block transfer; this is programmed into the CHx_BLOCK_TS.BLOCK_TS field. Therefore, the total number of bytes to be transferred in a block is:

    blk_size_bytes_dma = CHx_BLOCK_TS.BLOCK_TS * src_single_size_bytes

  - If source peripheral is flow controller

    blk_size_bytes_src = (Number of source burst transactions in block * src_burst_size_bytes) + (Number of source single transactions in block * src_single_size_bytes)

  - If destination peripheral is block flow controller

    blk_size_bytes_dst = (Number of destination burst transactions in block * dst_burst_size_bytes) + (Number of destination single transactions in block * dst_single_size_bytes)

The single transaction region applies only to a peripheral that is not the flow controller. The precise definition of when this region is entered depends on what acts as the flow controller.

- If the DW_axi_dmac is the flow controller:
  - The source peripheral enters the single transaction region when the number of bytes left to complete in the source block transfer is less than src_burst_size_bytes.

    If blk_size_bytes_dma / src_burst_size_bytes is an integer, then the source peripheral never enters this region, and the source block transfer uses only burst transactions.
  - The destination peripheral enters the single transaction region when the number of bytes left to complete in the destination block transfer is less than dst_burst_size_bytes.

    If blk_size_bytes_dma / dst_burst_size_bytes is an integer, then the destination peripheral never enters this region, and the destination block transfer uses only burst transactions.
- If either the source or the destination peripheral is the flow controller:
  - Single transaction region is not applicable for the flow controller peripheral.
  - If the source peripheral is the flow controller, the destination peripheral enters the single transaction region when the flow control peripheral – that is, the source – signals the last transaction in the block and when the amount of data left to be transferred in the destination block is less than the value specified by dst_burst_size_bytes.
  - If the destination peripheral is the flow controller, the source peripheral enters the single transaction region when the flow control peripheral – that is, the destination – signals the last transaction in the block and when the amount of data left to be transferred in the source block is less than the value specified by src_burst_size_bytes.

If a peripheral is not the flow controller, DW_axi_dmac ignores dma_single input outside the single transaction region.

2.10 Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or burst transactions. The operation of the handshaking interface depends on whether the peripheral or the DW_axi_dmac is the flow controller. The peripheral uses the handshaking interface to indicate to the DW_axi_dmac that it is ready to transfer or accept data over the AXI bus.

DW_axi_dmac supports a maximum of 16 hardware handshaking interfaces, which are configurable in coreConsultant. The type of handshaking interface used (software or hardware) is independently programmable for each channel source and destination in the channel configuration register, CHx_CFG. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

2.10.1 Hardware Handshaking

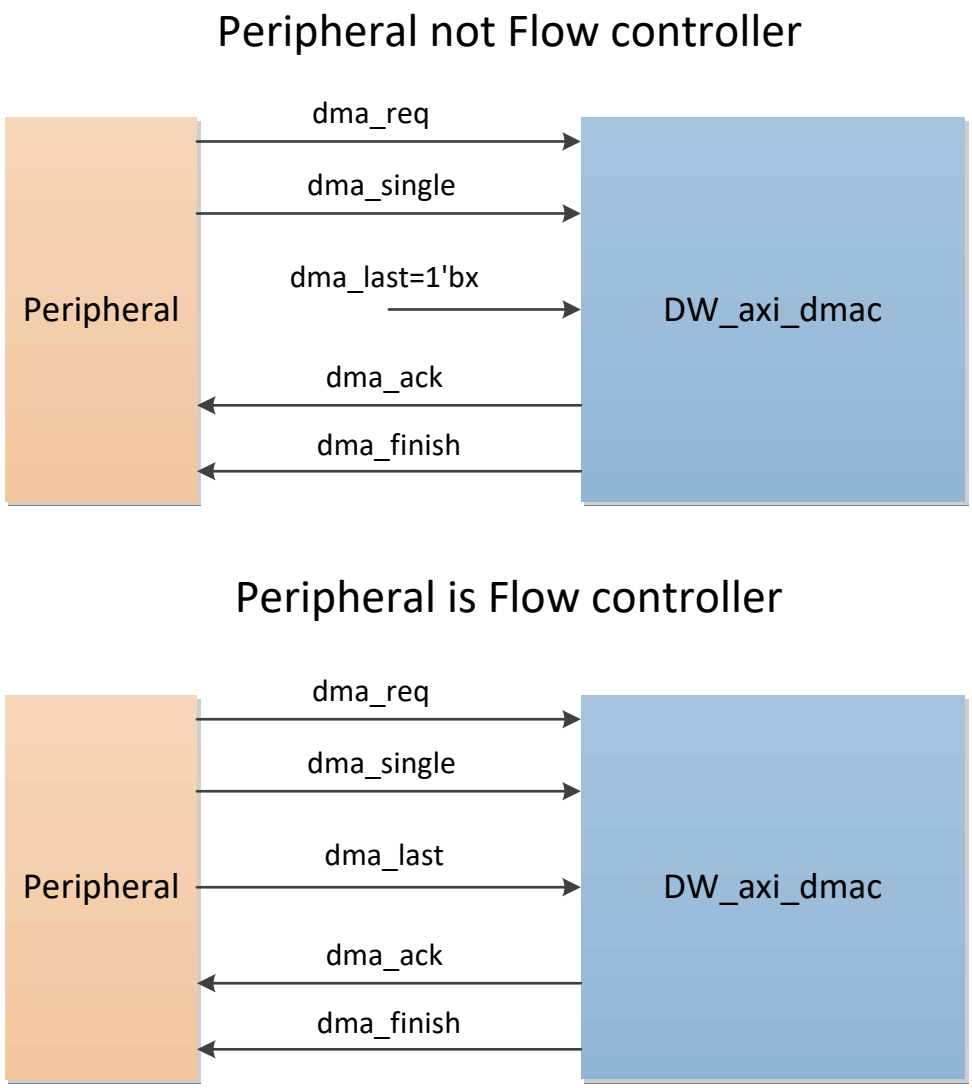
The following set of I/O signals are used for hardware handshaking.

- **dma_req** – Burst transaction request input from peripheral. The functionality of this signal depends on whether the peripheral is the flow controller or not.
  - **If peripheral is not flow controller** – The DW_axi_dmac always interprets the dma_req signal as a burst transaction request, regardless of the level of dma_single. Once dma_req is asserted, it must remain asserted until dma_ack is asserted. When the peripheral that is driving dma_req determines that dma_ack is asserted, it must de-assert dma_req. If an active level on dma_req is detected in the single transaction region, then the block transfer is completed using an early terminated burst transaction.
  - **If peripheral is flow controller** – An active level on dma_req initiates a transaction request. The type of transaction – whether single or burst – is qualified by the dma_single input. Once dma_req is asserted, it must remain asserted until dma_ack is asserted. When the peripheral that is driving dma_req determines that “dma_ack” is asserted, it must de-assert dma_req.
- **dma_single** - Single transaction request input from peripheral. The functionality of this signal depends on whether the peripheral is the flow controller or not.
  - **If peripheral is not flow controller** – The dma_single signal is a status signal that is asserted by a source or destination peripheral when it can transmit or accept at least one source or destination data item; otherwise it is cleared. This signal is sampled by the DW_axi_dmac only in the single transaction region of the block transfer. Outside this region, dma_single is ignored and all transactions are burst transactions, which means DW_axi_dmac waits for dma_req to be asserted. Once dma_single is asserted, it must remain asserted until dma_ack is asserted. When the peripheral that is driving dma_single determines that dma_ack is asserted, it must de-assert dma_single.
  - **If peripheral is flow controller** – The dma_single signal is asserted by a source or destination peripheral to request a single transaction. If dma_single is asserted in the same clock cycle as dma_req is asserted, a single transaction is requested by the peripheral. If dma_single is de-asserted, the peripheral is requesting a burst transaction. Once dma_single is asserted, it must remain asserted until dma_ack is asserted. When the peripheral that is driving dma_single determines that dma_ack is asserted, it must de-assert dma_single.

- **dma_last** - The last transaction in a block request from a peripheral. The functionality of this signal depends on whether the peripheral is the flow controller or not.
  - **If peripheral is not flow controller** – The dma_last signal is not relevant and it is ignored.
  - **If peripheral is flow controller** – The peripheral asserts dma_last on the same cycle as dma_req is asserted in order to signal that this transaction request is the last in the block and the block transfer is complete after this transaction is complete. If dma_single is high in the same cycle as dma_last (and dma_req) is asserted, the last transaction is a single transaction. If dma_single is low in the same cycle as dma_last (and dma_req) is asserted, the last transaction is a burst transaction. Once dma_last is asserted, it must remain asserted until dma_ack is asserted. When the peripheral that is driving dma_last determines that dma_ack is asserted, it must de-assert dma_last.
- **dma_ack** - DW_axi_dmac acknowledges signal output to the peripheral. The functionality of this signal depends on whether the peripheral is the flow controller or not.
  - **If peripheral is not flow controller** – The dma_ack signal is asserted after the last AXI data transfer in the current transaction (single or burst) to the peripheral has completed. For a single transaction, dma_ack remains asserted until the peripheral de-asserts dma_single (that is, dma_ack is de-asserted one dmac_core_clock cycle after the de-assertion of dma_single). For a burst transaction, dma_ack remains asserted until the peripheral de-asserts dma_req (that is, dma_ack is de-asserted one dmac_core_clock cycle after the de-assertion of dma_req).
  - **If peripheral is flow controller** – The dma_ack signal is asserted after the last AXI data transfer in the current transaction (single or burst) to the peripheral has completed. It forms a handshaking loop with dma_req and remains asserted until the peripheral de-asserts dma_req (that is, dma_ack is de-asserted one dmac_core_clock cycle after the de-assertion of dma_req).
- **dma_finish** - This is the DW_axi_dmac block transfer complete indication signal output to the peripheral. The dma_finish signal is asserted to signal block completion. This uses the same timing as dma_ack and forms a handshaking loop with dma_req and/or dma_single.

Figure 2-22 shows the hardware handshaking interface between a destination or source peripheral and DW_axi_dmac.

Figure 2-22 Hardware Handshaking Interface



Note

- Once a peripheral asserts dma_req, dma_single, or dma_last, de-asserting the signal before the DW_axi_dmac asserts dma_ack is not allowed. Doing so might result in unpredictable behavior.
- Irrespective of who is the flow controller and whether the peripheral is in Single Transaction Region, dma_req, dma_single, and dma_last signals must be de-asserted once DW_axi_dmac asserts dma_ack. It is recommended that you follow this requirement, otherwise, unpredictable behavior may occur.

Table 2-3 shows all possible combinations of Flow Controller and Hardware Handshaking Interface signal values.

Table 2-3 Flow Controller and Hardware Handshaking Interface

Peripheral Flow Controller?	Peripheral in Single Transaction Region?	dma_req	dma_single	dma_last	DW_axi_dmac Action
No	No	0	0	X	Not a valid transaction request
		0	1	X	Not a valid transaction request. If peripheral is not Flow Controller, “dma_single” is not sampled by DW_axi_dmac outside Single Transaction Region.
		1	X	X	Burst transaction request. If peripheral is not Flow Controller, “dma_single” is not sampled by DW_axi_dmac outside Single Transaction Region. If peripheral is not the flow controller, “dma_last” does not have any relevance and it is ignored.
	Yes	0	0	X	Not a valid transaction request.
		0	1	X	Single Transaction request. DW_axi_dmac initiates AXI burst of burst length 1. If peripheral is not the flow controller, “dma_last” does not have any relevance and it is ignored.
		1	0	X	Burst Transaction request. DW_axi_dmac initiates AXI burst of burst length needed to complete the transaction (AXI burst length <= transaction size). This is called Early Burst Termination. If peripheral is not the flow controller, “dma_last” does not have any relevance and it is ignored.
		1	1	X	Burst Transaction request. DW_axi_dmac initiates AXI burst of burst length needed to complete the transaction (AXI burst length <= transaction size). This is called Early Burst Termination. If peripheral is not the flow controller, “dma_last” does not have any relevance and it is ignored.

Table 2-3 Flow Controller and Hardware Handshaking Interface (Continued)

Peripheral Flow Controller?	Peripheral in Single Transaction Region?	dma_req	dma_single	dma_last	DW_axi_dmac Action
Yes	Single Transaction Region is not applicable to Flow Control Peripheral	0	0	X	Not a valid transaction request.
		0	1	X	Not a valid transaction request. If peripheral is flow controller, “dma_req” MUST be asserted to in it ate a transaction. DW_axi_dmac waits till “dma_req” is asserted. The type of transaction - single or burst - is qualified by “dma_single” input.
		1	0	0	Burst transaction request (not the last transaction).
		1	0	1	Last Burst transaction request.
		1	1	0	Single transaction request (not the last transaction).
		1	1	1	Last single transaction request.

2.10.1.1 Hardware Handshaking Transaction Examples

This section provides various examples of hardware handshaking scenarios, as follows:

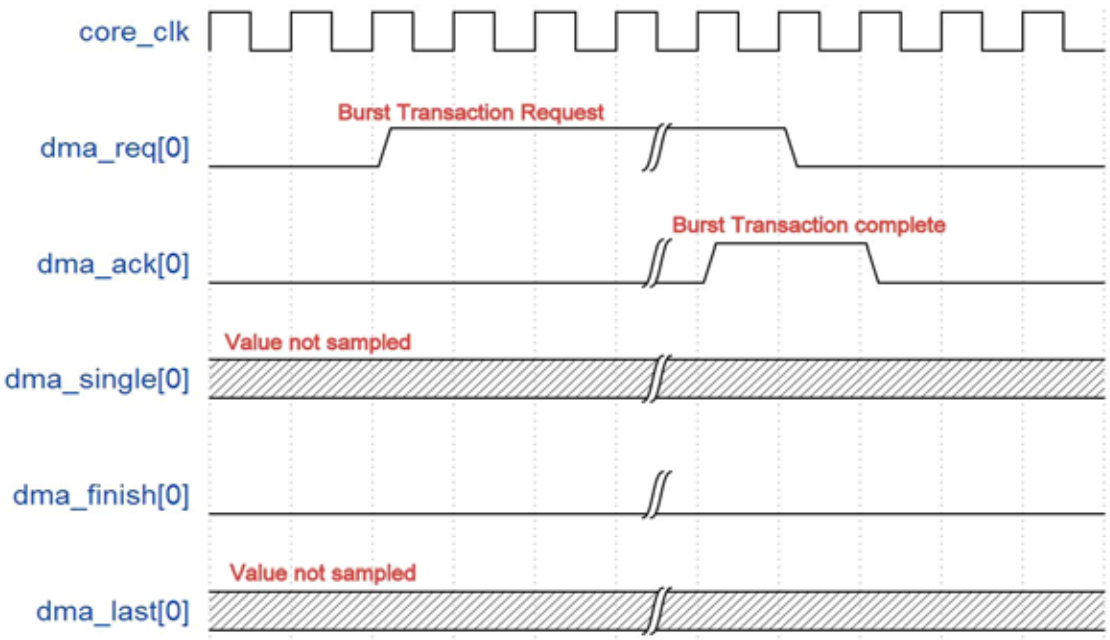
Burst Transaction – DMA Flow Controller

Figure 2-23 shows the timing diagram of a burst transaction. Because the peripheral is outside the Single Transaction Region, DW_axi_dmac does not sample dma_single[0].

The handshaking loop is as follows:

- dma_req asserted by peripheral
- dma_ack asserted by DW_axi_dmac
- dma_req de-asserted by peripheral
- dma_ack de-asserted by DW_axi_dmac

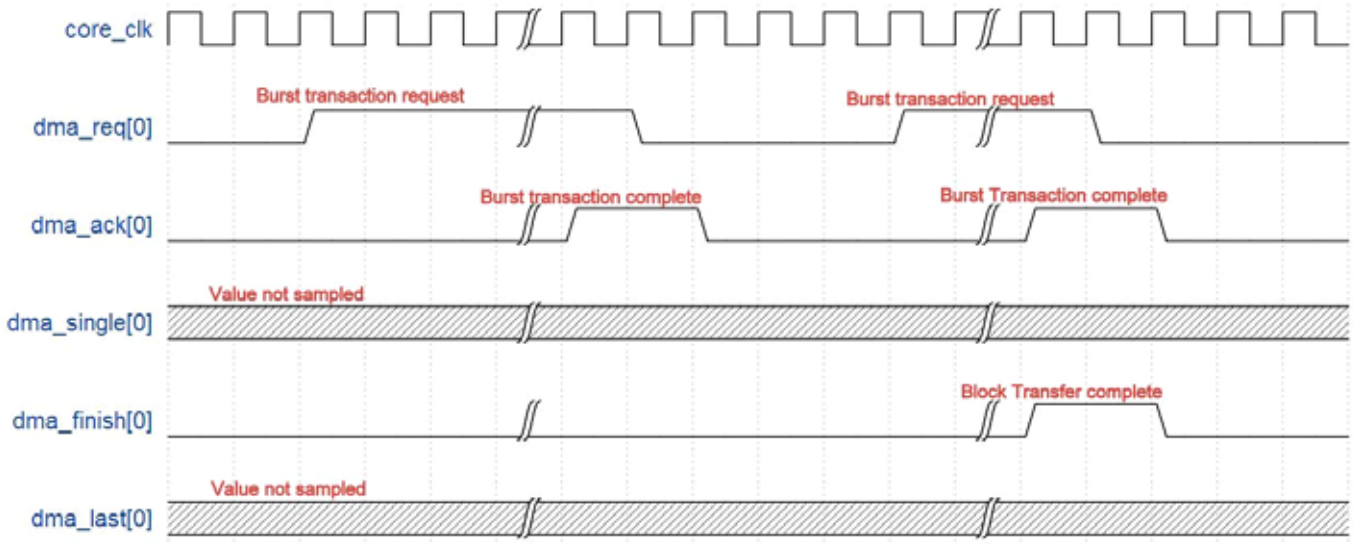
Figure 2-23 Burst Transaction



Back-to-Back Burst Transaction – DMA Flow Controller

Figure 2-24 shows the timing diagram of a back-to-back burst transaction. Because the peripheral is outside the Single Transaction Region, DW_axi_dmac does not sample dma_single[0]. The second burst terminates the block, and dma_finish[0] is asserted to indicate the block completion.

Figure 2-24 Back-to-Back Burst Transaction



Note

There are two things to note when designing the hardware handshaking interface:

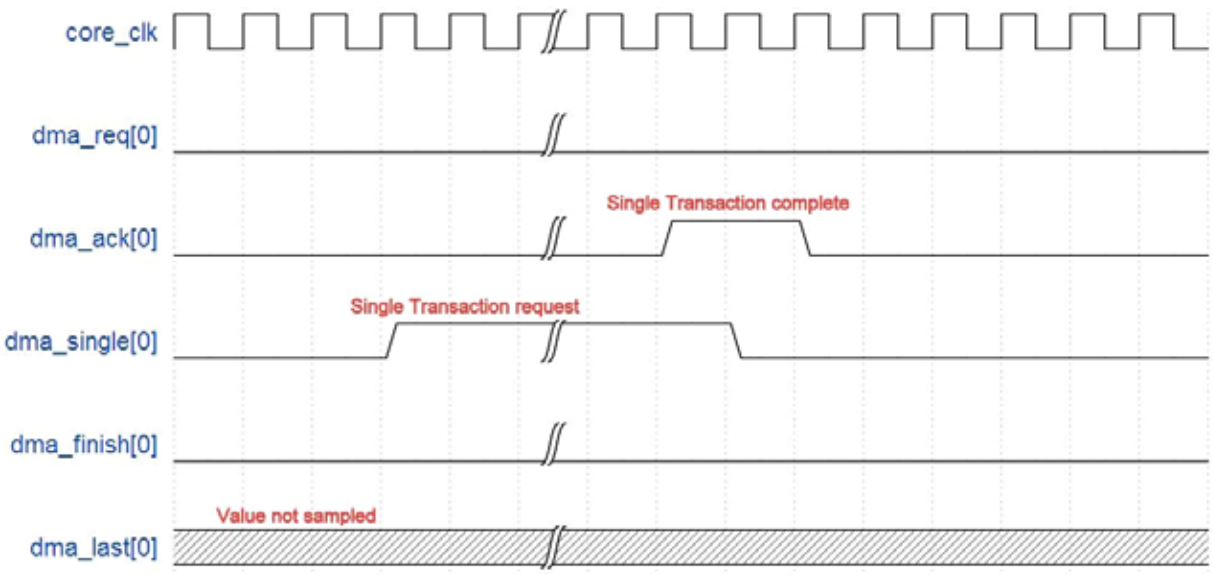
- Once asserted, the dma_req burst request signal must remain asserted until the corresponding dma_ack signal is received, even if the condition that generates dma_req in the peripheral is False.
- The dma_req signal must be de-asserted when dma_ack is asserted, even if the condition that generates dma_req in the peripheral is true.

Single Transaction – DMA Flow Controller

Figure 2-25 shows a single transaction that occurs in the Single Transaction Region. The handshaking loop is as follows:

- dma_single asserted by peripheral
- dma_ack asserted by DW_axi_dmac
- dma_single de-asserted by peripheral
- dma_ack de-asserted by DW_axi_dmac

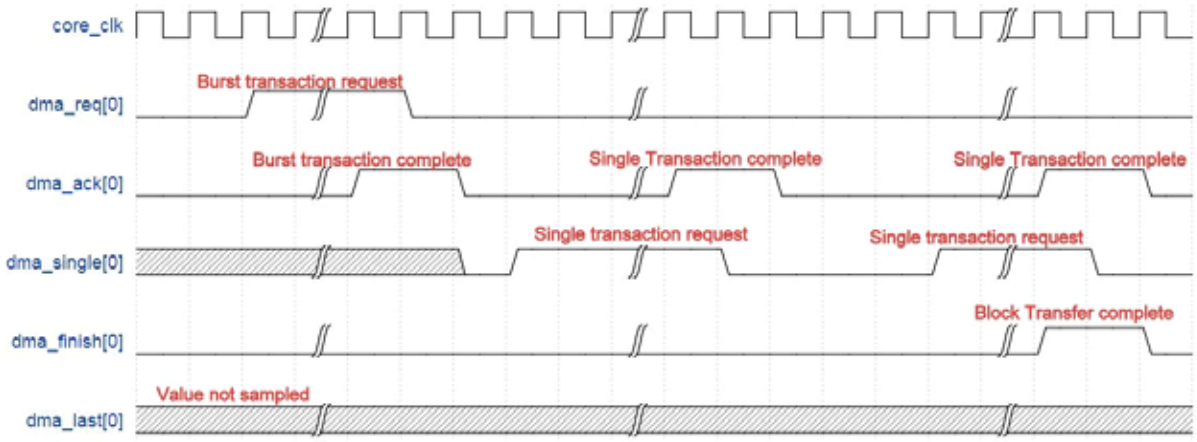
Figure 2-25 Single Transaction



Burst Followed by Back-to-Back Single Transaction – DMA Flow Controller

After the first burst transaction, the peripheral enters the Single Transaction Region and DW_axi_dmac starts sampling the dma_single. The second single transaction terminates the block transfer; dma_finish[0] is asserted to indicate block completion. Figure 2-26 illustrates this scenario.

Figure 2-26 Burst Followed by Back-to-Back Single Transaction

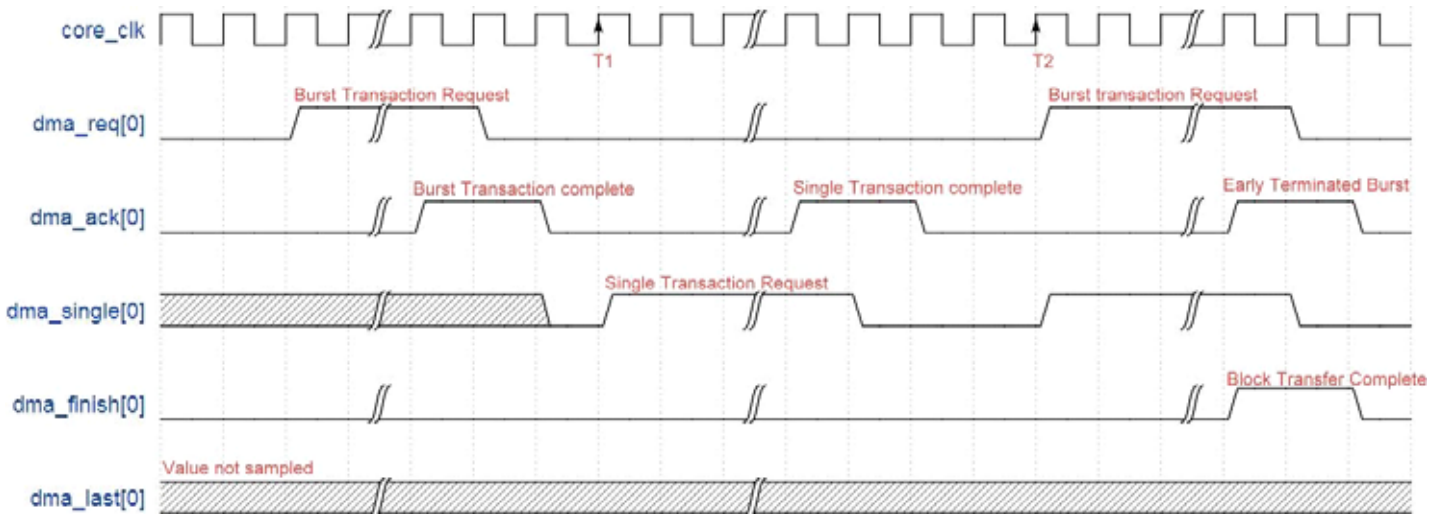


Early Terminated Burst Transaction – DMA Flow Controller

In the Single Transaction Region, if an active level on dma_req and dma_single occur on the same cycle — or if the active level on dma_single occurs in same cycle as dma_req - then the burst transaction takes precedence over the single transaction, and the block would be completed using an early terminated burst transaction.

In Figure 2-27, after the first burst DW_axi_dmac enters the single transaction region and samples dma_single[0] at T1 after that. When one single transaction is done, at time T2 dma_req[0] is asserted which results in an early terminated burst, hence completing the block indicated by dma_finish[0].

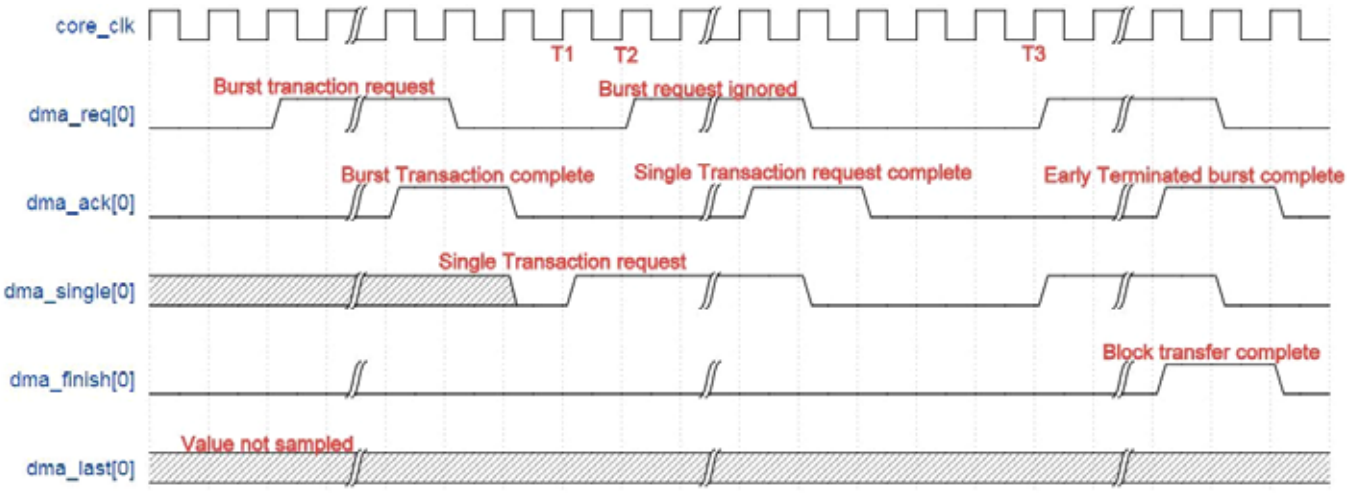
Figure 2-27 Early Terminated Burst Transfer



Burst Transaction Ignored During Active Single Transaction – DMA Flow Controller

As illustrated in Figure 2-28, after the first burst transaction completes, the peripheral is in the Single Transaction Region and DW_axi_dmac samples that dma_single[0] is asserted at T1. The dma_req[0] signal is triggered in the middle of this single transaction at time T2. This burst transaction request is ignored and is not serviced. An active edge on dma_req[0] is re-generated and sampled by DW_axi_dmac at time T3. This burst transaction completes the block transfer using an Early-Terminated Burst Transaction.

Figure 2-28 Burst Transaction Ignored During Active Single Transaction



Burst Transaction Followed by Single Transaction – Peripheral Flow Controller

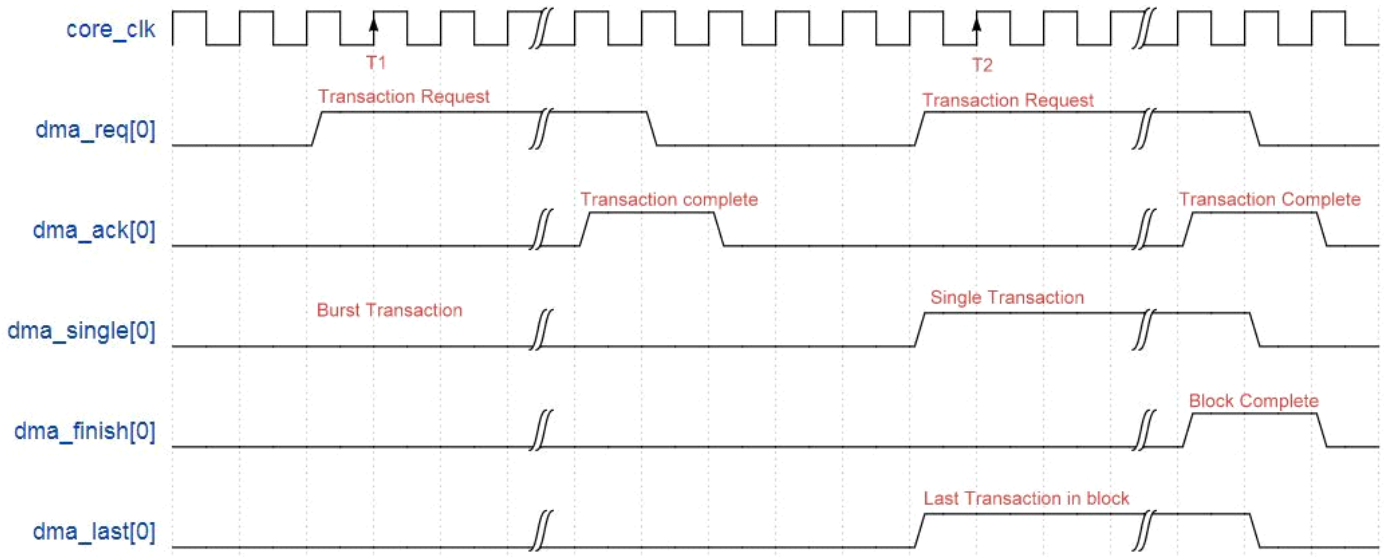
Figure 2-29 shows a burst transaction followed by a single transaction, where the single transaction is the last in the block. On clock edge T1, DW_axi_dmac samples that dma_req[0] is asserted, and dma_single[0] and dma_last[0] are de-asserted. This is a request for a burst transaction, which is not the last transaction in the block.

On clock edge T2, DW_axi_dmac samples that dma_req[0], dma_single[0], and dma_last[0] are all asserted. This is a request for a single transaction, which is the last transaction in the block.

The handshaking loop is as follows:

- dma_req along with dma_single and dma_last asserted by peripheral
- dma_ack along with dma_finish asserted by DW_axi_dmac
- dma_req, dma_single and dma_last de-asserted by peripheral
- dma_ack and dma_finish de-asserted by DW_axi_dmac

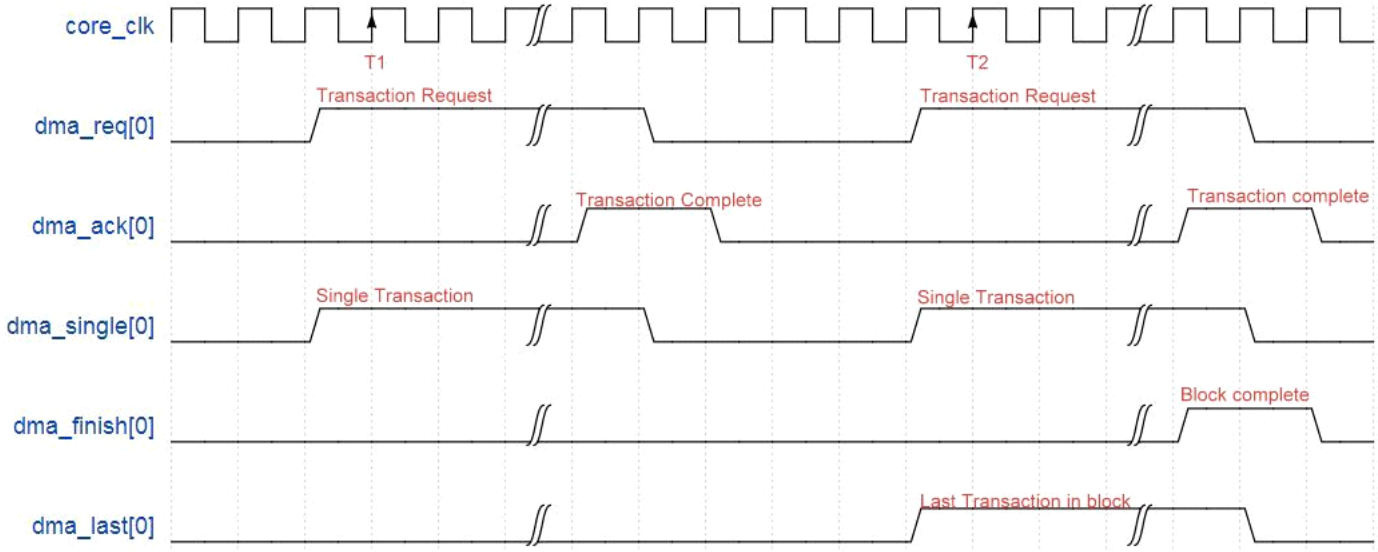
Figure 2-29 Burst Followed by Single Transaction (last in block)



Back-to-Back Single Transaction – Peripheral Flow Controller

Figure 2-30 shows a back-to-back single transaction, where the last single transaction is the last transaction in the block.

Figure 2-30 Back-to-Back Single Transation – Peripheral Flow Controller



2.10.1.2 Peripheral Interrupt Request Interface

This is a simplified version of the hardware handshaking interface. In this mode:

- The interrupt line from the peripheral is tied to the dma_req input.
- The dma_single input is tied low.
- The dma_last input is tied low.
- The dma_ack and dma_finish outputs are ignored (unconnected).

This interface can be used when the slave peripheral does not have hardware handshaking signals. To the DW_axi_dmac, this is the same situation as the case of the Hardware Handshaking Interface when the peripheral is not the flow contoller.

The peripheral can never be the flow controller because it cannot connect to the dma_last signal. The interrupt line from the peripheral is tied to the dma_req line and the timing of the interrupt line from the peripheral must be the same as the dma_req line as the case of the Hardware Handshaking Interface when the peripheral is not the flow controller.

Because the peripheral does not sample the dma_ack line, the handshaking loop is as follows:

1. Peripheral generates an interrupt that asserts “dma_req”.
2. DW_axi_dmac completes the burst transaction and generates an end-of-burst transaction interrupt, SRC_TransComp / DST_TransComp. Global Interrupt must be enabled in DMAC_CfgReg register and the corresponding channel's transaction completion indication interrupt must be enabled in CHx_IntStatus_EnableReg and CHx_IntSignal_EnableReg registers.
3. The interrupt service routine clears the interrupt in the peripheral so that the “dma_req” is de-asserted.

2.10.1.3 Asynchronous Hardware Handshake Support

The DW_axi_dmac supports hardware handshaking interface, that controls the flow of single or burst transactions. This hardware handshaking interface can be configured as synchronous or asynchronous with respect to dmac_core_clock. The DW_axi_dmac core provides the following configuration parameters that allows you to enable asynchronous hardware handshake interface. These parameters also allows you to configure each hardware handshaking interface as synchronous or asynchronous with respect to dmac_core_clock signal.

- DMAX_ASYNC_HS_EN - Includes/Excludes Asynchronous DMA Handshake Support
- DMAX_HS_SAME_ASYNC_CLK - All DMA Handshake interface has the same asynchronous clock
- DMAX_HS(y)_ASYNC_CLK - DMA Handshake interface y has asynchronous clock, where,

  y = 1...DMAX_NUM_HS_IF

All assertion and de-assertion requirements related to dma_req, dma_single, and dma_last are still applicable. See the section "Hardware Handshaking" on page 48.

2.10.2 Software Handshaking

When the slave peripheral requires the DW_axi_dmac to perform a DMA transaction, it communicates this request by sending an interrupt to the CPU or interrupt controller. The interrupt service routine then uses the software handshaking registers to initiate and control a DMA transaction. A group of software registers is used to implement the software handshaking interface. The HS_SEL_SRC/HS_SEL_DST bit in the channel configuration register CHx_CFG must be set to enable software handshaking.

The following registers are used for software handshaking:

- CHx_SWHSSrcReg – Software Handshake Register for Source of Channel x
- CHx_SWHSDstReg – Software Handshake Register for Destination of Channel x

DW_axi_dmac clears the active request bits in SWHSSrcReg to 0 when the corresponding source transaction is completed. Similarly, active request bits in SWHSDstReg are cleared to 0 when the destination transaction is completed.

The functionality of the software handshaking register bits are same as that of the corresponding hardware handshaking signals except that there are no register fields corresponding to dma_ack and dma_finish signals.

Suspension of Transfer While Using Software Handshaking

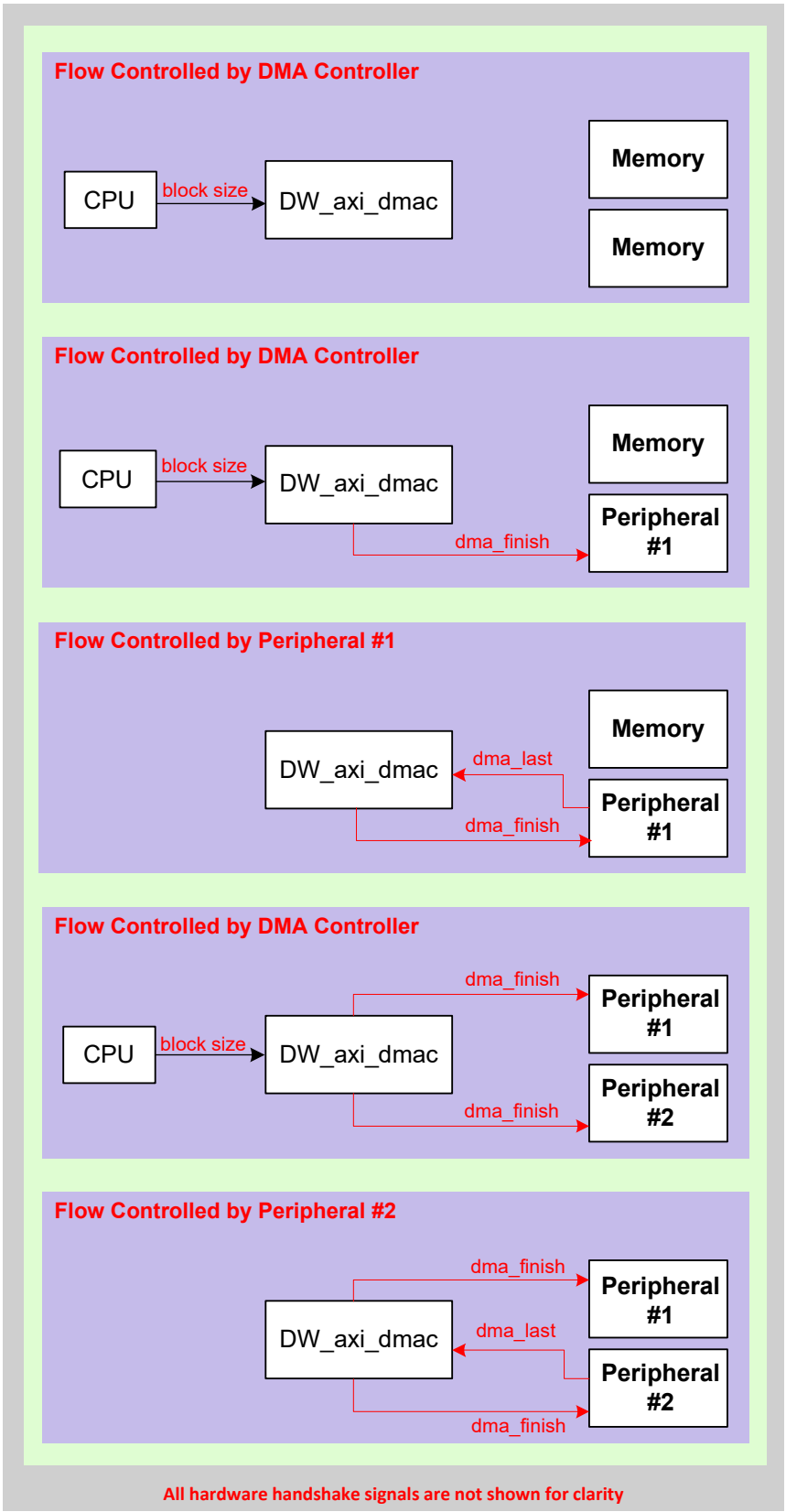
If software handshaking is used for a source and/or destination peripheral, the DMA transfer automatically stalls after completion of the requested source and/or destination transaction. The DW_axi_dmac does not proceed with the transfer until the hardware detects that software has set CHx_SWHSSrcReg.SWHS_SglReq_Src and/or CHx_SWHSDstReg.SWHS_SglReq_Dst bit to 1.

2.11 Flow Control Configurations

The transfer type and flow control configurations are decided by the value of the TT_FC field in the CHx_CFG register, which is programmable for a particular DMA transfer.

Figure 2-31 indicates different flow control configurations using hardware handshaking interfaces (a simplified version of the interface is shown). These scenarios can also be used for software handshaking, which uses software registers instead of hardware signals.

Figure 2-31 DW_axi_dmac Flow Control Configuration



2.12 Early Terminated Burst Transaction

When a source or destination peripheral is in the single transaction region, a burst transaction can still be requested. However, in this case, src_burst_size_bytes or dst_burst_size_bytes is greater than the number of bytes left to complete the source or destination block transfer at the time the burst transaction is triggered. In this case, the burst transaction is started and “early-terminated” at block completion without transferring the programmed amount of data. Only the amount of data required to complete the block transfer is transferred.

An early terminated burst transaction occurs only when the peripheral is not the flow controller.

- If the source peripheral is the flow controller and it does not have enough data for a burst transfer towards the end of a transaction, it asserts dma_single until it signals the last transaction by asserting dma_last along with dma_single. (The destination peripheral might enter the single transaction region at this point.)
- If the destination peripheral is in the single transaction region and the destination peripheral still requests a burst transaction by asserting dma_req, then DW_axi_dmac starts a burst transaction and early terminates it at block completion without transferring the programmed amount (CHx_CTL.DST_MSIZE) of data.
- If the destination peripheral is the flow controller and it does not have enough data for a burst transfer towards the end of transaction, it asserts dma_single until it signals the last transaction by asserting dma_last along with dma_single. (The source peripheral might enter the single transaction region at this point.)
- If the source peripheral is in the single transaction region and the source peripheral still requests a burst transaction by asserting dma_req, then DW_axi_dmac starts a burst transaction and early terminates it at block completion without transferring the programmed amount (CHx_CTL.SRC_MSIZE) of data.

Note

When the destination peripheral is the flow controller, there can be data loss if the following conditions are met:

- CHx_CTL.SRC_TR_WIDTH > CHx_CTL.DST_TR_WIDTH
- blk_size_bytes_dst/src_single_size_bytes != integer

The amount of data lost is:

```
src_single_size_bytes – dst_single_size_bytes
```

2.13 Transfer Control

Transfer control logic facilitates the data transfer from a source peripheral of the channel to the destination peripheral. It interacts with multiple other modules, such as the channel source and destination state machine, linked list control logic, channel registers, channel FIFO control logic and so on.

Data from the source peripheral is temporarily stored in the channel FIFO before being sent to the destination peripheral. DW_axi_dmac implements the logic needed to pack and unpack data to fit the FIFO configuration, if source and destination peripherals use different transfer size (arsize, awsize) for data transfer.

2.13.1 Single Block Transfer

If a DMA transfer consists of a single block, the software sets the multi-block type bits of both source and destination peripherals in the CHx_CFG register to 2'b00. In this case, the DW_axi_dmac disables the channel once the block transfer corresponding to the block length programmed in CHx_BLOCK_TS register is completed.

The CHx_IntStatusReg register is updated with the status corresponding to the completed block transfer. The CHx_IntStatusReg register is updated based on the settings of the CHx_IntMaskReg register and interrupts are generated based on the settings of the CHx_IntMaskReg, DMAC_IntMaskReg, and DMAC_CfgReg registers.

2.13.2 Multiblock Transfer

If DW_axi_dmac is programmed for multiblock transfers, the CHx_SAR and CHx_DAR registers are reprogrammed using either of the following methods on successive blocks of a multiblock transfer:

- Contiguous Address
- Auto Reloading
- Shadow Register
- Linked List

The CHx_CTL and CHx_BLOCK_TS registers are reprogrammed using either of the following methods on successive blocks of a multiblock transfer:

- Auto Reloading
- Shadow Register
- Linked List

The CHx_IntStatusReg register is updated with the status corresponding to the completed block transfer. The CHx_IntStatusReg register is updated based on the settings of the CHx_IntMaskReg and interrupts are generated based on the settings of the CHx_IntMaskReg, DMAC_IntMaskReg, and DMAC_CfgReg registers.

Note

- If the coreConsultant parameter DMAX_CHx_MULTI_BLK_EN is set to 0, the logic for multiblock transfer is disabled and only single block transfers are allowed.
- If a multiblock transfer is enabled, there must be at least two blocks in the complete DMA transfer for Contiguous Address and Auto-Reloading-based multiblock transfers. Otherwise, it will result in unpredictable behavior.

The register update method for multiblock transfers depends on the value of the multiblock type bits in the CHx_CFG register. All possible combinations of the multiblock register update methods are captured in the following table.

Table 2-4 Register Update Methods for Multiblock Transfer

DMAX_CHx_MULTI_BLK_EN	Multiblock Type Bits in CHx_CFG Register		Register Update Method				Multiblock Transfer Type (MLTBLK_TFR_TYPE)	Remarks
	SRC_MLTBLK_TYPE	DST_MLTBLK_TYPE	CHx_SAR	CHx_DAR	CHx_CTL	CHx_BLOCK_TS		
0	X X	X X	No update	No update	No update	No update	-	Single block
1	0 0	0 0	No update	No update	No update	No update	-	Single block or last block of multiblock
1	0 0	0 1	Contiguous	Reloaded from initial value	Reloaded from initial value	Reloaded from initial value	1	-
1	0 0	1 0	Contiguous	Loaded from shadow register	Loaded from shadow register	Loaded from shadow register	2	
1	0 1	0 0	Reloaded from initial value	Contiguous	Reloaded from initial value	Reloaded from initial value	3	-
1	0 1	0 1	Reloaded from initial value	Reloaded from initial value	Reloaded from initial value	Reloaded from initial value	4	-
1	0 1	1 0	Reloaded from initial value	Loaded from shadow register	Loaded from shadow register	Loaded from shadow register	5	-
1	1 0	0 0	Loaded from shadow register	Contiguous	Loaded from shadow register	Loaded from shadow register	6	-
1	1 0	0 1	Loaded from shadow register	Reloaded from initial value	Loaded from shadow register	Loaded from shadow register	7	-
1	1 0	1 0	Loaded from shadow register	Loaded from shadow register	Loaded from shadow register	Loaded from shadow register	8	-
1	0 0	1 1	Contiguous	Loaded from next LLI	Loaded from next LLI	Loaded from next LLI	9	-
1	0 1	1 1	Reloaded from initial value	Loaded from next LLI	Loaded from next LLI	Loaded from next LLI	10	-
1	1 1	0 0	Loaded from next LLI	Contiguous	Loaded from next LLI	Loaded from next LLI	11	-

Table 2-4 Register Update Methods for Multiblock Transfer (Continued)

DMAX_CHx_MULTI_BLK_EN	Multiblock Type Bits in CHx_CFG Register		Register Update Method				Multiblock Transfer Type (MLTBLK_TFR_TYPE)	Remarks
	SRC_MLTBLK_TYPE	DST_MLTBLK_TYPE	CHx_SAR	CHx_DAR	CHx_CTL	CHx_BLOCK_TS		
1	1 1	0 1	Loaded from next LLI	Reloaded from initial value	Loaded from next LLI	Loaded from next LLI	12	-
1	1 1	1 1	Loaded from next LLI	Loaded from next LLI	Loaded from next LLI	Loaded from next LLI	13	-
1	1 0	1 1	Loaded from shadow register	Loaded from next LLI	Loaded from next LLI	Loaded from next LLI	-	Invalid programming. SLVIF_MultiBlkType_ERR interrupt is generated
1	1 1	1 0	Loaded from next LLI	Loaded from shadow register	Loaded from next LLI	Loaded from next LLI	-	Invalid programming. SLVIF_MultiBlkType_ERR interrupt is generated

2.13.2.1 Contiguous Address

In this case, the address between successive blocks is selected as a continuation from the end of the previous block. Enabling the source or destination address to be contiguous between blocks is a function of CHx_CTL.SRC_MLTBLK_TYPE and CHx_CTL.DST_MLTBLK_TYPE register fields.

CHx_SAR and CHx_DAR updates cannot be selected to be contiguous at the same time. If required, this functionality can be achieved indirectly, by using linked lists. To do this, set up the LLI.CHx_SAR address of the next block descriptor to be one greater than the end address of the previous block. Similarly, set up the LLI.CHx_DAR address of the next block descriptor to be one greater than the end address of the previous block.

Note If a Contiguous-Address-based multiblock transfer is enabled, there must be at least two blocks in the complete DMA transfer. Otherwise, it will result in unpredictable behavior.

2.13.2.2 Auto Reloading

In this case, the channel transfer control registers are reloaded with their initial values at the completion of each block and these values are used for the new block. Some or all of the CHx_SAR, CHx_DAR, CHx_BLOCK_TS, and CHx_CTL channel registers are reloaded from their initial value at the start of a new block transfer, depending on the multi-block transfer type selected for source and destination peripherals.

The DW_axi_dmac does not proceed to the next block transfer until software clears the corresponding channel's block transfer complete interrupt by writing 1 to the corresponding bit in the DMAC_IntClear_Reg register if this interrupt is not masked off.

Note If Auto-Reloading-based multi-block transfer is enabled, there should be at least 2 blocks in the complete DMA transfer. Otherwise, it will result in unpredictable behavior.

2.13.2.3 Shadow Register

In this case, the channel transfer control registers are loaded from their corresponding shadow registers at the completion of each block and these values are used for the new block. Some or all of the CHx_SAR, CHx_DAR, CHx_BLOCK_TS, and CHx_CTL channel registers are loaded from their corresponding shadow registers at the start of a new block transfer, depending on the multiblock transfer type selected for source and destination peripherals.

A separate memory map is not defined for the shadow register access. Software always writes to the CHx_SAR, CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers irrespective of the type of multiblock transfer used. If a shadow-register-based multiblock transfer is used for a source or destination transfer, DW_axi_dmac internally routes the data to the corresponding shadow registers. DW_axi_dmac copies the shadow register contents to CHx_SAR, CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers before starting a new block transfer.

Read operations to the CHx_SAR, CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers always return the data corresponding to the current block transfer and not the shadow register contents (which corresponds to the next block).

For shadow-register-based multiblock transfer, the ShadowReg_Or_LLI_Valid bit in the CHx_CTL register indicates whether the shadow register contents are valid or not. Zero indicates that the shadow register contents are invalid, while 1 indicates that the shadow register contents are valid. If this bit is read as zero during a shadow register fetch phase, DW_axi_dmac discards the shadow register contents and generates a ShadowReg_Or_LLI_Invalid_ERR interrupt. DW_axi_dmac waits until the software writes any value to the CHx_BLK_TFR_ResumeReqReg register to indicate valid shadow register availability, before attempting another shadow register fetch operation to continue the next block transfer.

Note If the coreConsultant parameter DMAX_CHx_SHADOW_REG _EN is set to 0, shadow-register-based multi-block transfer is disabled and only other methods of multi-block transfers are allowed.

For more information about programming the Shadow Register, see "Programming Flow for Shadow-Register-Based Multi-Block Transfer" on page 467.

2.13.2.4 Linked List

In this case, the DW_axi_dmac reprograms the channel transfer control registers prior to the start of each block, by fetching the block descriptor for that block from system memory. This is known as an LLI update. DW_axi_dmac block chaining uses a linked list pointer register (CHx_LLP) to store the address in memory of the next linked list item. Each LLI contains the following block descriptors:

- CHx_SAR
- CHx_DAR
- CHx_BLOCK_TS
- CHx_CTL
- CHx_LLP

To set up block chaining, a sequence of linked lists should be programmed in memory. DW_axi_dmac allows dynamic extension of linked lists, which eliminates the need for creating the entire linked list in the system memory in advance. The CHx_CTL.ShadowReg_Or_LLI_Valid and CHx_CTL.LLI_Last fields of the LLI are used to achieve this functionality.

For linked-list-based multi-block transfers, the ShadowReg_Or_LLI_Valid bit of the LLI.CHx_CTL register indicates whether the linked list item fetched from the memory is valid or not. If this bit is set to 0, it indicates the LLI is invalid, while 1 indicates the LLI is valid. If the LLI is invalid, DW_axi_dmac discards the LLI and generates an LLI Error interrupt (if the corresponding channel error interrupt mask bit is set to 0). This error condition causes the DW_axi_dmac to halt the corresponding channel gracefully. DW_axi_dmac waits till software writes any value to the CHx_BLK_TFR_ResumeReqReg register to indicate the availability of a valid LLI, before attempting another LLI read operation.

In the case of LLI pre-fetching, the ShadowReg_Or_LLI_Invalid_ERR interrupt is not generated even if the ShadowReg_Or_LLI_Valid bit is set to 0 for the pre-fetched LLI. In this case, DW_axi_dmac re-attempts the LLI fetch operation after completing the current block transfer. DW_axi_dmac generates a ShadowReg_Or_LLI_Invalid_ERR interrupt only if the ShadowReg_Or_LLI_Valid bit is still set to 0.

LLI access always uses a burst size (arsize or awsize) that is the same as the data bus width and cannot be changed or programmed to anything other than this. Burst length (awlen or arlen) is chosen based on the data bus width so that the access does not cross one complete LLI structure of 64 bytes. DW_axi_dmac fetches the entire LLI (40 bytes) in one AXI burst, if the burst length is not limited by other setting.

If the status write back option is enabled, DW_axi_dmac writes back CHx_CTL, CHx_LLP_STATUS, CHx_SSTAT, and CHx_DSTAT information to the location defined for this field, which is from address [CHx_LLP] + 0x20 to [CHx_LLP] + 0x34. The CHx_SSTAT and CHx_DSTAT write back can be independently enabled or disabled in coreConsultant and by programming the corresponding field in the

CHx_CTL register. If CHx_SSTAT and / or CHx_DSTAT write back is not enabled, DW_axi_dmac de-asserts the write data strobes corresponding to these write operations.

Note

- CHx_LLP_STATUS[63] and CHx_LLP_STATUS[62] indicates DMA_TFR_DONE and BLOCK_TFR_DONE status respectively and these are the last fields to be updated during LLI write-back. Software should wait until BLOCK_TFR_DONE bit is set to 1 before reading CHx_SSTAT and CHx_DSTAT information. DMA_TFR_DONE bit will be set to 1 along with BLOCK_TFR_DONE bit after transferring the last block in DMA transfer.
- LLI.CHx_CTL.ShadowReg_Or_LLI_Valid bit will be 0 after the LLI write-back operation.

Figure 2-32 DW_axi_dmac Linked List Item (Descriptor)

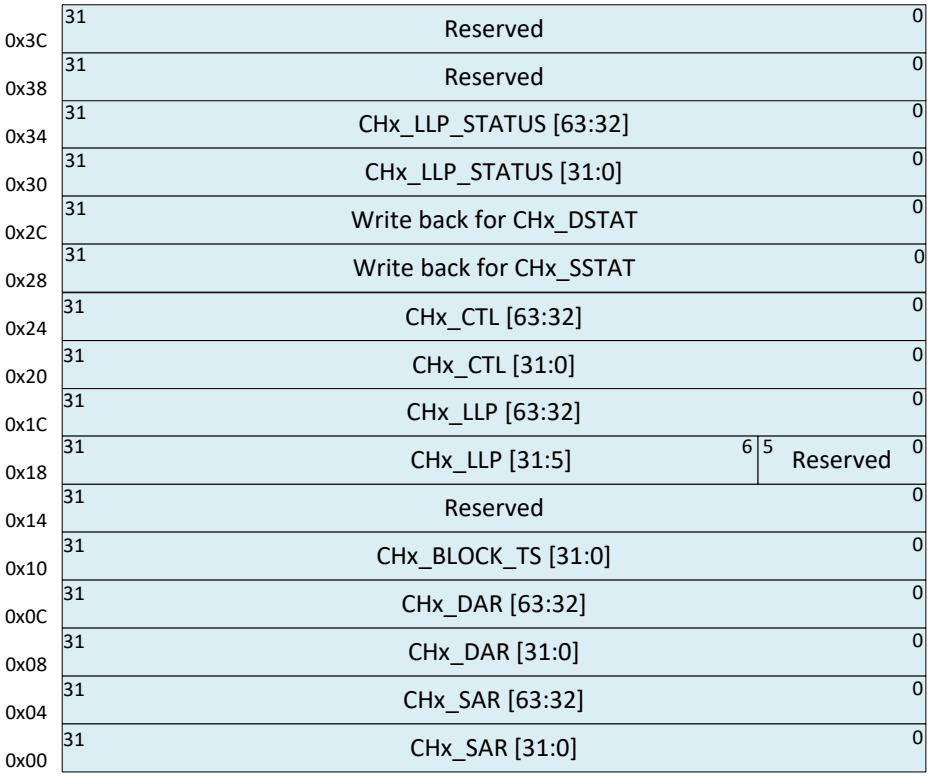
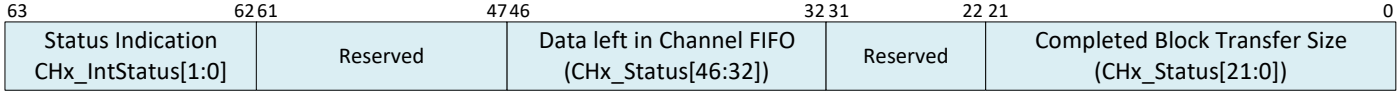


Figure 2-33 CHx_LLP_STATUS Write-Back Field of LLI



For more information about programming linked-list-based multi-block transfers, see “Programming Flow for Linked-List-Based Multi-Bock Transfer” on page 470.

2.13.2.5 Suspension of Transfers Between Blocks

At the end of every block transfer, a Block Transfer Done Interrupt is asserted if:

- Global Interrupt is enabled (DMAC_CfgReg.INT_EN = 1)
- The channel block transfer completion interrupt is enabled (CHx_InStatus_EnableReg.Enable_BLOCK_TFR_DONE_IntStat = 1 AND CHx_IntSignal_EnableReg. Enable_BLOCK_TFR_DONE_IntSignal = 1 AND CHx_CTL.IOC_BLKTFR = 1)

For Contiguous Address and Auto-Reloading-based multiblock transfers (neither source nor destination peripheral uses Shadow Register or Linked-List-based multiblock transfers), the DMA transfer automatically stalls after the Block Transfer Done Interrupt is asserted, if the Block Transfer Done Interrupt is enabled and unmasked. The DW_axi_dmac does not proceed to the next block transfer until a write to the appropriate field in CHx_IntClearReg register, done by software to clear the channel Block Transfer Done Interrupt, is detected by hardware.

Channel suspension between blocks is used to ensure that the Block Transfer Done ISR (Interrupt Service Routine) of the next-to-last block is serviced before the start of the final block commences. This ensures that the ISR has cleared the CHx_CFG.SRC_MLTBLK_TYPE and / or CHx_CFG.DST_MLTBLK_TYPE bits before completion of the final block. The CHx_CFG.SRC_MLTBLK_TYPE and / or CHx_CFG.DST_MLTBLK_TYPE bits should be cleared in the Block Transfer Done ISR for the next-to-last block transfer.

2.13.2.6 End of Multi-Block Transfers

If either source or destination peripheral uses Shadow Register or Linked-List-based multi-block transfers, the corresponding last block indication bit, ShadowReg_Or_LLI_Last, in the CHx_CTL_ShadowReg / LLI.CHx_CTL register indicates whether the current block is the last block in the transfer or not. If this bit is set to 1 for the current block, the DW_axi_dmac understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of current block transfer.

For Contiguous Address and Auto-Reloading-based multiblock transfers (if neither source nor destination peripheral uses Shadow Register or Linked-List-based multi-block transfers), if the corresponding multi-block type selection bits namely CHx_CFG.SRC_MLTBLK_TYPE and / or CHx_CFG.DST_MLTBLK_TYPE bits are 2'b00 at the end of a block transfer, the DW_axi_dmac understands that the previous block was the final block in the transfer and completes the DMA transfer operation.

2.14 AXI Unaligned Transfer Support

The DW_axi_dmac supports the generation of unaligned DMA transfers on the AXI master interfaces. This feature can be enabled through a configurable parameter DMAX_UNALIGNED_XFER_EN.

The DW_axi_dmac component supports backward compatibility in such a way that even if this feature is not required, the old drivers and register programming works without any change.

2.14.1 Description of AXI Unaligned Transfers

The unaligned memory accesses are common in the AXI-based systems, where the memory access address it not aligned to the natural boundary of the transfer width. The DW_axi_dmac supports the generation of the unaligned memory DMA transfers on the AXI master interface. This feature support eliminates the following restrictions for the memory accesses:

- **Source AXI data transfers** - SAR addresses must align to the source transfer width that is CHx_CTL. SRC_TR_WIDTH (AXI bus arsize)
- **Destination AXI data transfers** - DAR addresses must align to the destination transfer width that is CHx_CTL. DST_TR_WIDTH (AXI bus awsize)

Thus, user can program CHx_SAR and CHx_DAR registers with any address, and it can be unaligned or aligned to the source or destination transfer width for memory accesses. The unaligned access is supported for combinations of CHx_CFG.TT_FC where the memory access is involved (MEM_TO_PER_DMAC, PER_TO_MEM_DMAC, PER_TO_MEM_SRC, MEM_TO_PER_DST).

DW_axi_dmac handles the packing and the unpacking of the data considering the unaligned offset at both source and destination transfers. The data received from the unaligned source transaction is packed to align with FIFO width, and then it is stored in the FIFO. At the destination end, the data is read from the FIFO and unpacked based on the CHx_DAR register and destination transfer width. The invalid bytes of the destination transfer are invalidated by driving the respective write strobes to 0. The following section provides more details about unaligned transfer based on DMA, source, or destination as a flow controller.

2.14.1.1 DW_axi_dmac as a Flow Controller

The definition of the CHx_BLOCK_TS register remains the same that is the number programmed into BLOCK_TS field indicates the total number of data width CHx_CTL.SRC_TR_WIDTH to be transferred in a DMA block transfer. If the source address is unaligned, then the unaligned bytes in the source are considered to derive at the total bytes transferred in a DMA block transfer.

The equation to calculate the total number of valid bytes transferred in a block for an unaligned transfer is as follows:

blk_size_bytes_dma_u = (CHx_BLOCK_TS.BLOCK_TS * src_single_size_bytes) - Source Unaligned bytes

- The DW_axi_dmac fetches the blk_size_bytes_dma_u bytes of data from the source and transfers all bytes to the destination based on the CHx_DAR register.
- The destination unaligned bytes does not increase or decrease the valid number of bytes transferred in a DMA block. Only the source unaligned bytes affect the total number of bytes transferred in a DMA block.
- The unaligned transfer is supported for combinations of Transfer Type and Flow Controller wherever memory is involved (MEM_TO_MEM_DMAC, MEM_TO_PER_DMAC, PER_TO_MEM_DMAC).
- When Multi Block Type is contiguous for either source or destination, initial address of the subsequent block is computed as below (Unaligned nature of the SAR/DAR also result in sub-sequent block initial address to be unaligned):
  - **Source Contiguous:** Next CHx_SAR = Current CHx_SAR + blk_size_bytes_dma_u
  - **Destination Contiguous:** Next CHx_DAR = Current CHx_DAR + blk_size_bytes_dma_u + Destination Unaligned bytes

2.14.1.2 Source is the Flow Controller

The equation to calculate the total number of valid bytes transferred in a block transfer for an unaligned transfer is as follows:

blk_size_bytes_src_u = (Number of source burst transactions in block * src_burst_size_bytes) + (Number of source single transactions in block * src_single_size_bytes) - Source Unaligned bytes

- The DW_axi_dmac fetches the blk_size_bytes_src_u bytes of data from the source and transfers all bytes to the destination based on the CHx_DAR register.
- The destination unaligned bytes does not increase or decrease the valid number of bytes transferred in DMA block. Only the source unaligned bytes affect the total number of bytes transferred in a DMA block.
- The unaligned transfer is supported for combinations of Transfer Type and Flow Controller wherever a memory is involved - PER_TO_MEM_SRC, when the source is the Flow Controller.
- When Multi Block Type is Contiguous for either source or destination, initial address the subsequent block is computed as below (Unaligned nature of the SAR / DAR also results in sub-sequent block initial address to be unaligned):
  - **Source Contiguous:** Next CHx_SAR = Current CHx_SAR + blk_size_bytes_src_u
  - **Destination Contiguous:** Next CHx_DAR = Current CHx_DAR + blk_size_bytes_src_u + Destination Unaligned bytes

2.14.1.3 Destination is the Flow Controller

The equation to calculate the total number of valid bytes transferred in a block transfer for an unaligned transfer is as follows:

blk_size_bytes_dst_u = (Number of destination burst transactions in block * dst_burst_size_bytes) + (Number of destination single transactions in block * dst_single_size_bytes) - Destination Unaligned bytes

- The DW_axi_dmac always provides the blk_size_bytes_dst_u bytes of data to the destination.
- If source address is unaligned, then DW_axi_dmac fetches an additional src_single_size_bytes to compensate the source unaligned bytes and to provide blk_size_bytes_dst_u bytes to the destination for a block.
  - Along with that, in few scenarios when destination is unaligned, then DW_axi_dmac fetches

  ceil (Destination Unaligned bytes / src_single_size_bytes) amount of additional data from the source.
  - For example, if CHx_CTL.SRC_TR_WIDTH = 64 bit, CHx_CTL.DST_TR_WIDTH = 64 bit, CHx_CFG.TT_FC = MEM_TO_PER_DST, CHx_CTL.DST_MSIZE = 16 data items, CHx_SAR = 0x1007 (source is unaligned), and CHx_DAR = 0xF008 (destination is aligned).
    - Number of bytes to be fetched from Source for a destination request (DST_REQ_DATA)
      = CHx_CTL.DST_MSIZE* CHx_CTL.DST_TR_WIDTH
      = 16*8
      = 128 bytes
    - Data can be fetched from the source for the first destination request
      = (DST_REQ_DATA / CHx_CTL.SRC_TR_WIDTH)* CHx_CTL.SRC_TR_WIDTH - source unaligned bytes
      =(128 / 8)*8 - 7
      = 121 bytes

From the previous example, source needs to fetch 7 more bytes of data for the first destination request - because of the unaligned source transfer. Hence, DW_axi_dmac fetches an additional src_single_size_bytes (CHx_CTL.SRC_TR_WIDTH) of data to compensate unaligned bytes.

- The unaligned transfer is supported for combinations of Transfer Type and Flow Controller wherever a memory is involved - MEM_TO_PER_DST, when Destination is the Flow Controller.
- When Multi Block Type is Contiguous for either source or destination, initial address the subsequent block is computed based on the following formula (Unaligned nature of the SAR / DAR also result in sub-sequent block initial address to be unaligned):
  - **Source Contiguous:** Next CHx_SAR = Current CHx_SAR (Aligned) + blk_size_bytes_dst_u + Source Unaligned bytes
  - **Destination Contiguous:** Next CHx_DAR = Current CHx_DAR + blk_size_bytes_dst_u

2.14.2 AXI Unaligned Transfer Examples

Following examples explain how the unaligned transfers on the source and destination are handled, with the different configuration and programming options.

- Configuration parameters:
  - DMAX_M_DATA_WIDTH - AXI Master interface data bus width.
  - DMAX_CH(x)_FIFO_WIDTH (internal parameter) - channel x FIFO width, function of (DMAX_M_DATA_WIDTH, DMAX_CHx_STW, DMAX_CHx_DTW).
  - DMAX_STATIC_ENDIAN_SELECT_MSTIF and DMAX_ENDIAN_FORMAT_MSTIF – endianness selection
- Programming registers:
  - CHx_SAR - Source Address Register of DMA transfer
  - CHx_DAR – Destination Address Register of DMA transfer
  - CHx_CTL.SRC_MSIZE – Source Burst Transaction Length. Each of width CHx_CTL.SRC_TR_WIDTH
  - CHx_CTL.DST_MSIZE – Destination Burst Transaction Length. Each of width CHx_CTL.DST_TR_WIDTH
  - CHx_CTL.SRC_TR_WIDTH - Source Transfer Width (arsize).
  - CHx_CTL.DST_TR_WIDTH - Destination Transfer Width (awsize).
  - CHx_CFG.TT_FC – Transfer Type and Flow Control.

Assumptions

In the following examples, it is assumed that,

- Each row of the DMA burst / single transaction can be accomplished either through a single AXI transfer or a burst transfer based on the DMAX_CHx_MAX_AMBA_BURST_LENGTH and CHx_CTL.ARLEN / CHx_CTL.AWLEN.
- The unaligned transfers shown in these examples are only the DMA Burst / Single transactions corresponding to a particular block. The whole DMA transfers may have multiple blocks.

- The default endianness is from little-endian to little-endian unless it is explicitly specified otherwise.
- For information on the number of bytes transferred in a DMA transfer, see the section "AXI Unaligned Transfer Support" on page 69.

Conventions

Following are the conventions used for the bytes of the DMA transfer.

	Invalid byte of the AXI DMA Transfer
	Valid byte of the DMA transfer with N as the byte address for source transfers

Note If the source and/or destination transfers are unaligned, then AXI burst type must be INCR for both source and destination transfers (CHx_CTL.SINC/DINC to be programmed as 0). In this case, burst type cannot be programmed to FIXED for both source and destination. At the system level FIXED unaligned transfer support is not required because of the following reasons:

- FIXED transfers are directed to the FIFOs. But these FIFOs does not have strobe signals to selectively write the bytes.
- Unaligned FIXED transfer has deep impact on the system bandwidth utilization, and hence the system performance.

For example, if the source transfer width is 512 and the source is unaligned by 63. Then each of the AXI beat received has only one byte. This results in major impact on the bandwidth utilization.

2.14.2.1 Example 1

Assume that,

- the source address is unaligned to the source transfer width
- the destination address is unaligned to the destination transfer width

Table Table 2-5 describes all major DMA parameter settings considered for this example.

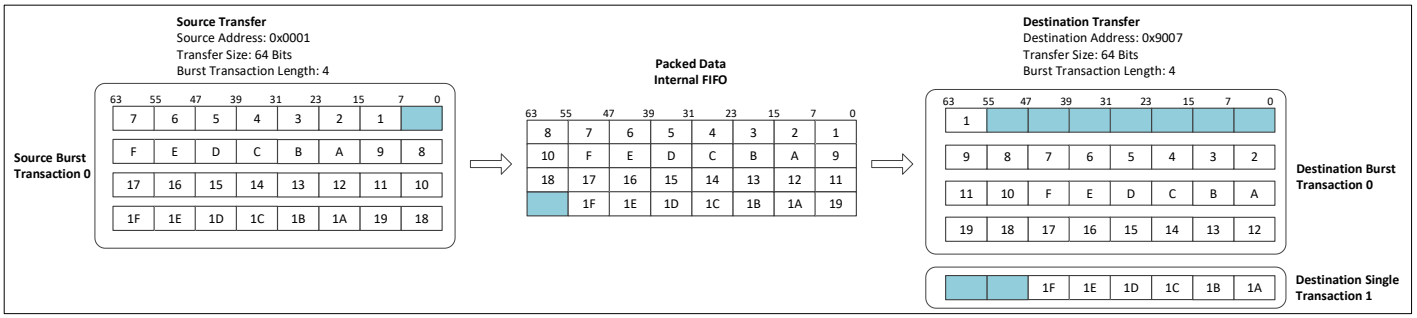
Table 2-5 Example 1 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		

Table 2-5 Example 1 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_SAR	0x0001	Unaligned source address
CHx_DAR	0x9007	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-34 Example 1 - Unaligned DMA Transfer Handling



In this example, the single source burst transaction gets converted to multiple burst / single transaction - even though the following programming options remain the same:

- Source and destination transaction length
- Source and destination transfer widths are same

This is due to the unaligned source address. Figure 2-35 shows the source transfer where the source address 0x0001 is unaligned. This unaligned source transfer has 4 beats, with the AWSIZE of 3 that is 8 bytes. In the first data beat received, the least significant byte is ignored as the address is unaligned with one byte. All other bytes are packed into FIFO as shown in Figure 2-34.

Figure 2-35 Example 1 - Unaligned Source Transfer Waveform

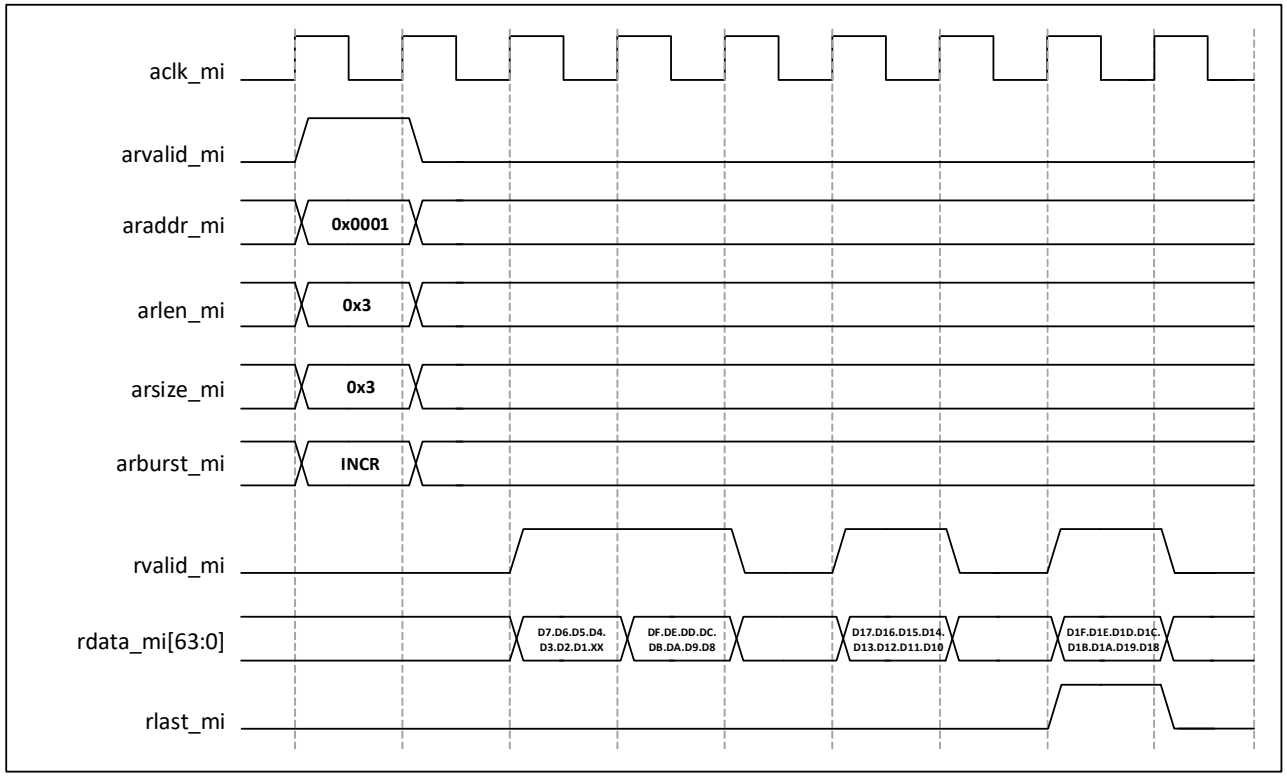
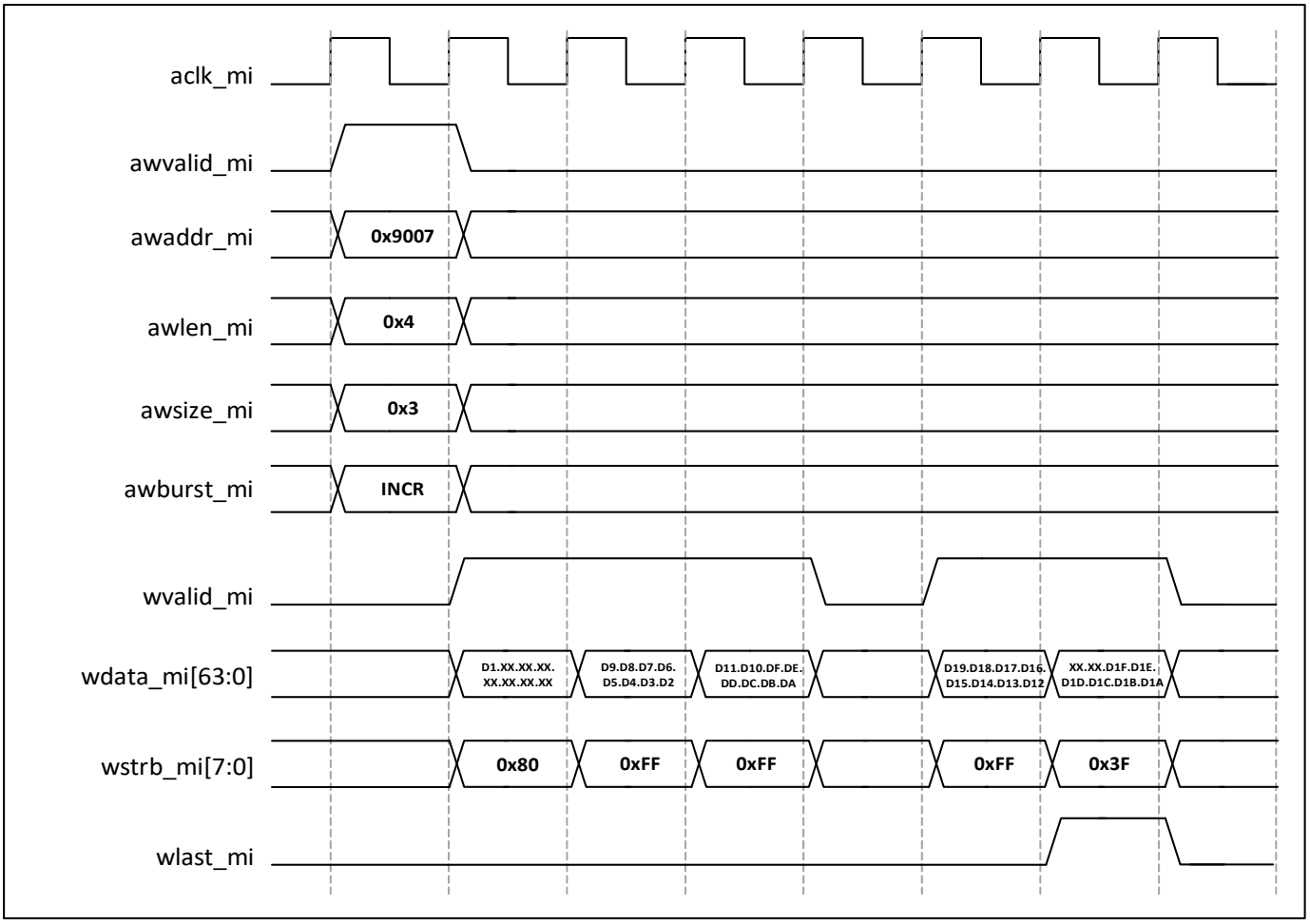


Figure 2-36 shows the destination transfer where the destination address is unaligned - 0x9007. This unaligned destination transfer has 5 beats, with the AWSIZE of 3 that is 8 bytes. In the first write data beat generated, only the most significant data byte is valid as the address is unaligned with 7 bytes, and the write strobe is asserted as 0x80 (as only most significant byte is valid). For all the other write data beats, write strobe is asserted as 0xFF as all bytes are valid - except for the last write data beat. For the last write data beat, only lower 6 data bytes are valid and the other two bytes are invalid - hence strobe is asserted as 0x3F.

Figure 2-36 Example 1 - Unaligned Destination Transfer Waveform



2.14.2.2 Example 2

Assume that

- the source address is unaligned to source transfer width
- destination address is aligned to the destination transfer width

Table 2-6 describes all major DMA parameter settings considered for this example.

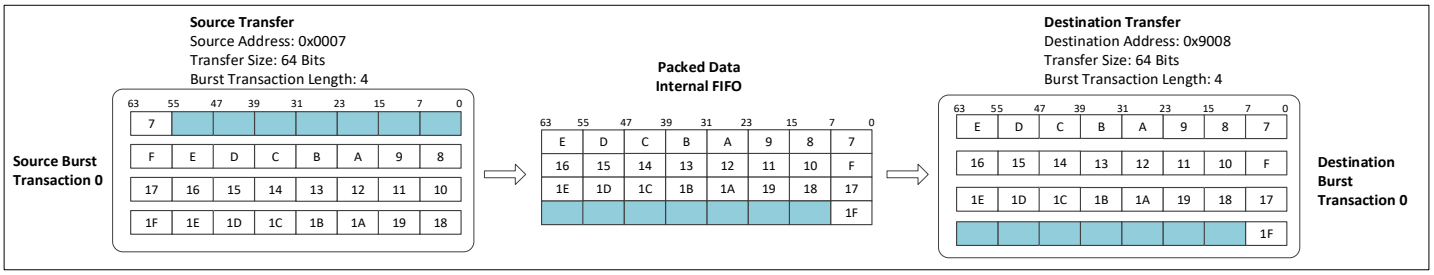
Table 2-6 Example 2 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		

Table 2-6 Example 2 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_SAR	0x0007	Unaligned source address
CHx_DAR	0x9008	Aligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-37 Example 2 - Unaligned DMA Transfer Handling



2.14.2.3 Example 3

Assume that,

- the source address is aligned to source transfer width
- destination address is unaligned to the destination transfer width

Table Table 2-7 describes all major DMA parameter settings considered for this example.

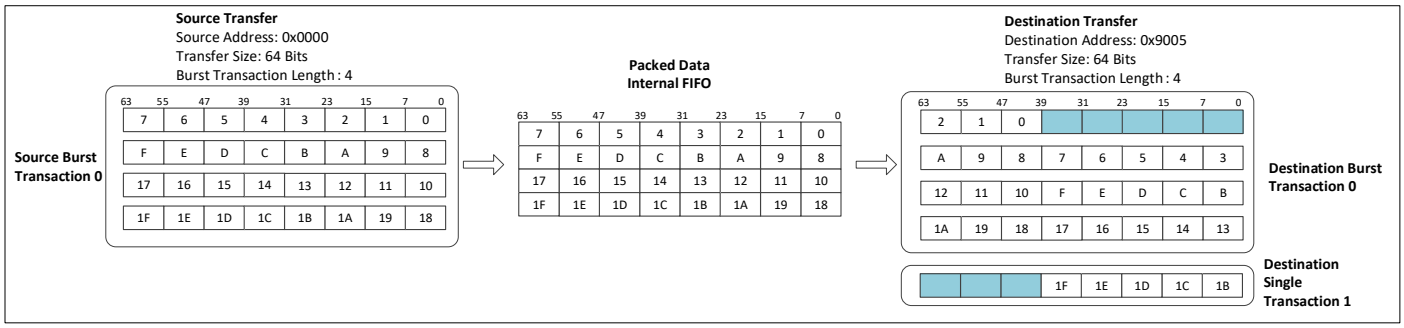
Table 2-7 Example 3 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0000	Aligned source address
CHx_DAR	0x9005	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4

Table 2-7 Example 3 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-38 Example 3 - Unaligned DMA Transfer Handling



2.14.2.4 Example 4

Assume that,

- the source address is unaligned to the source transfer width,
- the destination address is also unaligned to the destination transfer width
- the source transfer width is not the same as the master interface data bus width with multiple source burst / single DMA transactions

Table Table 2-8 describes all major DMA parameter settings considered for this example.

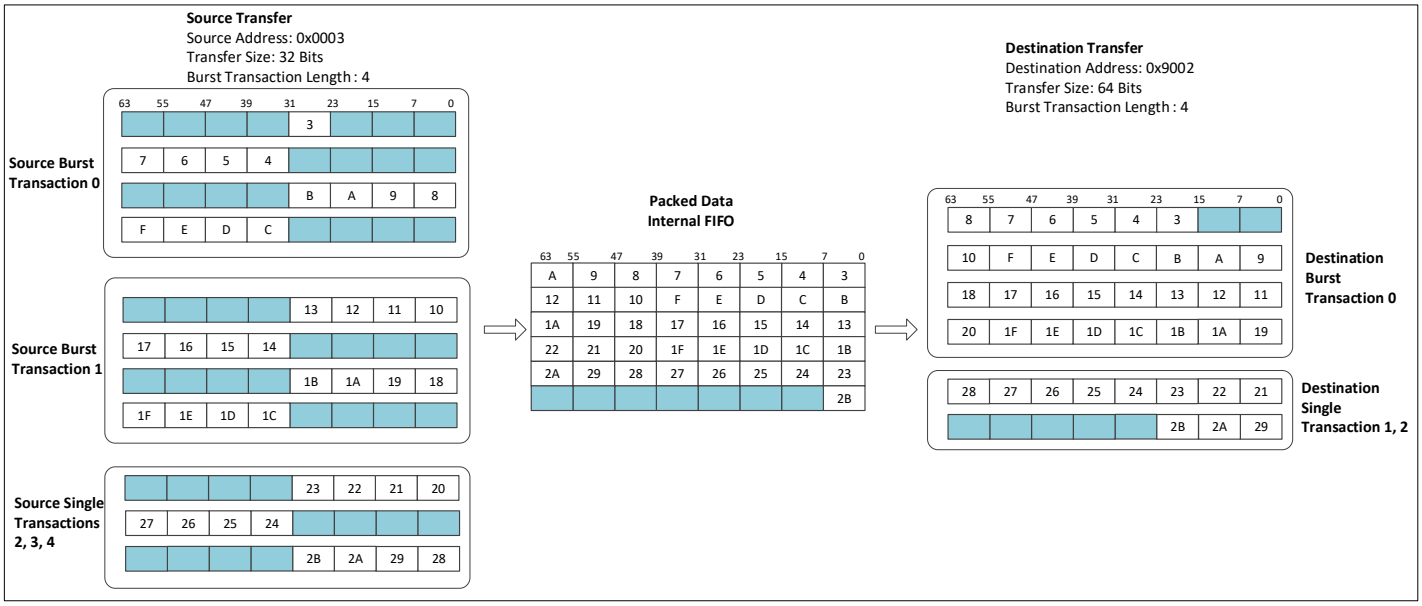
Table 2-8 Example 4 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0003	Unaligned source address
CHx_DAR	0x9002	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x2	Source Transfer Width is 32 bits

Table 2-8 Example 4 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-39 Example 4 - Unaligned DMA Transfer Handling



2.14.2.5 Example 5

Assume that,

- the source address is unaligned to the source transfer width
- the destination address is also unaligned to the destination transfer width
- the destination transfer width is not the same as the master interface data bus width with multiple destination burst / single DMA transactions

Table Table 2-9 describes all major DMA parameter settings considered for this example.

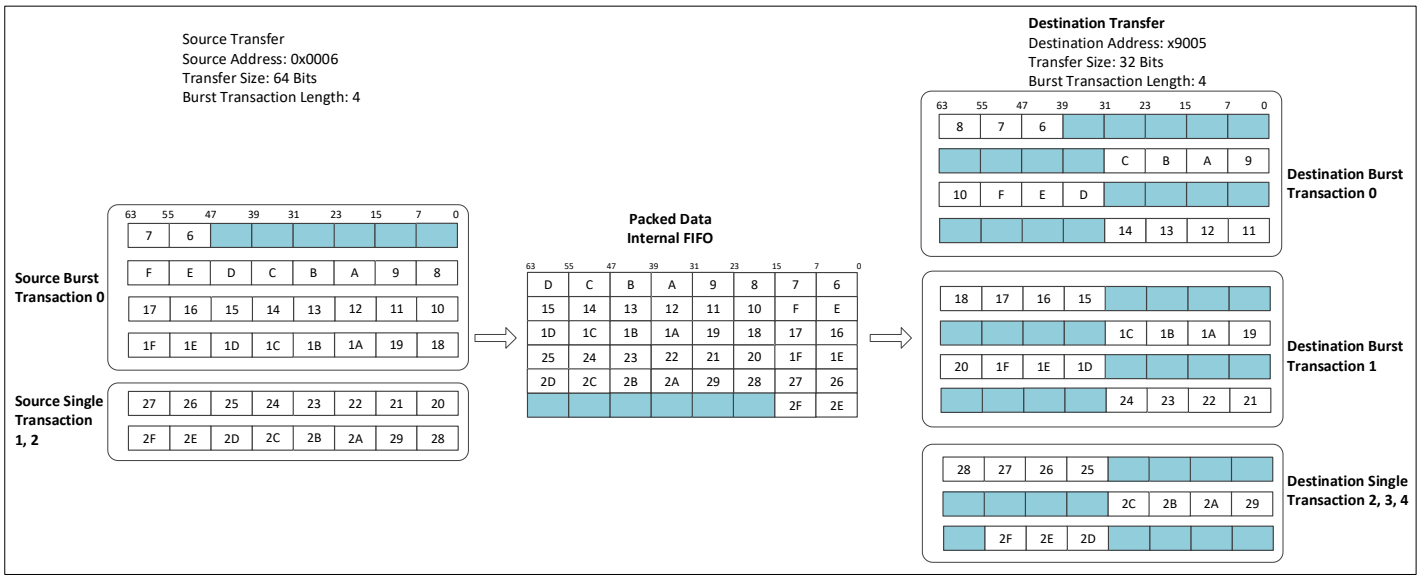
Table 2-9 Example 5 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0006	Unaligned source address

Table 2-9 Example 5 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_DAR	0x9005	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x2	Destination Transfer Width is 32 bits

Figure 2-40 Example 5 - Unaligned DMA Transfer Handling



2.14.2.6 Example 6

Assume that,

- the source address is unaligned to source transfer width
- the destination address is also unaligned to the destination transfer width
- both the source and the destination transfer widths are not the same as the master interface data bus width with multiple source burst / single DMA transactions.

Table Table 2-10 describes all major DMA parameter settings considered for this example.

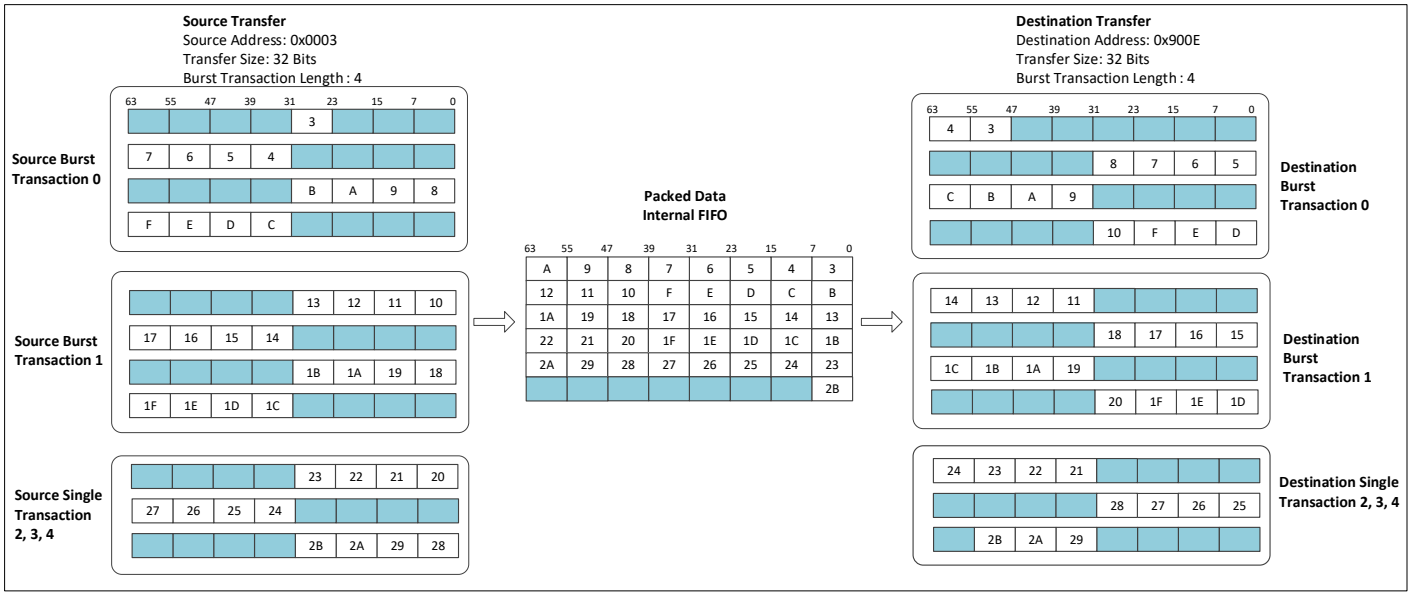
Table 2-10 Example 6 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit

Table 2-10 Example 6 - Configuration and Programming Options

Configuration/Programming	Value	Description
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0003	Unaligned source address
CHx_DAR	0x900E	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x2	Source Transfer Width is 32 bits
CHx_CTL.DST_TR_WIDTH	0x2	Destination Transfer Width is 32 bits

Figure 2-41 Example 6 - Unaligned DMA Transfer Handling



2.14.2.7 Example 7

Assume that,

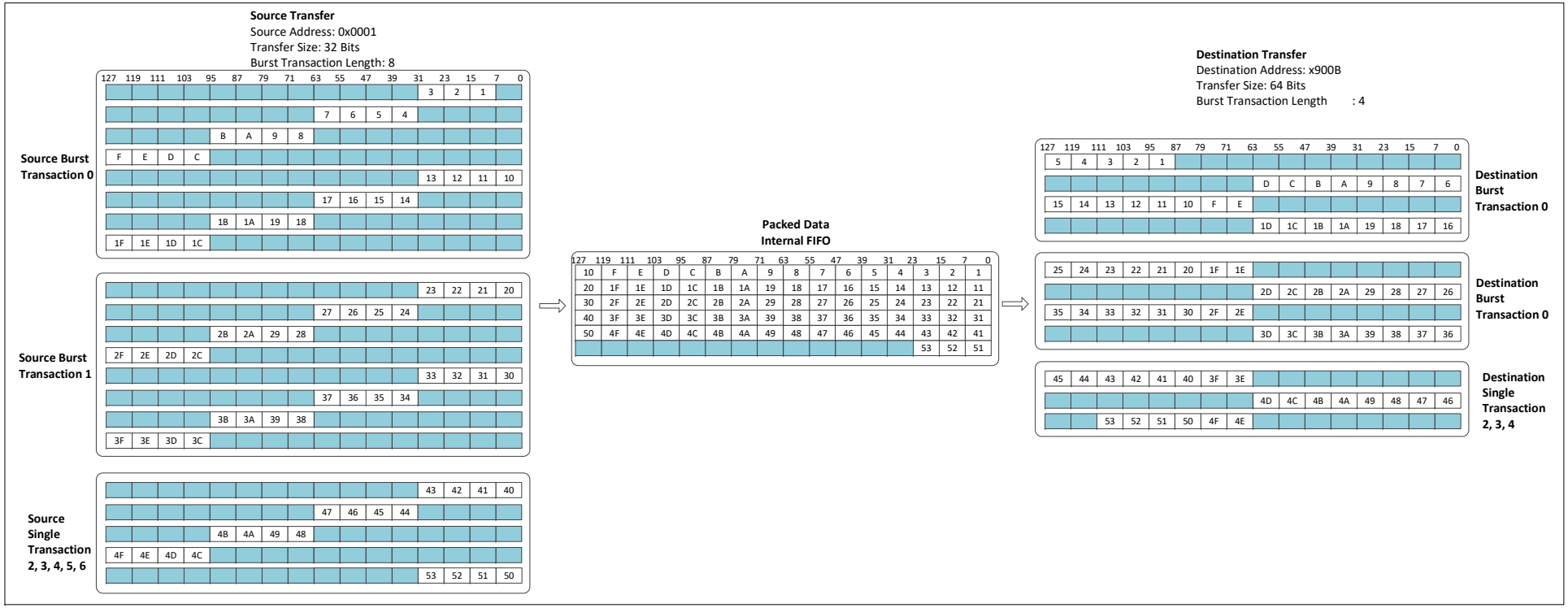
- the source address is unaligned to source transfer width
- the destination address is also unaligned to the destination transfer width
- both the source and the destination transfer widths are not the same as the master interface data bus width with multiple source burst/single DMA transactions

Table Table 2-11 describes all major DMA parameter settings considered for this example.

Table 2-11 Example 7- Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	128	128 bit
DMAX_CH(x)_FIFO_WIDTH	128	128 bit
Programming Register		
CHx_SAR	0x0001	Unaligned source address
CHx_DAR	0x900B	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b002	Source burst transaction length = 8
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x2	Source Transfer Width is 32 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-42 Example 7 - Unaligned DMA Transfer Handling



2.14.2.8 Example 8

Assume that,

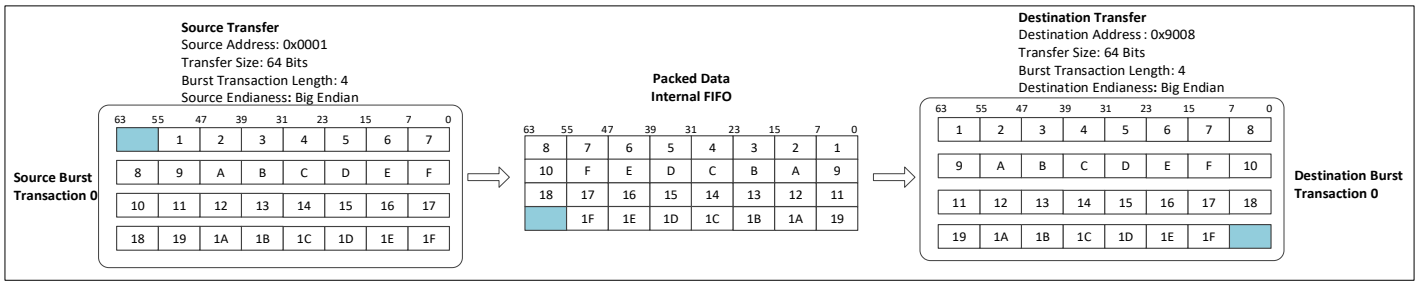
- the source address is unaligned to the source transfer width
- the destination address is aligned to destination transfer width
- the source and the destination endianness is big-endian

Table Table 2-12 describes all major DMA parameter settings considered for this example.

Table 2-12 Example 8 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0001	Unaligned source address
CHx_DAR	0x9008	Aligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-43 Example 8 - Unaligned DMA Transfer Handling



2.14.2.9 Example 9

Assume that,

- the source address is unaligned to the source transfer width
- the destination address is also unaligned to the destination transfer width
- the source and the destination endianness is big-endian

Table Table 2-13 describes all major DMA parameter settings considered for this example.

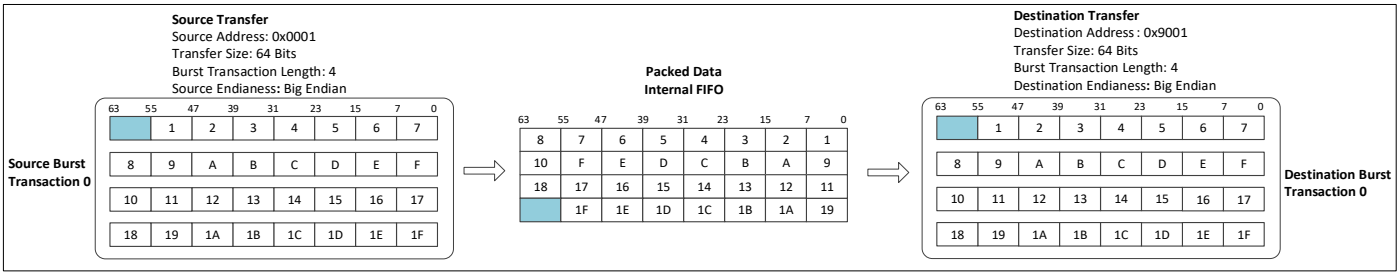
Table 2-13 Example 9 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit

Table 2-13 Example 9 - Configuration and Programming Options

Configuration/Programming	Value	Description
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0001	Unaligned source address
CHx_DAR	0x9001	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-44 Example 9 - Unaligned DMA Transfer Handling



2.14.2.10 Example 10

Assume that,

- the source address is aligned to the source transfer width
- the destination address is unaligned to the destination transfer width
- the source and the destination endianness is big-endian

Table Table 2-14 describes all major DMA parameter settings considered for this example.

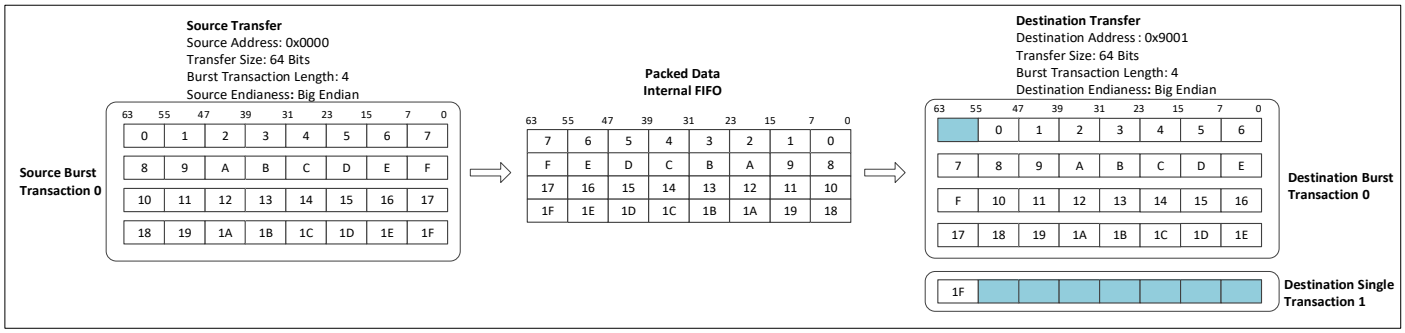
Table 2-14 Example 10 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0000	Aligned source address

Table 2-14 Example 10 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_DAR	0x9001	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x3	Source Transfer Width is 64 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-45 Example 10 - Unaligned DMA Transfer Handling



2.14.2.11 Example 11

Assume that,

- the source address is unaligned to the source transfer width
- the destination address is also unaligned to the destination transfer width
- the source transfer width is not the same as the master interface data bus width with multiple source burst / single DMA transactions
- the source and destination endianness is big-endian

Table Table 2-15 describes all major DMA parameter settings considered for this example.

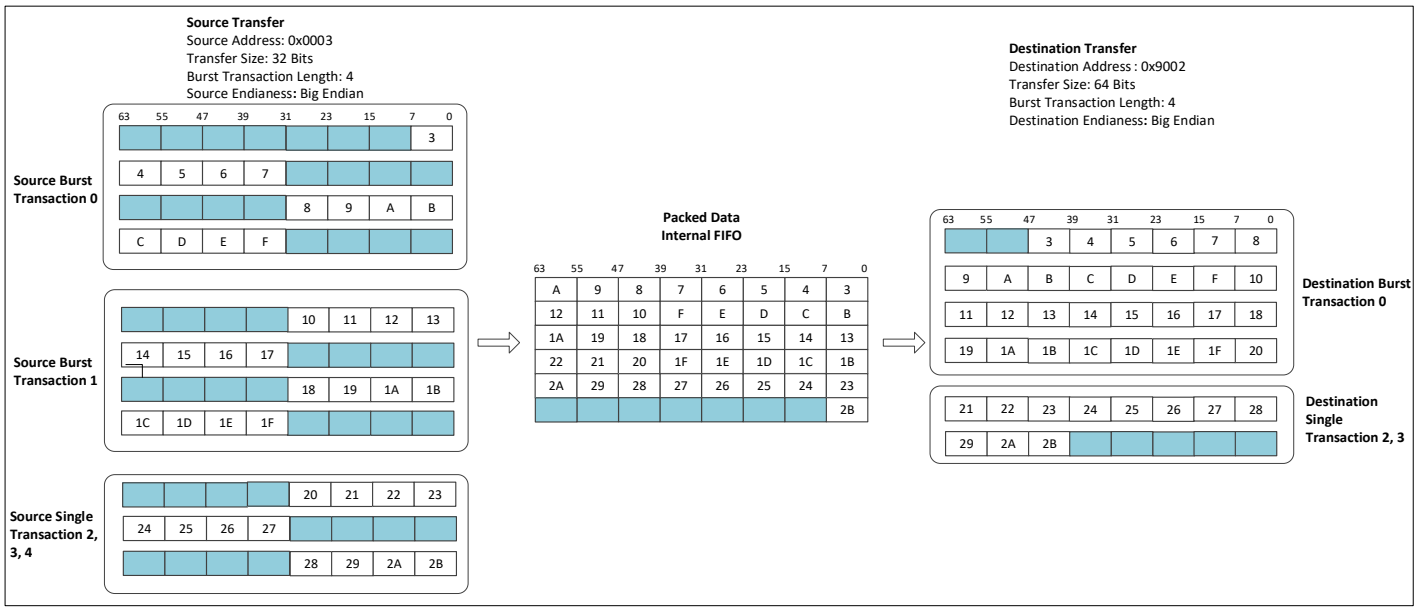
Table 2-15 Example 11 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0003	Unaligned source address

Table 2-15 Example 11 - Configuration and Programming Options

Configuration/Programming	Value	Description
CHx_DAR	0x9002	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x2	Source Transfer Width is 32 bits
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-46 Example 11 - Unaligned DMA Transfer Handling



In “Example 11”, the data byte-3 is packed with data byte-4, 5, and 6 to form a data item of source transfer width. This data item is used to perform the little-endian conversion as shown in the Figure 2-14 that is all bytes of the data item are flipped. Similar operation is continued, on the subsequent bytes.

BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Word Access)

2.14.2.12 Example 12

Assume that,

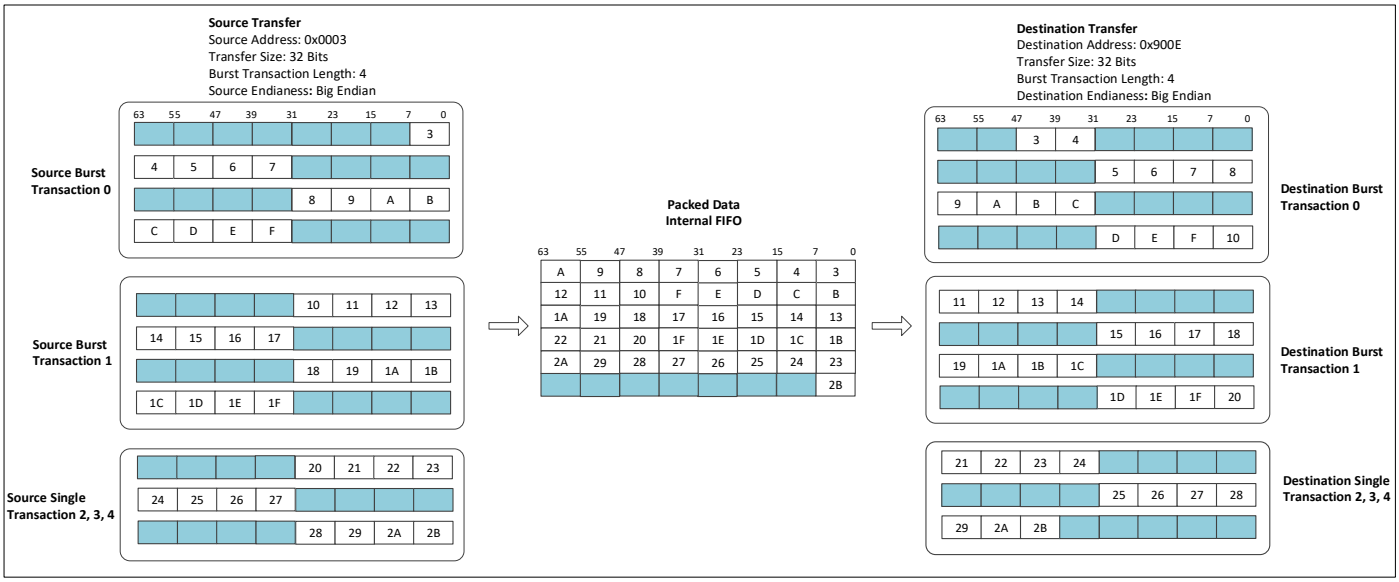
- the source address is unaligned to the source transfer width
- the destination address is also unaligned to the destination transfer width
- the source and the destination transfer widths are not the same as the master interface data bus width
- the source and the destination endianness is big-endian

Table Table 2-16 describes all major DMA parameter settings considered for this example.

Table 2-16 Example 12 - Configuration and Programming Options

Configuration/Programming	Value	Description
Configuration Parameter		
DMAX_M_DATA_WIDTH	64	64 bit
DMAX_CH(x)_FIFO_WIDTH	64	64 bit
Programming Register		
CHx_SAR	0x0003	Unaligned source address
CHx_DAR	0x900E	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4
CHx_CTL.DST_MSIZE	3b001	Destination burst transaction length = 4
CHx_CTL.SRC_TR_WIDTH	0x2	Source Transfer Width is 32 bits
CHx_CTL.DST_TR_WIDTH	0x2	Destination Transfer Width is 32 bits

Figure 2-47 Example 12 - Unaligned DMA Transfer Handling



In “Example 12”, the data byte-3 is packed with data byte-4, 5, and 6 to form a data item of source transfer width. This data item is used to perform the big-endian conversion (As per the Figure 2-14) that is all bytes of the data item are flipped. Similar operation is continued, on the subsequent bytes.

2.15 Channel Suspend, Disable, and Abort

Under normal operation, software enables a channel by writing a 1 to the channel enable register, DMAC_ChEnReg.CH_EN, and hardware disables a channel on transfer completion by clearing the DMAC_ChEnReg.CH_EN register.

Software can suspend, disable, or abort a channel before a transfer completes. The suspend, disable, and abort procedures are explained in the following sections.

2.15.1 Channel Suspend

To suspend a channel during DMA transfer:

1. Software writes a 1 to the channel suspend bit CH_SUSP in the channel enable register, DMAC_ChEnReg.
2. DW_axi_dmac gracefully halts all transfers from the source peripheral, after completing all AXI transfers initiated on the source peripheral.
3. DW_axi_dmac sets CHx_IntStatusReg.CH_SRC_SUSPENDED bit to 1 to indicate that source data transfer is suspended and generates the interrupt if it is not masked off.

Note

If the channel FIFO is full and the destination peripheral is not requesting data transfer, DW_axi_dmac cannot receive any more data on the corresponding master interface, which could lead to a deadlock.

- The requests initiated by source/destination/LLI state machines and present in the Master Interface Read Address Channel and Write Address Channel FIFOs to be sent on the AXI Master Interface will be sent on the AXI Master Interface even if the Channel Suspend request is initiated. Based on the Master Interface Read Address and Write Address Channel FIFO depth configuration, maximum 8 read/write requests may be initiated and DW_axi_dmac waits for the data/response for these requests also before suspending the channel.

4. DW_axi_dmac transfers all the data in channel FIFO to destination peripheral.

   When CHx_CTL.SRC_TR_WIDTH < CHx_CTL.DST_TR_WIDTH and the DMAC_ChEnReg.CH_SUSP bit is high, there may still be data in the channel FIFO, but not enough to form a single transfer of CHx_CTL.DST_TR_WIDTH. The data remaining in the channel FIFO will be transferred to destination if channel is resumed later which results in filling of more data in channel FIFO.
5. DW_axi_dmac clears channel locking and resets the channel locking settings in the CHx_CFG register.
6. DW_axi_dmac sets the CHx_IntStatusReg.ChLock_Cleared bit to 1 to indicate that channel locking is cleared.
7. DW_axi_dmac sets the CHx_IntStatusReg.CH_SUSPENDED bit to 1 to indicate that the channel is suspended.
8. DW_axi_dmac generates a CH_SUSPENDED interrupt (if it is not masked off).

After a channel suspend, software may either resume the channel after some time, or disable the channel.

2.15.2 Channel Suspend and Resume

To suspend and resume a channel:

1. Follow steps 1 to 4 in Channel Suspend.
2. Software writes a 0 to the channel suspend bit CH_SUSP in the channel enable register, DMAC_ChEnReg.
3. DW_axi_dmac resumes the DMA transfer from the point where it got suspended.

Note	Once software initiates the channel suspend procedure by writing a 1 to the channel suspend bit DMAC_ChEnReg.CH_SUSP, writing 0 to DMAC_ChEnReg.CH_SUSP bit to resume the channel before DW_axi_dmac asserts CHx_IntStatusReg.CH_SUSPENDED bit is not allowed. DW_axi_dmac ignores this write operation.

2.15.3 Channel Suspend and Disable Prior to Transfer Completion

To suspend and disable a channel:

1. Follow steps 1 to 4 in Channel Suspend.
2. To disable the suspended channel using software, write a 0 to the channel enable bit (CH_EN) in the channel enable register (DMAC_ChEnReg) after DW_axi_dmac asserts the CHx_IntStatusReg.CH_SUSPENDED bit to 1 to indicate that channel is suspended.

   When CHx_CTL.SRC_TR_WIDTH < CHx_CTL.DST_TR_WIDTH and the DMAC_ChEnReg.CH_SUSP bit is high, there may still be data in the channel FIFO, but not enough to form a single transfer of CHx_CTL.DST_TR_WIDTH.

   In this scenario, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral and is lost.
3. DW_axi_dmac sets the CHx_IntStatusReg.CH_DISABLED bit to 1 to indicate that the channel is disabled.
4. DW_axi_dmac generates a CH_DISABLED interrupt (if it is not masked off).
5. DW_axi_dmac clears the DMAC_ChEnReg.CH_EN bit to 0.

2.15.4 Channel Disable Prior to Transfer Completion without Suspend

To disable a channel without suspending:

1. Software writes a 0 to the channel enable bit CH_EN in the channel enable register, DMAC_ChEnReg.
2. DW_axi_dmac gracefully halts all transfers from the source peripheral, after completing all AXI transfers initiated on the source peripheral.

Note

- If the channel FIFO is full and the destination peripheral is not requesting data transfer, DW_axi_dmac cannot receive any more data on the corresponding master interface, which could lead to a deadlock.
- The requests initiated by source/destination/LLI state machines and present in the Master Interface Read Address Channel and Write Address Channel FIFOs to be sent on the AXI Master Interface will be sent on the AXI Master Interface even if the Channel Suspend request is initiated. Based on the Master Interface Read Address and Write Address Channel FIFO depth configuration, maximum 8 read/write requests may be initiated and DW_axi_dmac waits for the data/response for these requests also before suspending the channel.

3. DW_axi_dmac transfers all the data in the channel FIFO to the destination peripheral.

   If CHx_CTL.SRC_TR_WIDTH is less than CHx_CTL.DST_TR_WIDTH and the DMAC_ChEnReg.CH_EN bit is low, there may still be data in the channel FIFO, but not enough to form a single transfer of CHx_CTL.DST_TR_WIDTH.

   In this scenario, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral and is lost.
4. DW_axi_dmac clears channel locking and resets the channel locking settings in the CHx_CFG register.
5. DW_axi_dmac sets the CHx_IntStatusReg.ChLock_Cleared bit to 1 to indicate that channel locking is cleared.
6. DW_axi_dmac sets the CHx_IntStatusReg.CH_DISABLED bit to 1 to indicate that the channel is disabled.
7. DW_axi_dmac generates a CH_DISABLED interrupt (if it is not masked off).
8. DW_axi_dmac clears the DMAC_ChEnReg.CH_EN bit to 0.

Note

Once software initiates a channel disable procedure by writing a 0 to the channel enable bit DMAC_ChEnReg.CH_EN, writing a 1 to the DMAC_ChEnReg.CH_EN bit to re-enable the channel before DW_axi_dmac asserts the CHx_IntStatusReg.CH_DISABLED bit is not allowed. DW_axi_dmac ignores this write operation.

2.15.5 Abnormal Channel Abort

Aborting a channel is not a recommended procedure. This procedure should be used only when software wants to disable a channel without resetting the entire DW_axi_dmac, for example, if a particular channel hangs due to not receiving a response from the corresponding handshaking interface. Before aborting a channel, it is recommended that you first attempt to disable a channel.

To abort a channel:

1. Software writes a 1 to the channel abort bit CH_ABORT in the channel enable register, DMAC_ChEnReg.
2. DW_axi_dmac gracefully halts all transfers from the source/destination peripheral after completing all AXI transfers initiated on the source/destination peripheral.
3. The data in Channel FIFO is flushed and essentially be lost.
4. DW_axi_dmac clears channel locking and resets the channel locking settings in the CHx_CFG register.
5. DW_axi_dmac sets CHx_IntStatusReg.ChLock_Cleared bit to 1 to indicate that channel locking is cleared.
6. DW_axi_dmac sets CHx_IntStatusReg.CH_ABORTED bit to 1 to indicate that channel is aborted.
7. DW_axi_dmac generates CH_ABORTED interrupt if it is not masked off.
8. DW_axi_dmac clears DMAC_ChEnReg.CH_EN bit to 0.

2.16 Debug Interface

The Debug Interface in DW_axi_dmac consists of Status Indication signals and DMAC Hold Control signals.

2.16.1 Slave Interface and DMAC Core Status Indication

DW_axi_dmac supports status indication using two signals on the I/O: slvif_busy and dmac_busy.

- slvif_busy – indicates whether the Slave Interface is busy. This signal is asserted High if there is an active transfer on the slave interface.
- dmac_busy – indicates whether the DMAC Core is busy or not. This signal is asserted High if any of the channels in DW_axi_dmac are in a non-idle state. A non-idle state of the channel corresponds to the following cases.
  - There is an active transfer on the Master Interface (including posted requests) corresponding to one or multiple Channels.
  - Any of the Channels are about to initiate a transfer, which might correspond to waiting for the grant of the master interface from the arbiter.

2.16.2 DMAC Hold Control

DW_axi_dmac supports freezing an operation. This is supported using two signals on the I/O, dmac_hold_req and dmac_hold_ack. The dmac_hold_req signal is the request to put DW_axi_dmac in hold

(freeze) mode. Asserting this request puts the entire DW_axi_dmac in freeze mode without violating the AXI protocol. DW_axi_dmac asserts dmac_hold_ack after entering the hold mode.

To put DW_axi_dmac in hold mode:

1. External device (master) asserts dmac_hold_req to put DW_axi_dmac in hold mode.
2. DW_axi_dmac halts all transfers from the source peripheral of all channels.
   - DW_axi_dmac does not initiate any new read request on the AXI read address channel.
   - DW_axi_dmac de-asserts arvalid_m*N* output once arready_m*N* is received.
   - DW_axi_dmac de-asserts rready_m*N* output so that no new data is received on the AXI read data channel and hence no new data is written to the channel FIFO.

Note There may be outstanding read requests of different channels on the master interfaces, for which data transfer is not completed. This can lead to timeout issues in the interconnect.

3. DW_axi_dmac halts all transfers to the destination peripheral of all channels, which means the following is true:
   - DW_axi_dmac does not initiate any new write request on AXI write address channel.
   - DW_axi_dmac de-asserts awvalid_m*N* and wvalid_m*N* output when awready_m*N* and wready_m*N* is received.
   - DW_axi_dmac de-asserts bready_m*N* output when data transfer on the AXI write data channel is completed.

Note There may be write requests from many channels on the master interfaces, waiting for a response. This can lead to timeout issues in the interconnect.

4. The channel FIFOs are not guaranteed to be emptied. There may still be data in the channel FIFO after entering the hold mode.
5. DW_axi_dmac asserts dmac_hold_ack to indicate the entry to hold mode.
6. To exit the DMAC hold mode, dmac_hold_req can be de-asserted after DW_axi_dmac asserts dmac_hold_ack.
7. After after dmac_hold_req is de-asserted, DW_axi_dmac de-asserts dmac_hold_ack.

Note When a DMAC hold request is initiated by asserting dmac_hold_req, de-asserting dmac_hold_req to exit the DMAC hold mode before DW_axi_dmac asserts dmac_hold_ack is not allowed. DW_axi_dmac ignores this operation.

2.16.3 Error Handling

DW_axi_dmac can receive an error response from a source peripheral, a destination peripheral, or an LLP peripheral. Upon occurrence of an error in an AXI transfer (source or destination data transfer or status fetch, LLI fetch, or LLI write-back), the following occurs:

1. DMA transfer in progress stops gracefully.
2. If channel locking to arbiter is enabled at any transfer level, the locking is cleared.
3. Error Status bits in CHx_IntStatusReg register is updated if not masked off.
4. Source / Destination Transaction Completion, Block Transfer Completion and DMA Transfer Completion Status bits in CHx_IntStatusReg register is cleared to 0 if not masked off.
5. Interrupt Status registers are updated and an interrupt is issued, if not masked.
6. Relevant channel is disabled and CH_DISABLED Status bit is set to 1.

If an error occurs on a source or destination peripheral, the steps that need to access that peripheral are not performed. If multiple channels are enabled, only the channel where the AXI error was detected is disabled.

The FIFO pointers are reset, therefore, the previous FIFO contents become inaccessible and are overwritten once the channel is re-enabled to start a new sequence. There is no support for automatically resuming the transfer from the point where the error occurred, and the full or partial block transfer has to be re-initiated by the software in order to be successfully completed.

If hardware handshaking is enabled for source or destination, the DW_axi_dmac does not signal the end of a transfer. If a request from a peripheral is active when the error occurs (that is, dma_req is high), the channel is disabled without the DMA ever asserting dma_ack and dma_finish. The hardware handshake interface on the peripheral side has to be re-initiated by the CPU upon detection of the error interrupt. The dma_req signal needs to be brought low before the channel is re-enabled and then brought high when the channel has been enabled.

If software handshaking is enabled for SRC / DST, the DW_axi_dmac does not signal the end of a transfer. In practice, this means that if a request from a peripheral is active when the error occurs (all or some of SWHS_Req_Src, SWHS_SglReq_Src, SWHS_Lst_Src, SWHS_Req_Dst, SWHS_SglReq_Dst, SWHS_Lst_Dst bits are high), the channel is disabled without the DMA ever clearing the request bits to 0 in CHx_SWHSSrcReg and / or CHx_SWHSDstReg register.

2.17 Context Sensitive Low Power Option

In order to meet high performance bandwidth requirements of the system, the DMA controller must be operated at high frequency. This increases the overall power consumption of the DMA controller. Thus, it becomes very important to reduce the power consumption at the architecture level. This section describes the architectural low power technique incorporated in the DW_axi_dmac.

The DW_axi_dmac supports the Context Sensitive Low Power option, which is an intelligent low power logic that automatically senses the idle periods in multiple DMA channels, Slave Bus Interface, and AXI Master Interface channels under various scenarios. This cuts-off the clock to these modules, and reduces overall power consumption significantly. The logic is quick enough to provide the clocks back to these modules when any activity is detected without compromising on the performance.

This Context Sensitive Low Power option is enabled through the configuration parameter DMAX_CSLP_EN. You can enable or disable this Context Sensitive Low Power Technique option for DMA channels, Slave Bus interface, and AXI Master interfaces through the following parameters:

- DMA Channels Context Sensitive Low Power Technique - DMAX_CHNL_CSLP_EN
- Slave Bus Interface Context Sensitive Low Power Technique - DMAX_SBIU_CSLP_EN
- AXI Master Interface Context Sensitive Low Power Technique - DMAX_MXIF_CSLP_EN

The following sections describe the Context Sensitive Low Power Techniques used in the DMA Channel, Slave Bus Interface, and AXI Master Interface modules.

2.17.1 DMA Channel Context Sensitive Low Power Technique

The DMA Channel is the major module responsible for controlling the operations such as

- Reading the data from the source
- Temporary storage of the read data in the Channel FIFO
- Writing the data to the destination
- Handling the miscellaneous operation such as interrupt handling, handshake handling
- Channel specific-register read and write operation
- LLI fetch and Write back operation
- Channel Suspend operation
- Channel abort operation and so on

Due to all these operations, this module consumes major area share, and hence it consumes more power than any other module in DW_axi_dmac. Thus, it becomes very important to optimize the power in this module.

The Context Sensitive Low Power option monitors the activity on the DMA Channel by using a DMA Channel Low Power state machine. Whenever the DMA Channel low power condition is detected, DMA Channel low power delay counter starts. The Channel Low Power state machine waits until the DMA Channel delay counter expires, after the counter expires, the DMA Channel enters the Low Power state by gating the dma_core_clk. It remains in this state until any activity is detected on the DMA Channel. In certain scenarios, if an activity is detected when the DMA Channel low power delay counter is running, then DMA Channel delay counter is resets and stops until the low power condition is detected again.

Following are the conditions used to determine whether DMA Channel is idle:

- The DW_axi_dmac comes out of the reset and no channel-specific registers are being written or read.
- The DMA Channel is configured for the DMA operation and the channel is enabled - DMA Channel waits for any activity on the handshaking interface either through the handshaking signals or through the channel-specific software handshaking registers.
- The DMA Channel is configured for the DMA operation and the channel is enabled - DMA Channel source, destination, or LLI state machine has requested for the AXI master interface, but it is waiting for grant from the AXI master interface arbiter (because Other DMA Channels Source, Destination, and LLI state machines also competing for the AXI Master interface grant).
- The DMA Channel is disabled between the successive DMA blocks or Channel disable prior to the transfer completion with or without channel suspend (See “Channel Suspend, Disable, and Abort” on page 89).
- The DMA Channel is suspended based on the Channel Suspend procedures described in the section “Channel Suspend, Disable, and Abort” on page 89
- The DMA Channel is aborted based on the Abnormal Channel Abort procedure described in the section “Channel Suspend, Disable, and Abort” on page 89)

- The entire DMAC is disabled through DMAC_CFGREG.DMAC_EN bit.
- After the DMAC software reset DMAC_RESETREG.DMAC_RST is asserted.

Based on the previously described idle conditions the DMA Channel low power counter starts. If the DMA Channel low power delay counter expires, the corresponding DMA Channel enters the low power state.

Following are conditions under which the DMA Channel exits the low power state:

- Any read / write transaction occurs to any of the DMA Channel-specific registers (like CHx_SAR, CHx_DAR, CHx_BLOCK_TS, CHx_CTL, CHx_CFG and so on).
- An activity on the handshaking interface is detected either through the handshaking signals or through the channel specific software handshaking registers.
- The DMA Channel source, destination, or LLI state machine has been granted with access to the AXI Master interface.
- Clearing of Channel-specific interrupts by writing into the CHx_INTCLEARREG register.
- On the rising or falling edge of the Channel enable DMAC_CHENREG.CHx_EN or DMAC_CFGREG.DMAC_EN.
- On the rising or falling edge of the Channel suspend or Channel abort.
- On assertion of DMAC software reset

The Context Sensitive Low Power option can be enabled through the configuration parameter DMAX_CHNL_CSLP_EN. The DMA Channel low power delay counter load value can be configured through the DMAC_LOWPOWER_CFGREG.CHNL_LPDLY register field. The parameter DMAX_GLCH_LPDLY is used to configure the default load value of the DMAC_LOWPOWER_CFGREG.CHNL_LPDLY field.

2.17.2 Slave Bus Interface Context Sensitive Low Power Technique

The Slave Bus Interface module is used to access the internal registers of DW_axi_dmac by an external AHB Master interface. The Context Sensitive Low Power option is implemented to optimize the power consumption in the Slave Bus Interface module.

The Context Sensitive Low Power option monitors the activity on the Slave Bus Interface and based on based on the SBIU state machine. Whenever the Slave Bus Interface low power condition is detected, (that is when no AHB read or write transfers are in progress and SBIU state machine is in IDLE) SBIU low power delay counter is started. When the SBIU low power delay counter expires the Slave Bus Interface module enters the Low Power state by gating the hclk or dma_core_clk (based on the Clock Mode configuration). It remains in this state until an active read or write transaction is detected on the Slave Bus Interface. In certain scenarios, if an active transaction is detected when SBIU low power delay counter running, then SBIU delay counter is reset and stopped until the low power condition is detected again.

If a read or write transaction is detected, when Slave Bus Interface module is in low power state, then the context sensitive low power logic responds immediately to exit the low power state by un-gating the hclk or dma_core_clk. Since the low power exit operation occurs within a clock cycle, following are the major advantage of this architecture:

- No additional registers required to buffer the transaction control information
- No wait cycle insertion required

Condition when the Slave Bus Interface module enters the Low Power state:

- No active AHB read or write transaction is in progress and SBIU low power delay counter has been expired

Condition when the Slave Bus Interface module exits the Low Power state:

- Any AHB read or write transaction is initiated

The Context Sensitive Low Power option can be enabled through the configuration parameter DMAX_SBIU_CSLP_EN. The SBIU low power delay counter load value can be configured through the DMAC_LOWPOWER_CFGREG.SBIU_LPDLY register field. The parameter DMAX_SBIU_LPDLY is used to configure the default load value of the DMAC_LOWPOWER_CFGREG.SBIU_LPDLY field.

2.17.3 AXI Master Interface Context Sensitive Low Power Technique

The AXI Master Interface module implements the AXI bus to transfer the data between the memories and peripherals. Since the AXI bus operates at the high frequency to address the bandwidth requirement, which leads to the increased power consumption. The Context Sensitive Low Power option is implemented in AXI Master interface to optimize the power consumption in the AXI Master Interface module.

The Context Sensitive Low Power option is implemented for each of the following AXI Channels separately:

- Write address Channel
- Write data Channel
- Write response Channel
- Read address Channel
- Read data Channel

Whenever the AXI Channel low power condition is detected, the respective AXI Channel low power delay counter starts. The AXI Low Power logic waits until the AXI Channel delay counter expires, after the counter expires the AXI Channel enters the Low Power state by gating the aclk_mi and / or dma_core_clk (based on the Clock Mode configuration). It remains in this state until any activity is detected on the respective AXI Channel. In certain scenarios, if an activity is detected when the AXI Channel low power delay counter is running, then AXI Channel delay counter is reset and stopped until the low power condition is detected again.

Following is the condition used to determine whether AXI Channel is Idle:

- AXI Channel FIFO is empty and no data to perform the AXI transfers
- Clock Mode Synchronous: The FIFO flag is empty in the dma_core_clk domain
- Clock Mode Asynchronous: The FIFO flags from both the dma_core_clk and aclk_mi domain are empty

Following are the conditions under which the DMA Channel exits the low power state:

- The data is pushed into the AXI Channel FIFO:
  - **AW Channel, W Channel, AR Channel:** The low power logic is quick enough to exit the low power state as soon as it detects the push from the DMA Channel. This logic ensures that additional buffering is not required to avoid the loss of the push due to the delay in exiting the

low power state. As mentioned earlier, low power logic is implemented separately for each of these AXI channels.

- **B Channel, R Channel:** The low power logic is quick enough to exit the low power state as soon as it detects the push from the AXI Channel based bvalid / rvalid logic. This logic makes sure that additional buffering is not required to avoid the loss of the push due to the delay in exiting the low power state. As mentioned earlier, low power logic is implemented separately for each of these AXI channels.

- DMAC software reset is asserted

The Context Sensitive Low Power option can be enabled through the configuration parameter DMAX_MXIF_CSLP_EN. The AXI Channel low power delay counter load value can be configured through the DMAC_LOWPOWER_CFGREG.MXIF_LPDLY register field. The parameter DMAX_MXIF_LPDLY is used to configure the default load value of the DMAC_LOWPOWER_CFGREG.MXIF_LPDLY field.

2.17.4 Global Context Sensitive Low Power Technique

The Global Context Sensitive Low Power technique implements the low power option based on the DMA Channel, Slave Bus Interface, and AXI Master Interface idle condition. If all the mentioned modules indicate that the DW_axi_dmac is in Idle, then a Global low power delay counter is started. The Global Context Sensitive Low power logic waits until the low power delay counter expires, after the counter expires the DW_axi_dmac enters the Low Power state by gating the clocks to all modules like dma_core_clk, hclk, and / or aclk_mi (based on the Clock Mode configuration). It remains in low power state until any activity is detected on any of one of the module mentioned previously.

In certain scenarios, if an activity is detected when the Global low power delay counter is running, then Global delay counter is reset and stopped until the low power condition is again detected.

The Context Sensitive Low Power option can be enabled through the configuration parameter DMAX_CSLP_EN. The Global low power delay counter load value can be configured through the DMAC_LOWPOWER_CFGREG.GLBL_LPDLY register field. The parameter DMAX_GLCH_LPDLY is used to configure the default load value of the DMAC_LOWPOWER_CFGREG.GLBL_LPDLY field.

3 Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the complete configuration state of the core.** 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 core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

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

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
Top Level Parameters	
AXI DMAC ID	A 64-bit value that is hardwired and read back by a read to the DW_axi_dmac ID Register (DMAC_IDReg). **Values:** 0x0, ..., 0xffffffffffffffff **Default Value:** 0x0 **Enabled:** Always **Parameter Name:** DMAX_ID_NUM
Number of DMA Channels	Creates the specified number of DW_axi_dmac channels, each of which is unidirectional and transfers data from the channel source to the channel destination. The channel source and destination AXI layer, system address, and handshaking interface are under software control. **Values:** 1, ..., 32 **Default Value:** 1 **Enabled:** Always **Parameter Name:** DMAX_NUM_CHANNELS
Number of Handshaking Interfaces	Creates the specified number of hardware handshaking interfaces. DW_axi_dmac can be programmed to assign a handshaking interface for each channel source and destination. If 0 is selected, then no hardware handshaking signals are present on the I/O. **Values:** 0, ..., 64 **Default Value:** 2 **Enabled:** Always **Parameter Name:** DMAX_NUM_HS_IF

Table 3-1 Top Level Parameters (Continued)

Label	Description
Interrupt Pins to Appear as Outputs?	Selects which interrupt-related signals appear as outputs on the design. ■ COMBINED_ONLY: Only "intr" output exist. Bitwise OR of all bits of DMAC_IntStatusReg register is driven onto "intr" output. ■ CHANNEL_AND_COMMONREG: "intr_ch" and "intr_cmnreg" outputs exist. Bitwise OR of all the corresponding channel bits of DMAC_IntStatusReg register is driven onto the respective "intr_ch" output. Bitwise OR of all the bits of DMAC_CommonReg_IntStatusReg register is driven onto the respective "intr_cmnreg" output. ■ ALL_INTERRUPT_OUTPUTS: "intr", "intr_ch" and "intr_cmnreg" outputs exist. Bitwise OR of all bits of DMAC_IntStatusReg register is driven onto "intr" output. Bitwise OR of all the corresponding channel bits of DMAC_IntStatusReg register is driven onto the respective "intr_ch" output. Bitwise OR of all the bits of DMAC_CommonReg_IntStatusReg register is driven onto the respective "intr_cmnreg" output. **Values:** ■ COMBINED ONLY (0) ■ CHANNEL AND COMMONREG (1) ■ ALL INTERRUPTS OUTPUTS (2) **Default Value:** COMBINED ONLY **Enabled:** Always **Parameter Name:** DMAX_INTR_IO_TYPE
Include Status Indication Output on Slave Bus Interface?	By default, this option creates a slave interface status indication (Busy/Idle) signals on the I/O, which is synchronous to the slave interface clock. When you set this option to False (0), the signal is not included on the I/O. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_SLVIF_STATUS_OP_EN
Include DMAC Internal Status Indication Output?	By default, this option creates a DMAC internal status indication (Busy/Idle) signal on the I/O, which is synchronous to dmac_core_clock. When you set this option to False (0), the signal is not included on the I/O. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_CORE_STATUS_OP_EN

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include DMAC Hold Request Input & Hold Acknowledgement Output?	When set to Yes (1), this option creates a DMAC Hold Request Input signal (dmac_hold_req) and DMAC Hold Acknowledgement Output signal (dmac_hol_ack) on the I/O, which are synchronous to dmac_core_clock. When you set this option to False (0), this signal is not included on the I/O. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_HOLD_IO_EN
Enable Unaligned Transfer Support?	When set to Yes (1), this parameter enables the unaligned transfer support on CHx_SAR and CHx_DAR. This reduces the software overhead of converting the unaligned address to aligned address. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_UNALIGNED_XFER_EN
Enable Multi-Arbiter Feature?	When set to Yes (1), Multi-arbiter feature is enabled. This feature implements the Multiple Stage Arbiter Architecture (Multi-Arbiter) to arbitrate the requests from source, destination, and LLI state machine to access an AXI Master interface. Enabling this feature helps to improve the QoR for the implemented design. **Values:** ■ No (0) ■ Yes (1) **Default Value:** (DMAX_NUM_CHANNELS>8) ? 1 : 0 **Enabled:** DMAX_NUM_CHANNELS>8 **Parameter Name:** DMAX_MULT_ARB_EN
Include Channel Abort Feature?	When set to Yes (1), enables logic to support Channel Abort feature in DW_axi_dmac. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_CH_ABORT_EN

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include Transfer Completion Indication Signal on Master Interface?	When set to Yes (1), enables the additional handshaking signal last_write_mi on all AXI master interfaces. The last_write_mi signal is asserted on last data phase of every destination block transfer and remains asserted until the last data phase completes. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_ENABLE_LAST_WRITE
Include Debug Ports?	When set to Yes (1), enables debug_* ports in DW_axi_dmac. **Values:** ■ No (0) ■ Yes (1) **Default Value:** DMAX_CSLP_EN == 1 ? 1 : 0 **Enabled:** Always **Parameter Name:** DMAX_DEBUG_PORTS_EN

3.2 Master Interface Configuration Parameters

Table 3-2 Master Interface Configuration Parameters

Label	Description
Master Interface Configuration	
DW_axi_dmac Master Interface Protocol	The protocol used for the AXI Master Interface. **Values:** ■ AXI3 (0) ■ AXI4 (1) **Default Value:** AXI3 **Enabled:** Always **Parameter Name:** DMAX_MSTIF_MODE
Enable QoS feature?	When set to Yes (1), enables the QoS signals in the AXI4 interface. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_MSTIF_MODE==1 **Parameter Name:** DMAX_HAS_QOS
Number of AXI Master Interfaces	Creates the specified number of AXI master interfaces. A channel source or destination device can be programmed to be on any of the configured AXI layers attached to the AXI master interface. This setting determines if AXI master 2 interface signal set is present on the I/O. AXI master interface signals are always present. **Values:** 1, 2 **Default Value:** 1 **Enabled:** Always **Parameter Name:** DMAX_NUM_MASTER_IF
Master Interface i Outstanding Request Limit (for i = 1; i <= DMAX_NUM_MASTER_IF)	The maximum number of active write requests that can be generated by an AXI Master interface (i) without sending the respective write data. This parameter is used to select the depth of a FIFO, which tracks the write data transferred on AXI Master interface for the active AXI write requests. **Note:** AXI outstanding write/read requests are controlled through the programming registers CHx_CFG.SRC_OSR_LMT and CHx_CFG.DST_OSR_LMT. **Values:** 16, 32, 48, 64, 80, 96, 112, 128 **Default Value:** 16 **Enabled:** Always **Parameter Name:** DMAX_MSTIF(i)_OSR_LMT

Table 3-2 Master Interface Configuration Parameters (Continued)

Label	Description
Statically Configure Endian Scheme of Master Interfaces?	The endian scheme of the DW_axi_dmac master interfaces can be configured statically through coreConsultant or dynamically through pins on the I/O. ■ For the static case, there is a single coreConsultant parameter that controls the endianness of all AXI master interfaces. ■ For the dynamic case, there is an individual pin for each of the AXI master interfaces. When set to Yes (1), the endianness is configured based on the value of DMAX_ENDIAN_FORMAT_MSTIF; otherwise it is configured based on the value of input pin, dmac_endian_format_mstif_m(i). **Values:** ■ No (0) ■ Yes (1) **Default Value:** Yes **Enabled:** Always **Parameter Name:** DMAX_STATIC_ENDIAN_SELECT_MSTIF
Include Little Endian Scheme Selection Pin for LLI Access on AXI Master Interfaces?	When set to Yes (1), enables additional inputs are enabled to control the endian scheme used for LLI access. An individual pin is added for each AXI Master interface. LLI access on each AXI Master interface can be independently configured to support Big Endian scheme (BE-8) based on the endian scheme selected for that particular master interface or Little Endian scheme irrespective of the endian scheme selected for that particular master interface. ■ 0: Endian scheme used for LLI access is the same as that used for data access for M1/M2 interfaces. ■ 1: Endian scheme used for LLI access is the same as that used for data access for M1/M2 interfaces. or ■ Little Endian method is used for LLI access irrespective of the endian scheme used for data access depending on the value of dmac_le_select_lli_mstif_m(i) input. **Values:** 0, 1 **Default Value:** 0 **Enabled:** !((DMAX_STATIC_ENDIAN_SELECT_MSTIF == 1) && (DMAX_ENDIAN_FORMAT_MSTIF ==0)) **Parameter Name:** DMAX_LLI_ENDIAN_SELECTION_PIN_EN

Table 3-2 Master Interface Configuration Parameters (Continued)

Label	Description
Master Interface Endian Format?	DMAX_ENDIAN_FORMAT_MSTIF values for different AXI master interface endian formats are as follows. ■ 0: Little Endian ■ 1: Big Endian BE-8 **Values:** ■ Little Endian (0) ■ Big Endian BE-8 (1) **Default Value:** Little Endian **Enabled:** DMAX_STATIC_ENDIAN_SELECT_MSTIF == 1 **Parameter Name:** DMAX_ENDIAN_FORMAT_MSTIF
Master Interface Adddress Bus Width	AXI Master 1 interface address bus width. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** DMAX_M_ADDR_WIDTH
Master Interface Data Bus Width	AXI Master interface data bus width. **Values:** 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** DMAX_M_DATA_WIDTH
Master Interface ID Bus Width	AXI master interface ID bus width (common for awid_m(i), arid_m(i), wid_m(i), bid_m(i) and rid_m(i)). **Note**: The Minimum value of DMAC_M_ID_WIDTH is equal to log2(DMAX_NUM_CHANNELS) if multi-block transfer type is hardcoded to non-linked-list-based schemes, otherwise [log2(DMAX_NUM_CHANNELS)] + 1. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 **Default Value:** 6 **Enabled:** Always **Parameter Name:** DMAX_M_ID_WIDTH
Master Interface Burst Length Width	AXI master interface Burst Length (arlen(i)/awlen(i)) width. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** Always **Parameter Name:** DMAX_M_BURSTLEN_WIDTH

3.3 Slave Interface Configuration Parameters

Table 3-3 Slave Interface Configuration Parameters

Label	Description
Slave Interface Configuration	
DW_axi_dmac Slave Interface Protocol	The protocol is used for Slave Bus Interface. *Only AHB is supported in this version.* **Values:** ■ AHB (0) ■ AXI4-Lite (1) ■ APB3 (2) **Default Value:** AHB **Enabled:** 0 **Parameter Name:** DMAX_SLVIF_MODE
Synchronize Interrupt Outputs to Slave Interface Clock?	By default, interrupt output is synchronous to dmac_core_clock. When this parameter is set to Yes (1), the interrupt output pins are synchronous to slave Interface clock. In this case, additional synchronizers are added inside DW_axi_dmac. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_SLVIF_CLOCK_MODE!=0 **Parameter Name:** DMAX_INTR_SYNC2SLVCLK
Slave Interface Data Bus Width	Specifies the data bus width for the AHB/APB3/AXI4-Lite slave interface. **Values:** 32, 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** DMAX_S_DATA_WIDTH

3.4 Clocking Parameters

Table 3-4 Clocking Parameters

Label	Description
Synchronization options	
DW_axi_dmac Slave Interface Clocking Mode	Selects the relationship between the slave interface clock (AHB/AXI4-Lite/APB3) and the Core clock. ■ 0: Slave interface clock is synchronous to the core clock. ■ 1: Slave interface clock is asynchronous to the core clock. **Values:** ■ Synchronous (0) ■ Asynchronous (1) **Default Value:** Synchronous **Enabled:** Always **Parameter Name:** DMAX_SLVIF_CLOCK_MODE
Slave to Core Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Slave clock domain to the DW_axi_dmac Core Clock domain. ■ 1: Two-stage synchronization. First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_SLVIF_CLOCK_MODE==1 **Parameter Name:** DMAX_S_2_C_SYNC_DEPTH
Core to Slave Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Core Clock Domain to the Slave clock domain. ■ 1: Two-stage synchronization. First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_SLVIF_CLOCK_MODE==1 **Parameter Name:** DMAX_C_2_S_SYNC_DEPTH

Table 3-4 Clocking Parameters (Continued)

Label	Description
DW_axi_dmac Master i Interface Clocking Mode (for i = 1; i <= DMAX_NUM_MASTER_IF)	Selects the relationship between the Master i Interface clock (aclk_m(i)) and the Core clock. ■ 0: Master i interface clock is synchronous to the core clock. ■ 1: Master i interface clock is asynchronous to the core clock. **Values:** ■ Synchronous (0) ■ Asynchronous (1) **Default Value:** Synchronous **Enabled:** Always **Parameter Name:** DMAX_MSTIF(i)_CLOCK_MODE
Master i to Core Synchronization Depth (for i = 1; i <= DMAX_NUM_MASTER_IF)	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Master i clock domain to the DW_axi_dmac Core Clock domain. ■ 1: Two-stage synchronization: First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_MSTIF(i)_CLOCK_MODE == 1 **Parameter Name:** DMAX_M(i)_2_C_SYNC_DEPTH
Core to Master i Synchronization Depth (for i = 1; i <= DMAX_NUM_MASTER_IF)	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Core Clock domain to the Master i clock domain. ■ 1: Two-stage synchronization: First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge. **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_MSTIF(i)_CLOCK_MODE == 1 **Parameter Name:** DMAX_C_2_M(i)_SYNC_DEPTH

Table 3-4 Clocking Parameters (Continued)

Label	Description
Synchronizer FIFO Depths of Master Interface	
Master Interface Read/Write Address Channel FIFO Depth	AXI master interface read/write address channel FIFO depth. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 4, 8 **Default Value:** 4 **Enabled:** DMAX_MSTIF1_CLOCK_MODE == 1 || DMAX_MSTIF2_CLOCK_MODE == 1 **Parameter Name:** DMAX_M_ADDR_FIFO_DEPTH
Master Interface Read/Write Data Channel FIFO Depth	AXI master interface read/write data channel FIFO depth. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 4, 8 **Default Value:** 4 **Enabled:** DMAX_MSTIF1_CLOCK_MODE == 1 || DMAX_MSTIF2_CLOCK_MODE == 1 **Parameter Name:** DMAX_M_DATA_FIFO_DEPTH
Master Interface Write Response Channel FIFO Depth	AXI master interface write response channel FIFO depth. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 4, 8 **Default Value:** 4 **Enabled:** DMAX_MSTIF1_CLOCK_MODE == 1 || DMAX_MSTIF2_CLOCK_MODE == 1 **Parameter Name:** DMAX_M_BRESP_FIFO_DEPTH
Asynchronous Handshake Synchronization Options	
Include Asynchronous DMA Handshake Support?	When set to Yes (1), allows you to include or exclude asynchronous DMA handshake support. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_NUM_HS_IF > 0 **Parameter Name:** DMAX_ASYNC_HS_EN

Table 3-4 Clocking Parameters (Continued)

Label	Description
All DMA Handshake Interface has Same Asynchronous Clock?	When set to Yes (1), configures whether all DMA handshake interface has the same asynchronous clock. **Values:** ■ No (0) ■ Yes (1) **Default Value:** Yes **Enabled:** DMAX_ASYNC_HS_EN == 1 **Parameter Name:** DMAX_HS_SAME_ASYNC_CLK
Handshake to Core Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac handshake interface clock domain to the DW_axi_dmac Core Clock Domain. ■ 1: Two-stage synchronization: First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_ASYNC_HS_EN==1 **Parameter Name:** DMAX_HS_2_C_SYNC_DEPTH
Core to Handshake Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Core Clock domain to handshake interface clock domain. ■ 1: Two-stage synchronization: First stage - negative edge; Second stage - positive edge. ■ 2: Two-stage synchronization, both stages positive edge. ■ 3: Three-stage synchronization, all stages positive edge. ■ 4: Four-stage synchronization, all stages positive edge **Values:** 1, 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_ASYNC_HS_EN==1 **Parameter Name:** DMAX_C_2_HS_SYNC_DEPTH

Table 3-4 Clocking Parameters (Continued)

Label	Description
DMAX_HS(y)_ASYNC_CLK	
DMA Handshake interface y has asynchronous clock? (for y = 1; y <= DMAX_NUM_HS_IF)	The DMA handshake interface y has asynchronous clock. **Values:** ■ false (0) ■ true (1) **Default Value:** (DMAX_HS_SAME_ASYNC_CLK==1) ? 1 : 0 **Enabled:** (DMAX_ASYNC_HS_EN == 1) && (DMAX_HS_SAME_ASYNC_CLK == 0) && (DMAX_NUM_HS_IF >= y) **Parameter Name:** DMAX_HS(y)_ASYNC_CLK

3.5 Low Power Configuration Parameters

Table 3-5 Low Power Configuration Parameters

Label	Description
Low Power Configuration	
Include Context Sensitive Low Power Feature?	When set to Yes (1), allows you to include Context Sensitive Low Power feature. If enabled, it includes Global Context Sensitive Low Power feature and allows other Context Sensitive Low Power feature to be implemented optionally. Enabling this feature saves significant amount of power, with minimal area consumption. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_CSLP_EN
Include Context Sensitive Low Power Feature in DMA Channels?	When set to Yes (1), includes the logic that implements Context Sensitive Low Power feature in DMA Channels. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_CSLP_EN == 1 **Parameter Name:** DMAX_CHNL_CSLP_EN
Include Context Sensitive Low Power Feature in Slave Bus Interface?	When set to Yes (1), includes the logic that implements Context Sensitive Low Power feature in Slave Bus Interface. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_CSLP_EN == 1 **Parameter Name:** DMAX_SBIU_CSLP_EN
Include Context Sensitive Low Power Feature in AXI Master Interface?	Include or exclude logic that implements Context Sensitive Low Power feature in Master Bus Interface. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_CSLP_EN == 1 **Parameter Name:** DMAX_MXIF_CSLP_EN

Table 3-5 Low Power Configuration Parameters (Continued)

Label	Description
Width of Global and DMA Channel Low Power Delay Counter	Defines the width of the Global and DMA Channel Low Power Delay Counter. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** DMAX_CSLP_EN == 1 || DMAX_CHNL_CSLP_EN == 1 **Parameter Name:** DMAX_GLCH_LPDLY_WIDTH
Width of SBIU Low Power Delay Counter	Defines the width of the SBIU Low Power Delay Counter. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** DMAX_SBIU_CSLP_EN == 1 **Parameter Name:** DMAX_SBIU_LPDLY_WIDTH
Width of AXI Master Interface Low Power Delay Counter	Defines the width of the AXI Master Interface Low Power Delay Counter. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** DMAX_MXIF_CSLP_EN == 1 **Parameter Name:** DMAX_MXIF_LPDLY_WIDTH
Default load value of Global and DMA Channel Low Power Delay Counter	Defines the default load value of the Global and DMA Channel Low Power Delay Counter. **Values:** 4, ..., 255 **Default Value:** 4 **Enabled:** DMAX_CSLP_EN == 1 || DMAX_CHNL_CSLP_EN == 1 **Parameter Name:** DMAX_GLCH_LPDLY
Default load value of SBIU Low Power Delay Counter	Defines the default load value of the SBIU Low Power Delay Counter. **Values:** 4, ..., 255 **Default Value:** 4 **Enabled:** DMAX_SBIU_CSLP_EN == 1 **Parameter Name:** DMAX_SBIU_LPDLY
Default load value of AXI Master Interface Low Power Delay Counter	Defines the default load value of the AXI Master Interface Low Power Delay Counter. **Values:** 4, ..., 255 **Default Value:** 4 **Enabled:** DMAX_MXIF_CSLP_EN == 1 **Parameter Name:** DMAX_MXIF_LPDLY

3.6 Channel x Configuration Parameters

Table 3-6 Channel x Configuration Parameters

Label	Description
Channel x Configuration	
Channel x FIFO Depth (for x = 1; x <= DMAX_NUM_CHANNELS)	Channel x FIFO depth. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 4, 8, 16, 32, 64, 128, 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** DMAX_CH(x)_FIFO_DEPTH
Maximum Value of Burst Transaction Size (for x = 1; x <= DMAX_NUM_CHANNELS)	Maximum value of burst transaction size that can be programmed for channel x (CH(x)_CTL.SRC_MSIZE and CH(x)_CTL.DST_MSIZE). Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 1, 4, 8, 16, 32, 64, 128, 256, 512, 1024 **Default Value:** 8 **Enabled:** Always **Parameter Name:** DMAX_CH(x)_MAX_MSIZE
Maximum Value of Block Size in Source Transfer Width (for x = 1; x <= DMAX_NUM_CHANNELS)	The description of this parameter is dependent on what is assigned as the flow controller. ■ If DW_axi_dmac is the flow controller: Maximum block size, in multiples of source transfer width (CH(x)_BLOCK_TS.BLOCK_TS), that can be programmed for channel x. A programmed value greater than this results in inconsistent behavior. ■ If source/destination peripheral assigned as flow controller: In this case, the blocks can be greater than DMAX_CH(x)_MAX_BLOCK_TS in size, but the logic that keeps track of the size of a block saturates at DMAX_CH(x)_MAX_BLOCK_TS. This does not result in erroneous behavior, but a read back by software of the block size is incorrect when the block size exceeds the saturated value. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287, 1048575, 2097151, 4194303 **Default Value:** 31 **Enabled:** Always **Parameter Name:** DMAX_CH(x)_MAX_BLOCK_TS

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Maximum Value of AMBA Burst Length (for x = 1; x <= DMAX_NUM_CHANNELS)	Maximum AMBA Burst Length for Channel x. Setting appropriate value based on the system requirement allows some logic optimization of the implementation by reducing the internal FIFO depth requirement. **Dependencies**: ■ DMAX_CH(x)_MAX_AMBA_BURST_LENGTH <= DMAX_CH(x)_MAX_BLOCK_TS ■ DMAX_CH(x)_MAX_AMBA_BURST_LENGTH <= DMAX_CH(x)_MAX_MSIZE **Values:** 1, 4, 8, 16, 32, 64, 128, 256 **Default Value:** 8 **Enabled:** Always **Parameter Name:** DMAX_CH(x)_MAX_AMBA_BURST_LENGTH
Include Logic to Enable Channel Locking on Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	When set to Yes (1), includes logic to enable channel locking on channel x. When set to 1, the software can program the DW_axi_dmac to lock the arbitration for the master bus interface over the DMA transfer, block transfer, or transaction. When set to No (0), this option allows some logic optimization of the implementation. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_CH(x)_LOCK_EN
Hardcode Channel x's transfer Type and Flow Control Device to Allow for Logic Optimization (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcodes the transfer type and flow control peripheral for the channel x. If NO_HARDCODE is selected, then the transfer type and flow control device is not hardcoded, and software selects the transfer type and flow control device for a DMA transfer. Hardcoding the transfer type and flow control device allows some logic optimization of the implementation. **TT_FC Flow Controller Transfer Type** 0x0 DMAC Memory to Memory 0x1 DMAC Memory to Peripheral 0x2 DMAC Peripheral to Memory 0x3 DMAC Peripheral to Peripheral 0x4 Source Peripheral to Memory 0x5 Source Peripheral to Peripheral 0x6 Destination Memory to Peripheral 0x7 Destination Peripheral to Peripheral 0x8 No Hardcode No Hardcode **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8 **Default Value:** 0x8 **Enabled:** Always **Parameter Name:** DMAX_CH(x)_TT_FC

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Hardcode the Master Interface Attached to the Source of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI master interface attached to the source of channel x. If this is not hardcoded, software can program the source of channel x to be attached to any of the configured layers. Hardcoding this value allows some logic optimization of the implementation. **Values:** ■ Master 1 (0x0) ■ Master 2 (0x1) ■ No Hardcode (0x2) **Default Value:** DMAX_NUM_MASTER_IF == 1 ? 0 : 2 **Enabled:** DMAX_NUM_MASTER_IF > 1 **Parameter Name:** DMAX_CH(x)_SMS
Hardcode the Master Interface Attached to the Destination of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI master interface attached to the channel x destination. If this is not hardcoded, then software can program the destination of channel x to be attached to any of the configured layers. Hardcoding this value allows some logic optimization of the implementation. **Values:** ■ Master 1 (0x0) ■ Master 2 (0x1) ■ No Hardcode (0x2) **Default Value:** DMAX_NUM_MASTER_IF == 1 ? 0 : 2 **Enabled:** DMAX_NUM_MASTER_IF > 1 **Parameter Name:** DMAX_CH(x)_DMS
Hardcode Channel x's Source Transfer Width (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the source transfer width for transfers from the source of channel x. If this is not hardcoded, then software can program the source transfer width. Hardcoding the source transfer width allows some logic optimization of the implementation. **Values:** ■ No Hardcode (0) ■ Byte (8) ■ Half Word (16) ■ Word (32) ■ Two Word (64) ■ Four Word (128) ■ Eight Word (256) ■ Sixteen Word (512) **Default Value:** Word **Enabled:** Always **Parameter Name:** DMAX_CH(x)_STW

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Hardcode Channel x's Destination Transfer Width (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the destination transfer width for transfers to the destination of channel x. If this is not hardcoded, then software can program the destination transfer width. Hardcoding the destination transfer width allows some logic optimization of the implementation. **Values:** ■ No Hardcode (0) ■ Byte (8) ■ Half Word (16) ■ Word (32) ■ Two Word (64) ■ Four Word (128) ■ Eight Word (256) ■ Sixteen Word (512) **Default Value:** Word **Enabled:** Always **Parameter Name:** DMAX_CH(x)_DTW
Include Logic to Enable Multiblock DMA Transfers on Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	Includes or excludes logic to enable multi-block DMA transfers on channel x. If this option is set to 0, then hardware hardwires channel x to perform only single block transfers. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** DMAX_CH(x)_MULTI_BLK_EN

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Hardcode Multi-block Transfer Type? (for x = 1; x <= DMAX_NUM_CHANNELS)	This parameter allows to hardcode the type of multi-block transfers DW_axi_dmac can perform on channel x. This results in some logic optimization of the implementation. **MULTI_BLK_TYPE Source Multi Block Type Destination Multi Block Type** 0x0 No Hardcode No Hardcode 0x1 Contiguous Auto Reloading 0x2 Contiguous Shadow Register 0x3 Auto Reloading Contiguous 0x4 Auto Reloading Auto Reloading 0x5 Auto Reloading Shadow Register 0x6 Shadow Register Contiguous 0x7 Shadow Register Auto Reloading 0x8 Shadow Register Shadow Register 0x9 Contiguous Linked List 0xa Auto Reloading Linked List 0xb Linked List Contiguous 0xc Linked List Auto Reloading 0xd Linked List Linked List **Values:** 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd **Default Value:** 0x0 **Enabled:** DMAX_CH(x)_MULTI_BLK_EN==1 **Parameter Name:** DMAX_CH(x)_MULTI_BLK_TYPE
Enable LLI Prefetching on Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	Set this parameter to 1 to enable the LLI prefetching logic on channel x. If this parameter is enabled, DW_axi_dmac tries to fetch one LLI in advance (while the data transfer corresponding to the previous LLI is in progress) and store internally and thus increases bus utilization. Enabling this parameter doesn't guarantee that LLI will be always pre-fetched. This parameter needs to be enabled only if linked-list-based multi-block transfer is used. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Disabled if DMAX_CH(x)_MULTI_BLK_EN = 0 or If DMAX_CH(x)_MULTI_BLK_TYPE is hardcoded to a value that does not use LLI **Parameter Name:** DMAX_CH(x)_LLI_PREFETCH_EN

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Include Shadow Registers on Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	Set this parameter to 1 to include shadow registers on channel x. This parameter needs to be enabled only if shadow register based multi-block transfer is used. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Not enabled if DMAX_CH(x)_MULTI_BLK_EN = 0 and DMAX_CH(x)_MULTI_BLK_TYPE is hardcoded to a value that does not use shadow **Parameter Name:** DMAX_CH(x)_SHADOW_REG_EN
Hardcode the Master Interface Attached to the LLP peripheral of channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI master interface attached to the peripheral that stores the LLI information (Linked List Item) for channel x. If this is not hardcoded, then software can program the peripheral that stores the LLI information of channel x to be attached to any of the configured layers. Hardcoding this value allows some logic optimization of the implementation. LLI access always uses the burst size (arsize/awsize) that is same as the data bus width. LLI access cannot be changed or programmed to anything other than this value. Burst length (awlen/arlen) is chosen based on the data bus width so that the access does not cross one complete LLI structure of 64 bytes. DW_axi_dmac fetches the entire LLI (40 bytes) in one AXI burst if the burst length is not limited by other settings. **Values:** ■ Master 1 (0x0) ■ Master 2 (0x1) ■ No Hardcode (0x2) **Default Value:** DMAX_NUM_MASTER_IF == 1 ? 0 : 2 **Enabled:** Enabled only when channel has LLI multi-block transfers enabled **Parameter Name:** DMAX_CH(x)_LMS
Fetch Status From Source of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Include or exclude logic to fetch status from source peripheral of channel x pointed to by the content of CH(x)_SSTATAR register and store it in CHx_SSTAT register if CH(x)_CTL.SRC_STAT_EN bit is set to 1. This value is written back to the CH(x)_SSTAT location of linked list at end of each block transfer if DMAX_CH(x)_LLI_WB_EN is set to 1 and if linked-list-based multiblock transfer is used by either source or destination peripheral. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** DMAX_CH(x)_SRC_STAT_EN

Table 3-6 Channel x Configuration Parameters (Continued)

Label	Description
Fetch Status From Destination of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Include or exclude logic to fetch status from destination peripheral of channel x pointed to by the content of CHx_DSTATAR register and store it in CH(x)_DSTAT register if CH(x)_CTL.DST_STAT_EN bit is set to 1. This value is written back to the CH(x)_DSTAT location of linked list at end of each block transfer if DMAX_CH(x)_LLI_WB_EN is set to 1 and if linked-list-based multiblock transfer is used by either source or destination peripheral. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** DMAX_CH(x)_DST_STAT_EN
Include Logic to Enable Register Write-back After Each Block Transfer? (for x = 1; x <= DMAX_NUM_CHANNELS)	Includes or excludes logic to enable write-back of the CH(x)_CTL, CH(x)_LLP_STATUS, CH(x)_SSTAT and CHx_DSTAT registers at the end of every block transfer. Write back happens only if linked-list-based multi-block transfer is used by either source or destination peripheral. Setting this parameter to 0 allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DMAX_CH(x)_MULTI_BLK_TYPE == 0 or DMAX_CH(x)_MULTI_BLK_TYPE >8 **Parameter Name:** DMAX_CH(x)_LLI_WB_EN

4
Signal Descriptions

This chapter details all possible I/O signals in the core. 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 core. It is for reference purposes only.

When you configure the core 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 core 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 in-active state (opposite of the active state).

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

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

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

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

The I/O signals are grouped as follows:

- Core Clock and Reset on page 125
- Test Interface on page 126
- AHB Signals on page 127
- Master Interface on page 130
- Hardware Handshaking Signals on page 141
- Debug Interface Signals on page 144
- Interrupt Interface Signals on page 153
- Low Power Interface Signals on page 154

4.1 Core Clock and Reset Signals

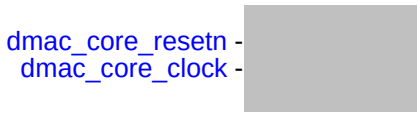


Table 4-1 Core Clock and Reset Signals

Port Name	I/O	Description
dmac_core_resetn	I	DW_axi_dmac core reset. Active low input that asynchronously resets the DW_axi_dmac. The reset must be synchronously de-asserted after the rising edge of dmac_core_clock. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_core_clock	I	DW_axi_dmac core clock. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.2 Test Interface Signals

scan_mode -

Table 4-2 Test Interface Signals

Port Name	I/O	Description
scan_mode	I	Scan Mode. This signal should be asserted during scan testing and should be de-asserted (tied to logic 0) at all the other times. **Exists:** (DMAX_CSLP_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** High.

4.3 AHB Signals

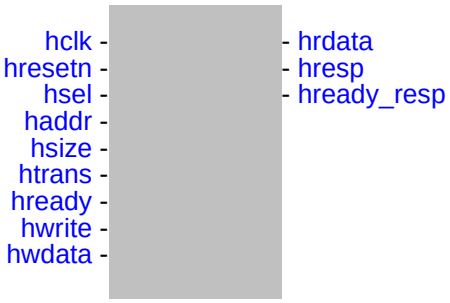


Table 4-3 AHB Signals

Port Name	I/O	Description
hclk	I	AHB slave interface clock. **Exists:** (DMAX_SLVIF_CLOCK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresetn	I	AHB slave interface reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of hclk. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (DMAX_SLVIF_CLOCK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
hsel	I	Slave select. Asserted when the current transfer to the AHB bus is intended for the DW_axi_dmac. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3 AHB Signals (Continued)

Port Name	I/O	Description
haddr[31:0]	I	Address. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hsize[2:0]	I	Transfer size. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
htrans[1:0]	I	Transfer type. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hready	I	Current transfer is complete. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
hwrite	I	Transfer direction. When high, this signal indicates a write transfer. When low, this signal indicates a read transfer. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** 1 for Write, 0 for Read

Table 4-3 AHB Signals (Continued)

Port Name	I/O	Description
hwdata[(DMAX_S_DATA_WIDTH-1):0]	I	Write data to slave. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hrdata[(DMAX_S_DATA_WIDTH-1):0]	O	Read data from slave. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresp[1:0]	O	Response type from slave. When the DW_axi_dmac is configured for the AHB Lite mode and instantiated in an AHB Lite system, the hresp[0] signal is connected to the hresp signal in the AHB bus fabric; the hresp[1] signal is left unconnected. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hready_resp	O	Transfer complete. **Exists:** Always **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals

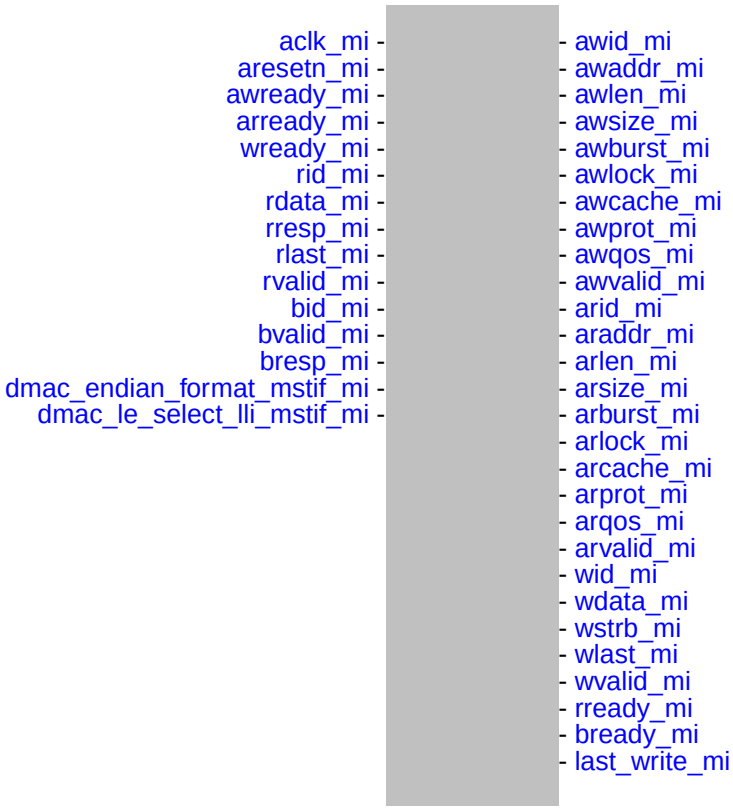


Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals

Port Name	I/O	Description
aclk_mi	I	AXI3/AXI4 Master i interface clock. **Exists:** (DMAX_MSTIF1_CLOCK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
aresetn_mi	I	AXI3/AXI4 Master i interface reset. Active low input that asynchronously resets the AXI3/AXI4 Master i interface to its default state. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of aclk_m1/dmac_core_clk. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
awid_mi[(DMAX_M_ID_WIDTH-1):0]	O	AXI3/AXI4 Master i interface write address ID. Identification tag for the write address group of signals. The upper 4 bits of awid_mi is derived from the channel number of the channel that is currently accessing the master interface (channel 1 = 4'b0000, channel 8 = 4'b0111, and so on). The lower bits are the same as the value programmed in the CHx_AXI_IDReg.AXI_Write_ID_Suffix field. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awaddr_mi[(DMAX_M_ADDR_WIDTH-1):0]	O	AXI3/AXI4 Master i interface write address. Specifies the address of the AXI write burst transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlen_mi[(DMAX_M_BURSTLEN_WIDTH-1):0]	O	AXI3/AXI4 Master i interface write burst length. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
awsize_mi[2:0]	O	AXI3/AXI4 Master i interface write burst size. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awburst_mi[1:0]	O	AXI3/AXI4 Master i interface write burst type. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock_mi[(DMAX_AXI_LOCK_WIDTH-1):0]	O	AXI3/AXI4 Master i interface write lock type. ■ awlock_mi[0] = 0 as DW_axi_dmac does not initiate exclusive transactions. ■ awlock_mi[1] = 0 as DW_axi_dmac does not support AXI bus locking. ■ awlock_mi[1] is not available in AXI4 mode. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache_mi[3:0]	O	AXI3/AXI4 Master i interface write cache type. CHx_CTL.AWCACHE field determines the value of this signal. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awprot_mi[2:0]	O	AXI3/AXI4 Master i interface write protection type. CHx_CTL.AWPROT field determines the value of this signal. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
awqos_mi[3:0]	O	AXI4 Master i interface Quality Of Service signal for write channel. The value of awqos_mi is the same as the value programmed in the CHx_AXI_QoSReg.AXI_AWQOS field of the channel that is currently accessing the master interface. **Exists:** (DMAX_HAS_QOS == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awvalid_mi	O	AXI3/AXI4 Master i interface write address valid. Indicates the availability of valid write address and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_HOLD_IO_EN == 1) ? "No": "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
awready_mi	I	AXI3/AXI4 Master i interface write address ready. Indicates the AXI slave readiness to accept write address and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
arid_mi[(DMAX_M_ID_WIDTH-1):0]	O	AXI3/AXI4 Master i interface read address ID. Identification tag for the read address group of signals. The upper 4 bits of arid_mi is derived from the channel number of the channel that is currently accessing the master interface. This varies for LLI fetch and source data transfer. ■ For source data transfer, arid_mi for channel 1 = 4'b0000, arid_mi for channel 8 = 4'b0111, and so on. ■ For LLI fetch access, arid_mi for channel 1 = 4'b1000, arid_mi for channel 8 = 4'b1111, and so on. Lower bits are the same as the value programmed in the CHx_AXI_IDReg.AXI_Read_ID_Suffix field. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
araddr_mi[(DMAX_M_ADDR_WIDTH-1):0]	O	AXI3/AXI4 Master i interface read address. Specifies the address of the AXI read burst transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlen_mi[(DMAX_M_BURSTLEN_WIDTH-1):0]	O	AXI3/AXI4 Master i interface read burst length. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize_mi[2:0]	O	AXI3/AXI4 Master i interface read burst size. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
arburst_mi[1:0]	O	AXI3/AXI4 Master i interface read burst type. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock_mi[(DMAX_AXI_LOCK_WIDTH-1):0]	O	AXI3/AXI4 Master i interface read lock type. ■ arlock_mi [0] = 0 as DW_axi_dmac does not initiate exclusive transactions. ■ arlock_mi [1] = 0 as DW_axi_dmac does not support AXI bus locking. ■ arlock_mi [1] = 0 in AXI4 mode. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache_mi[3:0]	O	AXI3/AXI4 Master i interface write cache type. The CHx_CTL.ARCACHE field determines the value of this signal. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arprot_mi[2:0]	O	AXI3/AXI4 Master i interface read protection type.The CHx_CTL.ARPROT field determines the value of this signal. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
arqos_mi[3:0]	O	AXI4 Master i interface Quality of Service signal for read channel. The value of arqos_mi is the same as the value programmed in the CHx_AXI_QoSReg.AXI_ARQOS field of the channel that is currently accessing the master interface. **Exists:** (DMAX_HAS_QOS == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arvalid_mi	O	AXI3/AXI4 Master i interface read address valid. Indicates the availability of a valid read address and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_HOLD_IO_EN == 1) ? "No": "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
arready_mi	I	AXI3/AXI4 Master i interface read address ready. Indicates the AXI slave readiness to accept read address and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wid_mi[(DMAX_M_ID_WIDTH-1):0]	O	AXI3 Master i interface write data ID. Identification tag for the write data group of signals. The upper 4 bits of wid_mi is derived from the channel number of the channel that is currently accessing the master interface (channel 1 = 4'b0000, channel 8 = 4'b0111, and so on). The lower bits are the same as the value programmed in the CHx_AXI_IDReg.AXI_Write_ID_Suffix field. DW_axi_dmac does not support write data interleaving. Therefore, the wid_mi for a particular write transaction is same as the awid_mi of the corresponding write request. **Exists:** (DMAX_MSTIF_MODE == 0) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
wdata_mi[(DMAX_M_DATA_WIDTH-1):0]	O	AXI3/AXI4 Master i interface write data. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wstrb_mi[((DMAX_M_DATA_WIDTH/8)-1):0]	O	AXI3/AXI4 Master i interface write data strobe. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wlast_mi	O	AXI3/AXI4 Master i interface write last. Indicates the last transfer in a write burst. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wvalid_mi	O	AXI3/AXI4 Master i interface write data valid. Indicates the availability of valid write data and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_HOLD_IO_EN == 1) ? "No": "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
wready_mi	I	AXI3/AXI4 Master i interface write data ready. Indicates the AXI slave readiness to accept write data and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
rid_mi[(DMAX_M_ID_WIDTH-1):0]	I	AXI3/AXI4 Master i interface read data ID. Identification tag for the read data group of signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rdata_mi[(DMAX_M_DATA_WIDTH-1):0]	I	AXI3/AXI4 Master i interface read data. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp_mi[1:0]	I	AXI3/AXI4 Master i Interface read response. AXI slave indicates the status of read transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rlast_mi	I	AXI3/AXI4 Master i interface read last. Indicates the last transfer in a read burst. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rvalid_mi	I	AXI3/AXI4 Master i interface read data valid. Indicates the availability of valid read data and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
rready_mi	O	AXI3/AXI4 Master i interface read data ready. Indicates the DW_axi_dmac readiness to accept read address and associated control signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_HOLD_IO_EN == 1) ? "No": "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
bid_mi[(DMAX_M_ID_WIDTH-1):0]	I	AXI3/AXI4 Master i interface write response ID. Identification tag for the write response group of signals. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bvalid_mi	I	AXI3/AXI4 Master i interface write response valid. AXI slave indicates the availability of a valid write response. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bresp_mi[1:0]	I	AXI3/AXI4 Master i interface write response. AXI slave indicates the status of write transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
bready_mi	O	AXI3/AXI4 Master i interface write response ready. Indicates the DW_axi_dmac readiness to accept write response. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_HOLD_IO_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 Master Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals (Continued)

Port Name	I/O	Description
last_write_mi	O	Last write data of block transfer. **Exists:** (DMAX_ENABLE_LAST_WRITE == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dmac_endian_format_mstif_mi	I	Master i interface endian format selection pins. Following are the supported endian formats: ■ 0: Little Endian ■ 1: Big Endian BE-8 **Exists:** (DMAX_STATIC_ENDIAN_SELECT_MSTIF == 0) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dmac_le_select_lli_mstif_mi	I	Master i interface endian format selection pin for LLI access (LLI fetch and LLI status write-back). The following are the supported endian formats: ■ 0: Endian scheme used for LLI access is the same as that used for data access Mi interface. ■ 1: Little endian method is used for LLI access irrespective of the endian scheme used for data access Mi interface. **Exists:** (DMAX_LLI_ENDIAN_SELECTION_PIN_EN == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 Hardware Handshaking Signals

Table 4-5 Hardware Handshaking Signals

Port Name	I/O	Description
hs_clk[(DMAX_HS_CLKRST_WIDTH-1):0]	I	Hardware Handshake Interface Clock. **Exists:** (DMAX_ASYNC_HS_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hs_rstn[(DMAX_HS_CLKRST_WIDTH-1):0]	I	Hardware Handshake Interface reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of hs_clk. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (DMAX_ASYNC_HS_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dma_req[(DMAX_NUM_HS_IF-1):0]	I	DMA transaction request from peripheral. Where, y = 1...DMAX_NUM_HS_IF and DMAX_HS1_ASYNC_CLK corresponds to dma_req[0], DMAX_HS2_ASYNC_CLK corresponds to dma_req[1], and so on. **Exists:** (DMAX_NUM_HS_IF > 0) **Synchronous To:** ((DMAX_HS(y)_ASYNC_CLK == 1) ? "hs_clk[y-1]" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 Hardware Handshaking Signals (Continued)

Port Name	I/O	Description
dma_single[(DMAX_NUM_HS_IF-1):0]	I	Single transfer request/status. Where, y = 1...DMAX_NUM_HS_IF and DMAX_HS1_ASYNC_CLK corresponds to dma_single[0], DMAX_HS2_ASYNC_CLK corresponds to dma_single[1], and so on. **Exists:** (DMAX_NUM_HS_IF > 0) **Synchronous To:** ((DMAX_HS(y)_ASYNC_CLK == 1) ? "hs_clk[y-1]" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dma_last[(DMAX_NUM_HS_IF-1):0]	I	Last transaction in block indicator. Where, y = 1...DMAX_NUM_HS_IF and DMAX_HS1_ASYNC_CLK corresponds to dma_last[0], DMAX_HS2_ASYNC_CLK corresponds to dma_last[1], and so on. **Exists:** (DMAX_NUM_HS_IF > 0) **Synchronous To:** ((DMAX_HS(y)_ASYNC_CLK == 1) ? "hs_clk[y-1]" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dma_ack[(DMAX_NUM_HS_IF-1):0]	O	Transaction complete acknowledge signal. Where, y = 1...DMAX_NUM_HS_IF and DMAX_HS1_ASYNC_CLK corresponds to dma_ack[0], DMAX_HS2_ASYNC_CLK corresponds to dma_ack[1], and so on. **Exists:** (DMAX_NUM_HS_IF > 0) **Synchronous To:** ((DMAX_HS(y)_ASYNC_CLK == 1) ? "hs_clk[y-1]" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dma_finish[(DMAX_NUM_HS_IF-1):0]	O	DMA block complete signal. Where, y = 1...DMAX_NUM_HS_IF and DMAX_HS1_ASYNC_CLK corresponds to dma_finish[0], DMAX_HS2_ASYNC_CLK corresponds to dma_fiish[1], and so on. **Exists:** (DMAX_NUM_HS_IF > 0) **Synchronous To:** ((DMAX_HS(y)_ASYNC_CLK == 1) ? "hs_clk[y-1]" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 Hardware Handshaking Signals (Continued)

Port Name	I/O	Description
debug_dma_ack[(DMAX_NUM_HS_IF-1):0]	O	Debug Transaction complete acknowledge signal. This signal is always synchronous to dmac_core_clock, irrespective type Asynchronous Handshake configuration mode chosen. **Exists:** (DMAX_NUM_HS_IF > 0) && (DMAX_ASYNC_HS_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.6 Debug Interface Signals

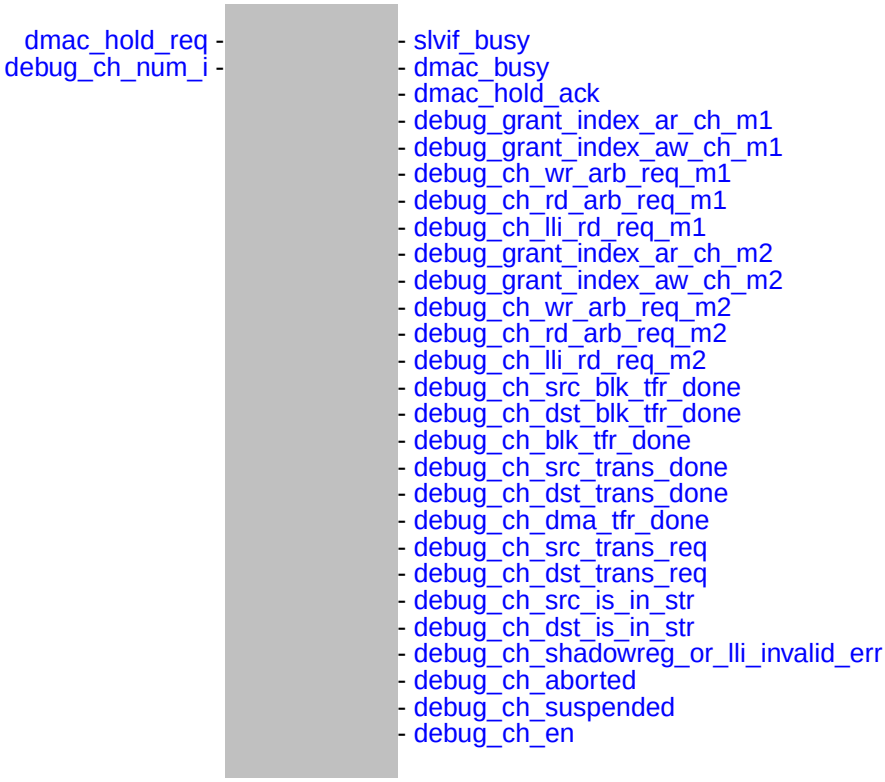


Table 4-6 Debug Interface Signals

Port Name	I/O	Description
dmac_hold_req	I	DMAC Core Hold Request. A request to put DW_axi_dmac in hold (freeze) mode. Asserting this request puts the entire DW_axi_dmac in freeze mode without violating the AXI protocol. To exit the hold mode, this signal can be de-asserted after DW_axi_dmac asserts dmac_hold_ack. **Exists:** (DMAX_HOLD_IO_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
slvif_busy	O	Slave Interface status signal indicating that whether the slave interface is busy or not. This signal is asserted if there is an active transfer on the slave interface. **Exists:** (DMAX_SLVIF_STATUS_OP_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High
dmac_busy	O	DMAC status signal indicates whether DW_axi_dmac core is busy or not. This signal is asserted if any of the channels in DW_axi_dmac are in a non-idle state. A non-idle state of the channel corresponds to the following cases: ■ There is an active transfer on the Master Interface (including posted requests) corresponding to one or multiple Channels. ■ Any of the Channels are about to initiate a transfer, which might correspond to waiting for the grant of the master interface from the arbiter. **Exists:** (DMAX_CORE_STATUS_OP_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
dmac_hold_ack	O	DMAC Core Hold request acknowledgement. DW_axi_dmac asserts this signal after entering the hold (freeze) mode. This signal is de-asserted when dmac_hold_req is de-asserted. It is not allowed to de-assert dmac_hold_req before asserting dmac_hold_ack. **Exists:** (DMAX_HOLD_IO_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_grant_index_ar_ch_m1[(LOG2_DMAX_ARB_RD_REQ_WIDTH-1):0]	O	Debug port AXI Master Interface 1 Read Address Channel Arbiter grant index. Where the Read Arbiter request is combination of: When LLI is enabled for at least one channel (DMAX_HAS_LLI_PARAM = 1): {CH(x)_LLI_REQ, CH(x)_DST_REQ, CH(x)_SRC_REQ, ... CH2_LLI_REQ, CH2_DST_REQ, CH2_SRC_REQ, CH1_LLI_REQ, CH1_DST_REQ, CH1_SRC_REQ} When LLI is not enabled for any of the channel (DMAX_HAS_LLI_PARAM = 0): {CH(x)_DST_REQ, CH(x)_SRC_REQ, ... CH2_DST_REQ, CH2_SRC_REQ, CH1_DST_REQ, CH1_SRC_REQ} Where x = DMAX_NUM_CHANNELS. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_grant_index_aw_ch_m1[(LOG2_DMAX_ARB_WR_REQ_WIDTH-1):0]	O	Debug port AXI Master Interface 1 Write Address Channel Arbiter grant index. Where the Write Arbiter request is combination of: When LLI is enabled for at least one channel (DMAX_HAS_LLI_PARAM = 1): {CH(x)_LLI_REQ, CH(x)_DST_REQ, ... CH2_LLI_REQ, CH2_DST_REQ, CH1_LLI_REQ, CH1_DST_REQ} When LLI is not enabled for any of the channel (DMAX_HAS_LLI_PARAM = 0): {CH(x)_DST_REQ, ... CH2_DST_REQ, CH1_DST_REQ} Where x = DMAX_NUM_CHANNELS. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_wr_arb_req_m1[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 1 Write Arbiter Request. Each bit of the vector corresponds to respective channels Write Arbiter request. Each request is a combination of Destination Data Write request and LLI Write request (DMAX_HAS_LLI_PARAM==1) of a channel. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_ch_rd_arb_req_m1[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 1 Read Arbiter Request. Each bit of the vector corresponds to respective Read Arbiter request. Each request is a combination of Source Data Read request (Data Fetch and SSTAT Fetch) and Destination Read request (DSTAT Fetch) of a channel. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_lli_rd_req_m1[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 1 LLI Read Arbiter Request. Each bit of the vector corresponds to respective channels LLI Read Arbiter request. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_HAS_LLI_PARAM == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_grant_index_ar_ch_m2[(LOG2_DMAX_ARB_RD_REQ_WIDTH-1):0]	O	Debug port AXI Master Interface 2 Read Address Channel Arbiter grant index. Where the Read Arbiter request is combination of: When LLI is enabled for at least one channel (DMAX_HAS_LLI_PARAM = 1): {CH(x)_LLI_REQ, CH(x)_DST_REQ, CH(x)_SRC_REQ, ... CH2_LLI_REQ, CH2_DST_REQ, CH2_SRC_REQ, CH1_LLI_REQ, CH1_DST_REQ, CH1_SRC_REQ} When LLI is not enabled for any of the channel (DMAX_HAS_LLI_PARAM = 0): {CH(x)_DST_REQ, CH(x)_SRC_REQ, ... CH2_DST_REQ, CH2_SRC_REQ, CH1_DST_REQ, CH1_SRC_REQ} Where x = DMAX_NUM_CHANNELS. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_grant_index_aw_ch_m2[(LOG2_DMAX_ARB_WR_REQ_WIDTH-1):0]	O	Debug port AXI Master Interface 2 Write Address Channel Arbiter grant index. Where the Write Arbiter request is combination of: When LLI is enabled for at least one channel (DMAX_HAS_LLI_PARAM = 1): {CH(x)_LLI_REQ, CH(x)_DST_REQ, ... CH2_LLI_REQ, CH2_DST_REQ, CH1_LLI_REQ, CH1_DST_REQ} When LLI is not enabled for any of the channel (DMAX_HAS_LLI_PARAM = 0): {CH(x)_DST_REQ, ... CH2_DST_REQ, CH1_DST_REQ} Where x = DMAX_NUM_CHANNELS **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_wr_arb_req_m2[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 2 Write Arbiter Request. Each bit of the vector corresponds to Write Arbiter request of the respective channel. Each request is a combination of Destination Data Write request and LLI Write request (DMAX_HAS_LLI_PARAM==1) of a channel. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_rd_arb_req_m2[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 2 Read Arbiter Request. Each bit of the vector corresponds to Read Arbiter request of the respective channel. Each request is a combination of Source Data Read request (Data Fetch and SSTAT Fetch) and Destination Data Read request (DSTAT Fetch) of a channel. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_ch_lli_rd_req_m2[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Master Interface 2 LLI Read Arbiter Request. Each bit of the vector corresponds to LLI Read Arbiter request of the respective channel. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_HAS_LLI_PARAM == 1) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_num_i[(LOG2_DMAX_NUM_CHANNELS-1):0]	I	Debug Interface Channel number. This channel number is used to multiplex debug signals from all channels to provide one set of debug signals corresponding to a Channel number - debug_ch_num_i. The following Channel specific debug signals are multiplexed based on the debug_ch_num_i: ■ ch_src_blk_tfr_done ■ ch_dst_blk_tfr_done ■ ch_blk_tfr_done ■ ch_src_trans_done ■ ch_dst_trans_done ■ ch_dma_tfr_done ■ ch_src_trans_req ■ ch_dst_trans_req ■ ch_src_is_in_str ■ ch_dst_is_in_str ■ ch_aborted ■ ch_suspended ■ ch_shadowreg_or_lli_invalid_err **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_src_blk_tfr_done	O	Multiplexed - Source Block Transfer Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_ch_dst_blk_tfr_done	O	Multiplexed - Destination Block Transfer Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_blk_tfr_done	O	Multiplexed - DMA Block Transfer Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_src_trans_done	O	Multiplexed - Source Transaction Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_dst_trans_done	O	Multiplexed - Destination Transaction Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_dma_tfr_done	O	Multiplexed - DMA Transfer Done debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_ch_src_trans_req	O	Multiplexed - internally generated Source Transaction request (generated based on the Source dma_req* and dma_sgl_req*) debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_dst_trans_req	O	Multiplexed - internally generated Destination Transaction request (generated based on the Destination dma_req* and dma_sgl_req*) debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_src_is_in_str	O	Multiplexed - Source State machine is in Single transaction region debug signal corresponding to the channel debug_ch_num_i. For more information, see "Single Transaction Region" section of the databook. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_dst_is_in_str	O	Multiplexed - Destination State machine is in Single transaction region debug signal corresponding to the channel debug_ch_num_i. For more information, see "Single Transaction Region" section of the databook. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 Debug Interface Signals (Continued)

Port Name	I/O	Description
debug_ch_shadowreg_or_lli_invalid_err	O	Multiplexed - Shadow register or LLI Invalid error debug signal corresponding to the channel debug_ch_num_i. For more information, see "Programming Flow for Shadow-Register-Based Multi-Block Transfer" and "Programming Flow for Linked-List-Based Multi-Bock Transfer" sections of the databook. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_aborted	O	Multiplexed - Channel Abort status (generated due to Channel Suspend request, Channel disable, User Channel abort, or due to reception of AXI error response) debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_CH_ABORT_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_suspended	O	Multiplexed - Channel Suspend status debug signal corresponding to the channel debug_ch_num_i. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
debug_ch_en[(DMAX_NUM_CHANNELS-1):0]	O	DMA Channel enable debug signal. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.7 Interrupt Interface Signals

Table 4-7 Interrupt Interface Signals

Port Name	I/O	Description
intr	O	Logical OR of all individual interrupts. **Exists:** (DMAX_INTR_IO_TYPE != 1) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" : "dmac_core_clock" **Registered:** ((DMAX_INTR_SYNC2SLVCLK == 1) && (DMAX_INTR_IO_TYPE == 0)) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Logical OR of all individual interrupts of each enabled channels. **Exists:** (DMAX_INTR_IO_TYPE > 0) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" : "dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
intr_cmnreg	O	Logical OR of all common register interrupts. **Exists:** (DMAX_INTR_IO_TYPE > 0) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" :"dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.8 Low Power Interface Signals

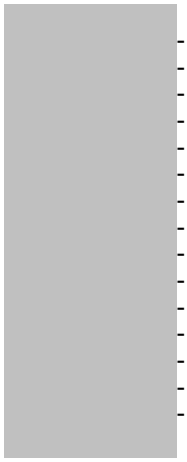


Table 4-8 Low Power Interface Signals

Port Name	I/O	Description
dmac_lp_req	O	Global Low Power request, generated based on the Global Context Sensitive Low Power feature. **Exists:** (DMAX_CSLP_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
ch_lp_req[(DMAX_NUM_CHANNELS-1):0]	O	DMA Channel Low Power request, generated based on the DMA Channel Context Sensitive Low Power feature. **Exists:** (DMAX_CHNL_CSLP_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
sbiu_lp_req	O	SBIU Low Power request, generated based on the SBIU Context Sensitive Low Power feature. **Exists:** (DMAX_SBIU_CSLP_EN== 1) **Synchronous To:** (DMAX_SLVIF_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-8 Low Power Interface Signals (Continued)

Port Name	I/O	Description
mxif_arch_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI AR Channel Low Power request, generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_arch_oclk_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI AR Channel Low Power request (synchronous to other clock - aclk_mi), generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** ((DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "None") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_awch_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI AW Channel Low Power request, generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_awch_oclk_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI AW Channel Low Power request (synchronous to other clock - aclk_mi), generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** ((DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "None") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_wch_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI W Channel Low Power request, generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-8 Low Power Interface Signals (Continued)

Port Name	I/O	Description
mxif_wch_oclk_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI W Channel Low Power request (synchronous to other clock - aclk_mi), generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** ((DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "None") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_bch_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI B Channel Low Power request, generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** aclk_mi **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_bch_oclk_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI B Channel Low Power request (synchronous to other clock - dmac_core_clock), generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_rch_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI R Channel Low Power request, generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** aclk_mi **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_rch_oclk_lp_req[(DMAX_NUM_MASTER_IF-1):0]	O	AXI R Channel Low Power request (synchronous to other clock - dmac_core_clock), generated based on the AXI Master Interface Context Sensitive Low Power feature. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-8 Low Power Interface Signals (Continued)

Port Name	I/O	Description
mxif_r_ch_idle[(DMAX_NUM_MASTER_IF-1):0]	O	Debug port AXI Master interface Read Data channel Idle status. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_CSLP_EN == 1) **Synchronous To:** ((DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
mxif_b_ch_idle[(DMAX_NUM_MASTER_IF-1):0]	O	Debug port AXI Master interface Write Response channel Idle status. **Exists:** (DMAX_DEBUG_PORTS_EN > 0) && (DMAX_CSLP_EN == 1) **Synchronous To:** ((DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock") **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Register Descriptions

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

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

When you configure the core 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 core 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

The Exist 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 Exists expression for a bit field in a register assumes that the register exists and does not include the conditions for existence of the register.

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
Common_Registers_Address_Block on page 162	DW_axi_dmac common register address block **Exists:** Always
DW_axi_dmac Channel x register address block (for x = 1; x <= DMAX_NUM_CHANNELS) on page 339	DW_axi_dmac Channel x register address block **Exists:** DMAX_NUM_CHANNELS >= x
DUMMY	**Exists:** Always

5.1 DW_axi_dmac_mem_map/Common_Registers_Address_Block Registers

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

Table 5-5 Registers for Address Block: DW_axi_dmac_mem_map/Common_Registers_Address_Block

Register	Offset	Description
DMAC_IDREG on page 163	0x0	DMAC ID Register contains a 64-bit value that is hardwired and read back by a read to the DW_axi_dmac...
DMAC_COMPVERREG on page 164	0x8	This register contains a 64-bit value that is hardwired and read back by a read to the DW_axi_dmac...
DMAC_CFGREG on page 165	0x10	This register is used to enable the DW_axi_dmac, which must be done before any channel activity...
DMAC_CHENREG on page 167	0x18	This is DW_axi_dmac Channel Enable Register. If software wants to set up a new channel, it can read...
DMAC_CHENREG2 on page 196	0x18	This is DW_axi_dmac Channel Enable Register. If software wants to set up a new channel, it can read...
DMAC_CHSUSPREG on page 224	0x20	This is DW_axi_dmac Channel Suspend Register. The channel suspend bit, DMAC_ChSuspReg.CH_SUSP, is...
DMAC_CHABORTREG on page 265	0x28	This is DW_axi_dmac Channel Abort Register. The channel abort bit, DMAC_ChAbortReg.CH_ABORT, is...
DMAC_INTSTATUSREG on page 308	0x30	DMAC Interrupt Status Register captures the combined channel interrupt for each channel and Combined...
DMAC_INTSTATUSREG2 on page 311	0x30	DMAC Interrupt Status Register captures the combined channel interrupt for each channel and Combined...
DMAC_COMMONREG_INTCLEARREG on page 319	0x38	Writing 1 to specific field clears the corresponding field in DMAC Common register Interrupt Status...
DMAC_COMMONREG_INTSTATUS_ENABLEREG on page 322	0x40	Writing 1 to specific field enables the corresponding interrupt status generation in DMAC Common...
DMAC_COMMONREG_INTSIGNAL_ENABLEREG on page 326	0x48	Writing 1 to specific field will propagate the corresponding interrupt status in DMAC Common register...
DMAC_COMMONREG_INTSTATUSREG on page 330	0x50	This Register captures Slave interface access errors. - Decode Error. - Write to read only register....
DMAC_RESETREG on page 335	0x58	This register is used to initiate the Software Reset to DW_axi_dmac.
DMAC_LOWPOWER_CFGREG on page 336	0x60	This register contains the fields that configures the Context Sensitive Low Power feature. This...

5.1.1 DMAC_IDREG

- **Description:** DMAC ID Register contains a 64-bit value that is hardwired and read back by a read to the DW_axi_dmac ID Register.
- **Size:** 64 bits
- **Offset:** 0x0
- **Exists:** Always

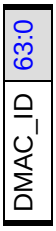


Table 5-6 Fields for Register: DMAC_IDREG

Bits	Name	Memory Access	Description
63:0	DMAC_ID	R	DMAC ID Number. **Value After Reset:** DMAX_ID_NUM **Exists:** Always

5.1.2 DMAC_COMPVERREG

- **Description:** This register contains a 64-bit value that is hardwired and read back by a read to the DW_axi_dmac Component Version Register.
- **Size:** 64 bits
- **Offset:** 0x8
- **Exists:** Always

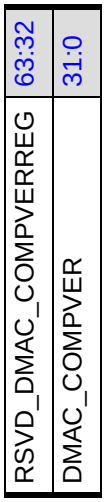


Table 5-7 Fields for Register: DMAC_COMPVERREG

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_COMPVERREG	R	DMAC_COMPVERREG Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
31:0	DMAC_COMPVER	R	DMAC Component Version Number. **Value After Reset:** DMAX_COMP_VER **Exists:** Always

5.1.3 DMAC_CFGREG

- **Description:** This register is used to enable the DW_axi_dmac, which must be done before any channel activity can begin. This register also contains global interrupt enable bit.
- **Size:** 64 bits
- **Offset:** 0x10
- **Exists:** Always

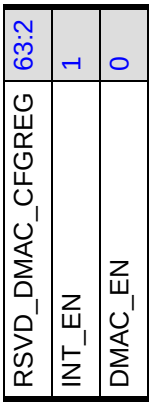


Table 5-8 Fields for Register: DMAC_CFGREG

Bits	Name	Memory Access	Description
63:2	RSVD_DMAC_CFGREG	R	DMAC_CFGREG Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
1	INT_EN	R/W	This bit is used to globally enable the interrupt generation. ■ 0: DW_axi_dmac Interrupts are disabled ■ 1: DW_axi_dmac Interrupt logic is enabled. **Values:** ■ 0x1 (ENABLED): DW_axi_dmac Interrupts are enabled ■ 0x0 (DISABLED): DW_axi_dmac Interrupts are disabled **Value After Reset:** 0x0 **Exists:** Always

Table 5-8 Fields for Register: DMAC_CFGREG (Continued)

Bits	Name	Memory Access	Description
0	DMAC_EN	R/W	This bit is used to enable the DW_axi_dmac. ■ 0: DW_axi_dmac disabled ■ 1: DW_axi_dmac enabled **NOTE**: If this bit DMAC_EN bit is cleared while any channel is still active, then this bit still returns 1 to indicate that there are channels still active until DW_axi_dmac hardware has terminated all activity on all channels, at which point this bit returns zero (0). **Values:** ■ 0x1 (ENABLED): DW_axi_dmac is enabled ■ 0x0 (DISABLED): DW_axi_dmac is disabled **Value After Reset:** 0x0 **Exists:** Always

5.1.4 DMAC_CHENREG

- **Description:** This is DW_axi_dmac Channel Enable Register. If software wants to set up a new channel, it can read this register to find out which channels are currently inactive and then enable an inactive channel with the required priority.

  All bits of this register are cleared to 0 when the DW_axi_dmac Global Enable bit (DMAC_CfgReg.DMAC_EN) is 0. When DMAC_CfgReg.DMAC_EN is 0, a write to the DMAC_ChEnReg register is ignored and a read always reads back 0.

  The channel enable bit, DMAC_ChEnReg.CH_EN, is written only if the corresponding channel write enable bit, DMAC_ChEnReg.CH_EN_WE, is asserted on the same slave interface write transfer. For example, writing hex XXXX01X1 writes a 1 into DMAC_ChEnReg [0], while DMAC_ChEnReg [7:1] remains unchanged. Writing hex XXXX00XX leaves DMAC_ChEnReg [7:0] unchanged.

  The channel suspend bit, DMAC_ChEnReg.CH_SUSP, is written only if the corresponding channel write enable bit, DMAC_ChEnReg.CH_SUSP_WE, is asserted on the same slave interface write transfer. For example, writing hex 01X1XXXX writes a 1 into DMAC_ChEnReg [16], while DMAC_ChEnReg [23:17] remains unchanged. Writing hex 00XXXXXX leaves DMAC_ChEnReg [23:16] unchanged. The channel abort bit, DMAC_ChEnReg.CH_ABORT, is written only if the corresponding channel write enable bit, DMAC_ChEnReg.CH_ABORT_WE, is asserted on the same slave interface write transfer.

- **Size:** 64 bits
- **Offset:** 0x18
- **Exists:** DMAX_NUM_CHANNELS <= 8

63:48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_DMAC_CHENREG	CH8_ABORT_WE	CH7_ABORT_WE	CH6_ABORT_WE	CH5_ABORT_WE	CH4_ABORT_WE	CH3_ABORT_WE	CH2_ABORT_WE	CH1_ABORT_WE	CH8_ABORT	CH7_ABORT	CH6_ABORT	CH5_ABORT	CH4_ABORT	CH3_ABORT	CH2_ABORT	CH1_ABORT	CH8_SUSP_WE	CH7_SUSP_WE	CH6_SUSP_WE	CH5_SUSP_WE	CH4_SUSP_WE	CH3_SUSP_WE	CH2_SUSP_WE	CH1_SUSP_WE	CH8_SUSP	CH7_SUSP	CH6_SUSP	CH5_SUSP	CH4_SUSP	CH3_SUSP	CH2_SUSP	CH1_SUSP	CH8_EN_WE	CH7_EN_WE	CH6_EN_WE	CH5_EN_WE	CH4_EN_WE	CH3_EN_WE	CH2_EN_WE	CH1_EN_WE	CH8_EN	CH7_EN	CH6_EN	CH5_EN	CH4_EN	CH3_EN	CH2_EN	CH1_EN

Table 5-9 Fields for Register: DMAC_CHENREG

Bits	Name	Memory Access	Description
63:48	RSVD_DMAC_CHENREG	R	DMAC_CHENREG Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
47	CH8_ABORT_WE	* Varies	This bit is used to write enable the Channel-8 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH8_ABORT): Enable Write to CH8_ABORT bit ■ 0x0 (DISABLE_WR_CH8_ABORT): Disable Write to CH8_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
46	CH7_ABORT_WE	* Varies	This bit is used to write enable the Channel-7 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH7_ABORT): Enable Write to CH7_ABORT bit ■ 0x0 (DISABLE_WR_CH7_ABORT): Disable Write to CH7_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
45	CH6_ABORT_WE	* Varies	This bit is used to write enable the Channel-6 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH6_ABORT): Enable Write to CH6_ABORT bit ■ 0x0 (DISABLE_WR_CH6_ABORT): Disable Write to CH6_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
44	CH5_ABORT_WE	* Varies	This bit is used to write enable the Channel-5 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH5_ABORT): Enable Write to CH5_ABORT bit ■ 0x0 (DISABLE_WR_CH5_ABORT): Disable Write to CH5_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
43	CH4_ABORT_WE	* Varies	This bit is used to write enable the Channel-4 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH4_ABORT): Enable Write to CH4_ABORT bit ■ 0x0 (DISABLE_WR_CH4_ABORT): Disable Write to CH4_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
42	CH3_ABORT_WE	* Varies	This bit is used to write enable the Channel-3 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH3_ABORT): Enable Write to CH3_ABORT bit ■ 0x0 (DISABLE_WR_CH3_ABORT): Disable Write to CH3_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
41	CH2_ABORT_WE	* Varies	This bit is used to write enable the Channel-2 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH2_ABORT): Enable Write to CH2_ABORT bit ■ 0x0 (DISABLE_WR_CH2_ABORT): Disable Write to CH2_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
40	CH1_ABORT_WE	* Varies	This bit is used to write enable the Channel-1 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH1_ABORT): Enable Write to CH1_ABORT bit ■ 0x0 (DISABLE_WR_CH1_ABORT): Disable Write to CH1_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
39	CH8_ABORT	* Varies	Channel-8 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH8_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH8_ABORT): Request for Channel-8 Abort ■ 0x0 (DISABLE_CH8_ABORT): No Request for Channel-8 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
38	CH7_ABORT	* Varies	Channel-7 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH7_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH7_ABORT): Request for Channel-7 Abort ■ 0x0 (DISABLE_CH7_ABORT): No Request for Channel-7 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
37	CH6_ABORT	* Varies	Channel-6 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH6_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH6_ABORT): Request for Channel-6 Abort ■ 0x0 (DISABLE_CH6_ABORT): No Request for Channel-6 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
36	CH5_ABORT	* Varies	Channel-5 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH5_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH5_ABORT): Request for Channel-5 Abort ■ 0x0 (DISABLE_CH5_ABORT): No Request for Channel-5 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
35	CH4_ABORT	* Varies	Channel-4 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH4_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH4_ABORT): Request for Channel-4 Abort ■ 0x0 (DISABLE_CH4_ABORT): No Request for Channel-4 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
34	CH3_ABORT	* Varies	Channel-3 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH3_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH3_ABORT): Request for Channel-3 Abort ■ 0x0 (DISABLE_CH3_ABORT): No Request for Channel-3 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
33	CH2_ABORT	* Varies	Channel-2 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH2_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH2_ABORT): Request for Channel-2 Abort ■ 0x0 (DISABLE_CH2_ABORT): No Request for Channel-2 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
32	CH1_ABORT	* Varies	Channel-1 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH1_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH1_ABORT): Request for Channel-1 Abort ■ 0x0 (DISABLE_CH1_ABORT): No Request for Channel-1 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}
31	CH8_SUSP_WE	W	This bit is used as a write enable to the Channel-8 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH8_SUSP): Enable Write to respective CH8_SUSP bit ■ 0x0 (DISABLE_WR_CH8_SUSP): Disable Write to CH8_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

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

Bits	Name	Memory Access	Description
30	CH7_SUSP_WE	W	This bit is used as a write enable to the Channel-7 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH7_SUSP): Enable Write to respective CH7_SUSP bit ■ 0x0 (DISABLE_WR_CH7_SUSP): Disable Write to CH7_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true
29	CH6_SUSP_WE	W	This bit is used as a write enable to the Channel-6 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH6_SUSP): Enable Write to respective CH6_SUSP bit ■ 0x0 (DISABLE_WR_CH6_SUSP): Disable Write to CH6_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true
28	CH5_SUSP_WE	W	This bit is used as a write enable to the Channel-5 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH5_SUSP): Enable Write to respective CH5_SUSP bit ■ 0x0 (DISABLE_WR_CH5_SUSP): Disable Write to CH5_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true
27	CH4_SUSP_WE	W	This bit is used as a write enable to the Channel-4 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH4_SUSP): Enable Write to respective CH4_SUSP bit ■ 0x0 (DISABLE_WR_CH4_SUSP): Disable Write to CH4_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true

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

Bits	Name	Memory Access	Description
26	CH3_SUSP_WE	W	This bit is used as a write enable to the Channel-3 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH3_SUSP): Enable Write to respective CH3_SUSP bit ■ 0x0 (DISABLE_WR_CH3_SUSP): Disable Write to CH3_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true
25	CH2_SUSP_WE	W	This bit is used as a write enable to the Channel-2 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH2_SUSP): Enable Write to respective CH2_SUSP bit ■ 0x0 (DISABLE_WR_CH2_SUSP): Disable Write to CH2_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true
24	CH1_SUSP_WE	W	This bit is used as a write enable to the Channel-1 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH1_SUSP): Enable Write to respective CH1_SUSP bit ■ 0x0 (DISABLE_WR_CH1_SUSP): Disable Write to CH1_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true

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

Bits	Name	Memory Access	Description
23	CH8_SUSP	R/W	Channel-8 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH8_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH8_SUSP bit to 1 and polls CH8_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH8_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH8_SUSP bit to 0, after DW_axi_dmac sets CH8_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH8_SUSP): Request to Suspended Channel-8 ■ 0x0 (DISABLE_CH8_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

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

Bits	Name	Memory Access	Description
22	CH7_SUSP	R/W	Channel-7 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH7_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH7_SUSP bit to 1 and polls CH7_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH7_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH7_SUSP bit to 0, after DW_axi_dmac sets CH7_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH7_SUSP): Request to Suspended Channel-7 ■ 0x0 (DISABLE_CH7_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true

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

Bits	Name	Memory Access	Description
21	CH6_SUSP	R/W	Channel-6 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH6_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH6_SUSP bit to 1 and polls CH6_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH6_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH6_SUSP bit to 0, after DW_axi_dmac sets CH6_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH6_SUSP): Request to Suspended Channel-6 ■ 0x0 (DISABLE_CH6_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true

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

Bits	Name	Memory Access	Description
20	CH5_SUSP	R/W	Channel-5 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH5_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH5_SUSP bit to 1 and polls CH5_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH5_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH5_SUSP bit to 0, after DW_axi_dmac sets CH5_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH5_SUSP): Request to Suspended Channel-5 ■ 0x0 (DISABLE_CH5_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true

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

Bits	Name	Memory Access	Description
19	CH4_SUSP	R/W	Channel-4 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH4_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH4_SUSP bit to 1 and polls CH4_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH4_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH4_SUSP bit to 0, after DW_axi_dmac sets CH4_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH4_SUSP): Request to Suspended Channel-4 ■ 0x0 (DISABLE_CH4_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true

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

Bits	Name	Memory Access	Description
18	CH3_SUSP	R/W	Channel-3 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH3_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH3_SUSP bit to 1 and polls CH3_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH3_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH3_SUSP bit to 0, after DW_axi_dmac sets CH3_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH3_SUSP): Request to Suspended Channel-3 ■ 0x0 (DISABLE_CH3_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true

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

Bits	Name	Memory Access	Description
17	CH2_SUSP	R/W	Channel-2 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH2_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH2_SUSP bit to 1 and polls CH2_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH2_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH2_SUSP bit to 0, after DW_axi_dmac sets CH2_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH2_SUSP): Request to Suspended Channel-2 ■ 0x0 (DISABLE_CH2_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true

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

Bits	Name	Memory Access	Description
16	CH1_SUSP	R/W	Channel-1 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH1_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH1_SUSP bit to 1 and polls CH1_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH1_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH1_SUSP bit to 0, after DW_axi_dmac sets CH1_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH1_SUSP): Request to Suspended Channel-1 ■ 0x0 (DISABLE_CH1_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true
15	CH8_EN_WE	W	DW_axi_dmac Channel-8 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH8_EN): Enable Write to CH8_EN bit ■ 0x0 (DISABLE_WR_CH8_EN): Disable Write to respective CH8_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

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

Bits	Name	Memory Access	Description
14	CH7_EN_WE	W	DW_axi_dmac Channel-7 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH7_EN): Enable Write to CH7_EN bit ■ 0x0 (DISABLE_WR_CH7_EN): Disable Write to respective CH7_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true
13	CH6_EN_WE	W	DW_axi_dmac Channel-6 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH6_EN): Enable Write to CH6_EN bit ■ 0x0 (DISABLE_WR_CH6_EN): Disable Write to respective CH6_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true
12	CH5_EN_WE	W	DW_axi_dmac Channel-5 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH5_EN): Enable Write to CH5_EN bit ■ 0x0 (DISABLE_WR_CH5_EN): Disable Write to respective CH5_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true

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

Bits	Name	Memory Access	Description
11	CH4_EN_WE	W	DW_axi_dmac Channel-4 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH4_EN): Enable Write to CH4_EN bit ■ 0x0 (DISABLE_WR_CH4_EN): Disable Write to respective CH4_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true
10	CH3_EN_WE	W	DW_axi_dmac Channel-3 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH3_EN): Enable Write to CH3_EN bit ■ 0x0 (DISABLE_WR_CH3_EN): Disable Write to respective CH3_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true
9	CH2_EN_WE	W	DW_axi_dmac Channel-2 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH2_EN): Enable Write to CH2_EN bit ■ 0x0 (DISABLE_WR_CH2_EN): Disable Write to respective CH2_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true

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

Bits	Name	Memory Access	Description
8	CH1_EN_WE	W	DW_axi_dmac Channel-1 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH1_EN): Enable Write to CH1_EN bit ■ 0x0 (DISABLE_WR_CH1_EN): Disable Write to respective CH1_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true
7	CH8_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-8. ■ 0: DW_axi_dmac Channel-8 is disabled ■ 1: DW_axi_dmac Channel-8 is enabled The bit 'DMAC_ChEnReg.CH8_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH8): DW_axi_dmac: Channel-8 is enabled ■ 0x0 (DISABLE_CH8): DW_axi_dmac: Channel-8 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true

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

Bits	Name	Memory Access	Description
6	CH7_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-7. ■ 0: DW_axi_dmac Channel-7 is disabled ■ 1: DW_axi_dmac Channel-7 is enabled The bit 'DMAC_ChEnReg.CH7_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH7): DW_axi_dmac: Channel-7 is enabled ■ 0x0 (DISABLE_CH7): DW_axi_dmac: Channel-7 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true
5	CH6_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-6. ■ 0: DW_axi_dmac Channel-6 is disabled ■ 1: DW_axi_dmac Channel-6 is enabled The bit 'DMAC_ChEnReg.CH6_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH6): DW_axi_dmac: Channel-6 is enabled ■ 0x0 (DISABLE_CH6): DW_axi_dmac: Channel-6 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true

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

Bits	Name	Memory Access	Description
4	CH5_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-5. ■ 0: DW_axi_dmac Channel-5 is disabled ■ 1: DW_axi_dmac Channel-5 is enabled The bit 'DMAC_ChEnReg.CH5_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH5): DW_axi_dmac: Channel-5 is enabled ■ 0x0 (DISABLE_CH5): DW_axi_dmac: Channel-5 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true
3	CH4_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-4. ■ 0: DW_axi_dmac Channel-4 is disabled ■ 1: DW_axi_dmac Channel-4 is enabled The bit 'DMAC_ChEnReg.CH4_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH4): DW_axi_dmac: Channel-4 is enabled ■ 0x0 (DISABLE_CH4): DW_axi_dmac: Channel-4 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true

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

Bits	Name	Memory Access	Description
2	CH3_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-3. ■ 0: DW_axi_dmac Channel-3 is disabled ■ 1: DW_axi_dmac Channel-3 is enabled The bit 'DMAC_ChEnReg.CH3_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH3): DW_axi_dmac: Channel-3 is enabled ■ 0x0 (DISABLE_CH3): DW_axi_dmac: Channel-3 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true
1	CH2_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-2. ■ 0: DW_axi_dmac Channel-2 is disabled ■ 1: DW_axi_dmac Channel-2 is enabled The bit 'DMAC_ChEnReg.CH2_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH2): DW_axi_dmac: Channel-2 is enabled ■ 0x0 (DISABLE_CH2): DW_axi_dmac: Channel-2 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true

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

Bits	Name	Memory Access	Description
0	CH1_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-1. ■ 0: DW_axi_dmac Channel-1 is disabled ■ 1: DW_axi_dmac Channel-1 is enabled The bit 'DMAC_ChEnReg.CH1_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH1): DW_axi_dmac: Channel-1 is enabled ■ 0x0 (DISABLE_CH1): DW_axi_dmac: Channel-1 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= x **Volatile:** true

5.1.5 DMAC_CHENREG2

- **Description:** This is DW_axi_dmac Channel Enable Register. If software wants to set up a new channel, it can read this register to find out which channels are currently inactive and then enable an inactive channel with the required priority.

  All bits of this register are cleared to 0 when the DW_axi_dmac Global Enable bit (DMAC_CfgReg.DMAC_EN) is 0. When DMAC_CfgReg.DMAC_EN is 0, a write to the DMAC_ChEnReg register is ignored and a read always reads back 0.

  The channel enable bit, DMAC_ChEnReg.CH_EN, is written only if the corresponding channel write enable bit, DMAC_ChEnReg.CH_EN_WE, is asserted on the same slave interface write transfer. For example, writing hex 0000_XXXX_0001_XXX1 writes a 1 into DMAC_ChEnReg [0], while DMAC_ChEnReg [15:1] and DMAC_ChEnReg [47:32] remains unchanged. . Writing hex 0000_XXXX_0000_XXXX leaves DMAC_ChEnReg [15:0] and DMAC_ChEnReg [47:32] unchanged.
- **Size:** 64 bits
- **Offset:** 0x18
- **Exists:** DMAX_NUM_CHANNELS > 8

Table 5-10 Fields for Register: DMAC_CHENREG2

Bits	Name	Memory Access	Description
63	CH32_EN_WE	W	DW_axi_dmac Channel-32 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH32_EN): Enable Write to CH32_EN bit ■ 0x0 (DISABLE_WR_CH32_EN): Disable Write to respective CH32_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
62	CH31_EN_WE	W	DW_axi_dmac Channel-31 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH31_EN): Enable Write to CH31_EN bit ■ 0x0 (DISABLE_WR_CH31_EN): Disable Write to respective CH31_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true
61	CH30_EN_WE	W	DW_axi_dmac Channel-30 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH30_EN): Enable Write to CH30_EN bit ■ 0x0 (DISABLE_WR_CH30_EN): Disable Write to respective CH30_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true
60	CH29_EN_WE	W	DW_axi_dmac Channel-29 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH29_EN): Enable Write to CH29_EN bit ■ 0x0 (DISABLE_WR_CH29_EN): Disable Write to respective CH29_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
59	CH28_EN_WE	W	DW_axi_dmac Channel-28 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH28_EN): Enable Write to CH28_EN bit ■ 0x0 (DISABLE_WR_CH28_EN): Disable Write to respective CH28_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true
58	CH27_EN_WE	W	DW_axi_dmac Channel-27 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH27_EN): Enable Write to CH27_EN bit ■ 0x0 (DISABLE_WR_CH27_EN): Disable Write to respective CH27_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true
57	CH26_EN_WE	W	DW_axi_dmac Channel-26 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH26_EN): Enable Write to CH26_EN bit ■ 0x0 (DISABLE_WR_CH26_EN): Disable Write to respective CH26_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
56	CH25_EN_WE	W	DW_axi_dmac Channel-25 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH25_EN): Enable Write to CH25_EN bit ■ 0x0 (DISABLE_WR_CH25_EN): Disable Write to respective CH25_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true
55	CH24_EN_WE	W	DW_axi_dmac Channel-24 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH24_EN): Enable Write to CH24_EN bit ■ 0x0 (DISABLE_WR_CH24_EN): Disable Write to respective CH24_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true
54	CH23_EN_WE	W	DW_axi_dmac Channel-23 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH23_EN): Enable Write to CH23_EN bit ■ 0x0 (DISABLE_WR_CH23_EN): Disable Write to respective CH23_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
53	CH22_EN_WE	W	DW_axi_dmac Channel-22 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH22_EN): Enable Write to CH22_EN bit ■ 0x0 (DISABLE_WR_CH22_EN): Disable Write to respective CH22_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true
52	CH21_EN_WE	W	DW_axi_dmac Channel-21 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH21_EN): Enable Write to CH21_EN bit ■ 0x0 (DISABLE_WR_CH21_EN): Disable Write to respective CH21_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true
51	CH20_EN_WE	W	DW_axi_dmac Channel-20 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH20_EN): Enable Write to CH20_EN bit ■ 0x0 (DISABLE_WR_CH20_EN): Disable Write to respective CH20_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
50	CH19_EN_WE	W	DW_axi_dmac Channel-19 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH19_EN): Enable Write to CH19_EN bit ■ 0x0 (DISABLE_WR_CH19_EN): Disable Write to respective CH19_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true
49	CH18_EN_WE	W	DW_axi_dmac Channel-18 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH18_EN): Enable Write to CH18_EN bit ■ 0x0 (DISABLE_WR_CH18_EN): Disable Write to respective CH18_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true
48	CH17_EN_WE	W	DW_axi_dmac Channel-17 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH17_EN): Enable Write to CH17_EN bit ■ 0x0 (DISABLE_WR_CH17_EN): Disable Write to respective CH17_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
47	CH32_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-32. ■ 0: DW_axi_dmac Channel-32 is disabled ■ 1: DW_axi_dmac Channel-32 is enabled The bit 'DMAC_ChEnReg.CH32_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH32): DW_axi_dmac: Channel-32 is enabled ■ 0x0 (DISABLE_CH32): DW_axi_dmac: Channel-32 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true
46	CH31_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-31. ■ 0: DW_axi_dmac Channel-31 is disabled ■ 1: DW_axi_dmac Channel-31 is enabled The bit 'DMAC_ChEnReg.CH31_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH31): DW_axi_dmac: Channel-31 is enabled ■ 0x0 (DISABLE_CH31): DW_axi_dmac: Channel-31 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
45	CH30_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-30. ■ 0: DW_axi_dmac Channel-30 is disabled ■ 1: DW_axi_dmac Channel-30 is enabled The bit 'DMAC_ChEnReg.CH30_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH30): DW_axi_dmac: Channel-30 is enabled ■ 0x0 (DISABLE_CH30): DW_axi_dmac: Channel-30 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true
44	CH29_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-29. ■ 0: DW_axi_dmac Channel-29 is disabled ■ 1: DW_axi_dmac Channel-29 is enabled The bit 'DMAC_ChEnReg.CH29_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH29): DW_axi_dmac: Channel-29 is enabled ■ 0x0 (DISABLE_CH29): DW_axi_dmac: Channel-29 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
43	CH28_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-28. ■ 0: DW_axi_dmac Channel-28 is disabled ■ 1: DW_axi_dmac Channel-28 is enabled The bit 'DMAC_ChEnReg.CH28_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH28): DW_axi_dmac: Channel-28 is enabled ■ 0x0 (DISABLE_CH28): DW_axi_dmac: Channel-28 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true
42	CH27_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-27. ■ 0: DW_axi_dmac Channel-27 is disabled ■ 1: DW_axi_dmac Channel-27 is enabled The bit 'DMAC_ChEnReg.CH27_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH27): DW_axi_dmac: Channel-27 is enabled ■ 0x0 (DISABLE_CH27): DW_axi_dmac: Channel-27 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
41	CH26_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-26. ■ 0: DW_axi_dmac Channel-26 is disabled ■ 1: DW_axi_dmac Channel-26 is enabled The bit 'DMAC_ChEnReg.CH26_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH26): DW_axi_dmac: Channel-26 is enabled ■ 0x0 (DISABLE_CH26): DW_axi_dmac: Channel-26 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true
40	CH25_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-25. ■ 0: DW_axi_dmac Channel-25 is disabled ■ 1: DW_axi_dmac Channel-25 is enabled The bit 'DMAC_ChEnReg.CH25_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH25): DW_axi_dmac: Channel-25 is enabled ■ 0x0 (DISABLE_CH25): DW_axi_dmac: Channel-25 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
39	CH24_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-24. ■ 0: DW_axi_dmac Channel-24 is disabled ■ 1: DW_axi_dmac Channel-24 is enabled The bit 'DMAC_ChEnReg.CH24_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH24): DW_axi_dmac: Channel-24 is enabled ■ 0x0 (DISABLE_CH24): DW_axi_dmac: Channel-24 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true
38	CH23_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-23. ■ 0: DW_axi_dmac Channel-23 is disabled ■ 1: DW_axi_dmac Channel-23 is enabled The bit 'DMAC_ChEnReg.CH23_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH23): DW_axi_dmac: Channel-23 is enabled ■ 0x0 (DISABLE_CH23): DW_axi_dmac: Channel-23 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
37	CH22_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-22. ■ 0: DW_axi_dmac Channel-22 is disabled ■ 1: DW_axi_dmac Channel-22 is enabled The bit 'DMAC_ChEnReg.CH22_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH22): DW_axi_dmac: Channel-22 is enabled ■ 0x0 (DISABLE_CH22): DW_axi_dmac: Channel-22 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true
36	CH21_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-21. ■ 0: DW_axi_dmac Channel-21 is disabled ■ 1: DW_axi_dmac Channel-21 is enabled The bit 'DMAC_ChEnReg.CH21_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH21): DW_axi_dmac: Channel-21 is enabled ■ 0x0 (DISABLE_CH21): DW_axi_dmac: Channel-21 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
35	CH20_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-20. ■ 0: DW_axi_dmac Channel-20 is disabled ■ 1: DW_axi_dmac Channel-20 is enabled The bit 'DMAC_ChEnReg.CH20_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH20): DW_axi_dmac: Channel-20 is enabled ■ 0x0 (DISABLE_CH20): DW_axi_dmac: Channel-20 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true
34	CH19_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-19. ■ 0: DW_axi_dmac Channel-19 is disabled ■ 1: DW_axi_dmac Channel-19 is enabled The bit 'DMAC_ChEnReg.CH19_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH19): DW_axi_dmac: Channel-19 is enabled ■ 0x0 (DISABLE_CH19): DW_axi_dmac: Channel-19 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
33	CH18_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-18. ■ 0: DW_axi_dmac Channel-18 is disabled ■ 1: DW_axi_dmac Channel-18 is enabled The bit 'DMAC_ChEnReg.CH18_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH18): DW_axi_dmac: Channel-18 is enabled ■ 0x0 (DISABLE_CH18): DW_axi_dmac: Channel-18 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true
32	CH17_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-17. ■ 0: DW_axi_dmac Channel-17 is disabled ■ 1: DW_axi_dmac Channel-17 is enabled The bit 'DMAC_ChEnReg.CH17_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH17): DW_axi_dmac: Channel-17 is enabled ■ 0x0 (DISABLE_CH17): DW_axi_dmac: Channel-17 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
31	CH16_EN_WE	W	DW_axi_dmac Channel-16 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH16_EN): Enable Write to CH16_EN bit ■ 0x0 (DISABLE_WR_CH16_EN): Disable Write to respective CH16_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true
30	CH15_EN_WE	W	DW_axi_dmac Channel-15 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH15_EN): Enable Write to CH15_EN bit ■ 0x0 (DISABLE_WR_CH15_EN): Disable Write to respective CH15_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true
29	CH14_EN_WE	W	DW_axi_dmac Channel-14 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH14_EN): Enable Write to CH14_EN bit ■ 0x0 (DISABLE_WR_CH14_EN): Disable Write to respective CH14_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
28	CH13_EN_WE	W	DW_axi_dmac Channel-13 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH13_EN): Enable Write to CH13_EN bit ■ 0x0 (DISABLE_WR_CH13_EN): Disable Write to respective CH13_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true
27	CH12_EN_WE	W	DW_axi_dmac Channel-12 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH12_EN): Enable Write to CH12_EN bit ■ 0x0 (DISABLE_WR_CH12_EN): Disable Write to respective CH12_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true
26	CH11_EN_WE	W	DW_axi_dmac Channel-11 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH11_EN): Enable Write to CH11_EN bit ■ 0x0 (DISABLE_WR_CH11_EN): Disable Write to respective CH11_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
25	CH10_EN_WE	W	DW_axi_dmac Channel-10 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH10_EN): Enable Write to CH10_EN bit ■ 0x0 (DISABLE_WR_CH10_EN): Disable Write to respective CH10_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true
24	CH9_EN_WE	W	DW_axi_dmac Channel-9 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH9_EN): Enable Write to CH9_EN bit ■ 0x0 (DISABLE_WR_CH9_EN): Disable Write to respective CH9_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true
23	CH8_EN_WE	W	DW_axi_dmac Channel-8 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH8_EN): Enable Write to CH8_EN bit ■ 0x0 (DISABLE_WR_CH8_EN): Disable Write to respective CH8_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
22	CH7_EN_WE	W	DW_axi_dmac Channel-7 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH7_EN): Enable Write to CH7_EN bit ■ 0x0 (DISABLE_WR_CH7_EN): Disable Write to respective CH7_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true
21	CH6_EN_WE	W	DW_axi_dmac Channel-6 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH6_EN): Enable Write to CH6_EN bit ■ 0x0 (DISABLE_WR_CH6_EN): Disable Write to respective CH6_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true
20	CH5_EN_WE	W	DW_axi_dmac Channel-5 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH5_EN): Enable Write to CH5_EN bit ■ 0x0 (DISABLE_WR_CH5_EN): Disable Write to respective CH5_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
19	CH4_EN_WE	W	DW_axi_dmac Channel-4 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH4_EN): Enable Write to CH4_EN bit ■ 0x0 (DISABLE_WR_CH4_EN): Disable Write to respective CH4_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true
18	CH3_EN_WE	W	DW_axi_dmac Channel-3 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH3_EN): Enable Write to CH3_EN bit ■ 0x0 (DISABLE_WR_CH3_EN): Disable Write to respective CH3_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true
17	CH2_EN_WE	W	DW_axi_dmac Channel-2 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH2_EN): Enable Write to CH2_EN bit ■ 0x0 (DISABLE_WR_CH2_EN): Disable Write to respective CH2_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
16	CH1_EN_WE	W	DW_axi_dmac Channel-1 Enable Write Enable bit. Read back value of this register bit is always '0'. **Values:** ■ 0x1 (ENABLE_WR_CH1_EN): Enable Write to CH1_EN bit ■ 0x0 (DISABLE_WR_CH1_EN): Disable Write to respective CH1_EN bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true
15	CH16_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-16. ■ 0: DW_axi_dmac Channel-16 is disabled ■ 1: DW_axi_dmac Channel-16 is enabled The bit 'DMAC_ChEnReg.CH16_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH16): DW_axi_dmac: Channel-16 is enabled ■ 0x0 (DISABLE_CH16): DW_axi_dmac: Channel-16 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
14	CH15_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-15. ■ 0: DW_axi_dmac Channel-15 is disabled ■ 1: DW_axi_dmac Channel-15 is enabled The bit 'DMAC_ChEnReg.CH15_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH15): DW_axi_dmac: Channel-15 is enabled ■ 0x0 (DISABLE_CH15): DW_axi_dmac: Channel-15 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true
13	CH14_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-14. ■ 0: DW_axi_dmac Channel-14 is disabled ■ 1: DW_axi_dmac Channel-14 is enabled The bit 'DMAC_ChEnReg.CH14_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH14): DW_axi_dmac: Channel-14 is enabled ■ 0x0 (DISABLE_CH14): DW_axi_dmac: Channel-14 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
12	CH13_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-13. ■ 0: DW_axi_dmac Channel-13 is disabled ■ 1: DW_axi_dmac Channel-13 is enabled The bit 'DMAC_ChEnReg.CH13_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH13): DW_axi_dmac: Channel-13 is enabled ■ 0x0 (DISABLE_CH13): DW_axi_dmac: Channel-13 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true
11	CH12_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-12. ■ 0: DW_axi_dmac Channel-12 is disabled ■ 1: DW_axi_dmac Channel-12 is enabled The bit 'DMAC_ChEnReg.CH12_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH12): DW_axi_dmac: Channel-12 is enabled ■ 0x0 (DISABLE_CH12): DW_axi_dmac: Channel-12 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
10	CH11_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-11. ■ 0: DW_axi_dmac Channel-11 is disabled ■ 1: DW_axi_dmac Channel-11 is enabled The bit 'DMAC_ChEnReg.CH11_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH11): DW_axi_dmac: Channel-11 is enabled ■ 0x0 (DISABLE_CH11): DW_axi_dmac: Channel-11 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true
9	CH10_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-10. ■ 0: DW_axi_dmac Channel-10 is disabled ■ 1: DW_axi_dmac Channel-10 is enabled The bit 'DMAC_ChEnReg.CH10_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH10): DW_axi_dmac: Channel-10 is enabled ■ 0x0 (DISABLE_CH10): DW_axi_dmac: Channel-10 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
8	CH9_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-9. ■ 0: DW_axi_dmac Channel-9 is disabled ■ 1: DW_axi_dmac Channel-9 is enabled The bit 'DMAC_ChEnReg.CH9_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH9): DW_axi_dmac: Channel-9 is enabled ■ 0x0 (DISABLE_CH9): DW_axi_dmac: Channel-9 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true
7	CH8_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-8. ■ 0: DW_axi_dmac Channel-8 is disabled ■ 1: DW_axi_dmac Channel-8 is enabled The bit 'DMAC_ChEnReg.CH8_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH8): DW_axi_dmac: Channel-8 is enabled ■ 0x0 (DISABLE_CH8): DW_axi_dmac: Channel-8 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
6	CH7_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-7. ■ 0: DW_axi_dmac Channel-7 is disabled ■ 1: DW_axi_dmac Channel-7 is enabled The bit 'DMAC_ChEnReg.CH7_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH7): DW_axi_dmac: Channel-7 is enabled ■ 0x0 (DISABLE_CH7): DW_axi_dmac: Channel-7 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true
5	CH6_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-6. ■ 0: DW_axi_dmac Channel-6 is disabled ■ 1: DW_axi_dmac Channel-6 is enabled The bit 'DMAC_ChEnReg.CH6_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH6): DW_axi_dmac: Channel-6 is enabled ■ 0x0 (DISABLE_CH6): DW_axi_dmac: Channel-6 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
4	CH5_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-5. ■ 0: DW_axi_dmac Channel-5 is disabled ■ 1: DW_axi_dmac Channel-5 is enabled The bit 'DMAC_ChEnReg.CH5_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH5): DW_axi_dmac: Channel-5 is enabled ■ 0x0 (DISABLE_CH5): DW_axi_dmac: Channel-5 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true
3	CH4_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-4. ■ 0: DW_axi_dmac Channel-4 is disabled ■ 1: DW_axi_dmac Channel-4 is enabled The bit 'DMAC_ChEnReg.CH4_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH4): DW_axi_dmac: Channel-4 is enabled ■ 0x0 (DISABLE_CH4): DW_axi_dmac: Channel-4 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
2	CH3_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-3. ■ 0: DW_axi_dmac Channel-3 is disabled ■ 1: DW_axi_dmac Channel-3 is enabled The bit 'DMAC_ChEnReg.CH3_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH3): DW_axi_dmac: Channel-3 is enabled ■ 0x0 (DISABLE_CH3): DW_axi_dmac: Channel-3 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true
1	CH2_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-2. ■ 0: DW_axi_dmac Channel-2 is disabled ■ 1: DW_axi_dmac Channel-2 is enabled The bit 'DMAC_ChEnReg.CH2_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH2): DW_axi_dmac: Channel-2 is enabled ■ 0x0 (DISABLE_CH2): DW_axi_dmac: Channel-2 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true

Table 5-10 Fields for Register: DMAC_CHENREG2 (Continued)

Bits	Name	Memory Access	Description
0	CH1_EN	R/W	This bit is used to enable the DW_axi_dmac Channel-1. ■ 0: DW_axi_dmac Channel-1 is disabled ■ 1: DW_axi_dmac Channel-1 is enabled The bit 'DMAC_ChEnReg.CH1_EN' is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer. **Values:** ■ 0x1 (ENABLE_CH1): DW_axi_dmac: Channel-1 is enabled ■ 0x0 (DISABLE_CH1): DW_axi_dmac: Channel-1 is disabled **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true

5.1.6 DMAC_CHSUSPREG

- **Description:** This is DW_axi_dmac Channel Suspend Register. The channel suspend bit, DMAC_ChSuspReg.CH_SUSP, is written only if the corresponding channel write enable bit, DMAC_ChSuspReg.CH_SUSP_WE, is asserted on the same slave interface write transfer. For example, writing hex 0000_XXXX_0001_XXX1 writes a 1 into DMAC_ChSuspReg [0], while DMAC_ChSuspReg [15:1] and DMAC_ChSuspReg [47:32] remains unchanged. Writing hex 0000_XXXX_0000_XXXX leaves DMAC_ChSuspReg [15:0] and DMAC_ChSuspReg [47:32] unchanged.
- **Size:** 64 bits
- **Offset:** 0x20
- **Exists:** DMAX_NUM_CHANNELS > 8

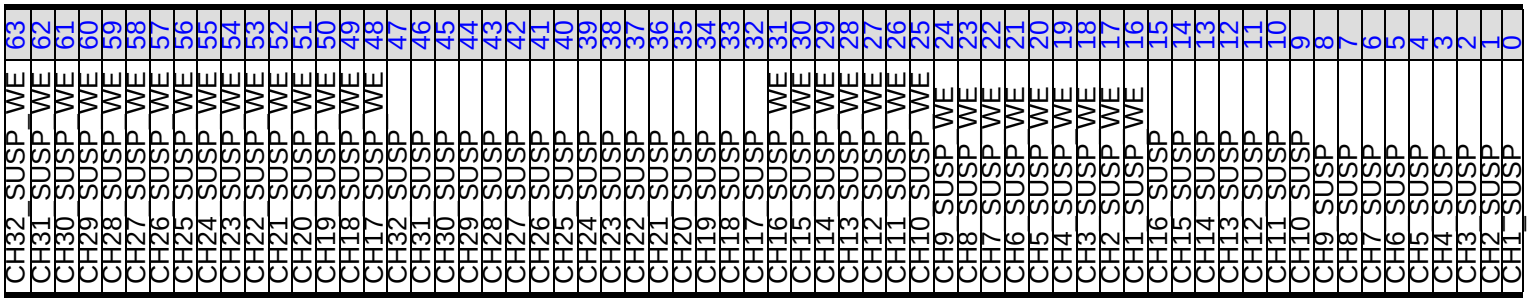


Table 5-11 Fields for Register: DMAC_CHSUSPREG

Bits	Name	Memory Access	Description
63	CH32_SUSP_WE	W	This bit is used as a write enable to the Channel-32 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH32_SUSP): Enable Write to respective CH32_SUSP bit ■ 0x0 (DISABLE_WR_CH32_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true

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

Bits	Name	Memory Access	Description
62	CH31_SUSP_WE	W	This bit is used as a write enable to the Channel-31 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH31_SUSP): Enable Write to respective CH31_SUSP bit ■ 0x0 (DISABLE_WR_CH31_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true
61	CH30_SUSP_WE	W	This bit is used as a write enable to the Channel-30 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH30_SUSP): Enable Write to respective CH30_SUSP bit ■ 0x0 (DISABLE_WR_CH30_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true
60	CH29_SUSP_WE	W	This bit is used as a write enable to the Channel-29 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH29_SUSP): Enable Write to respective CH29_SUSP bit ■ 0x0 (DISABLE_WR_CH29_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true
59	CH28_SUSP_WE	W	This bit is used as a write enable to the Channel-28 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH28_SUSP): Enable Write to respective CH28_SUSP bit ■ 0x0 (DISABLE_WR_CH28_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true

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

Bits	Name	Memory Access	Description
58	CH27_SUSP_WE	W	This bit is used as a write enable to the Channel-27 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH27_SUSP): Enable Write to respective CH27_SUSP bit ■ 0x0 (DISABLE_WR_CH27_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true
57	CH26_SUSP_WE	W	This bit is used as a write enable to the Channel-26 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH26_SUSP): Enable Write to respective CH26_SUSP bit ■ 0x0 (DISABLE_WR_CH26_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true
56	CH25_SUSP_WE	W	This bit is used as a write enable to the Channel-25 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH25_SUSP): Enable Write to respective CH25_SUSP bit ■ 0x0 (DISABLE_WR_CH25_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true
55	CH24_SUSP_WE	W	This bit is used as a write enable to the Channel-24 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH24_SUSP): Enable Write to respective CH24_SUSP bit ■ 0x0 (DISABLE_WR_CH24_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true

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

Bits	Name	Memory Access	Description
54	CH23_SUSP_WE	W	This bit is used as a write enable to the Channel-23 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH23_SUSP): Enable Write to respective CH23_SUSP bit ■ 0x0 (DISABLE_WR_CH23_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true
53	CH22_SUSP_WE	W	This bit is used as a write enable to the Channel-22 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH22_SUSP): Enable Write to respective CH22_SUSP bit ■ 0x0 (DISABLE_WR_CH22_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true
52	CH21_SUSP_WE	W	This bit is used as a write enable to the Channel-21 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH21_SUSP): Enable Write to respective CH21_SUSP bit ■ 0x0 (DISABLE_WR_CH21_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true
51	CH20_SUSP_WE	W	This bit is used as a write enable to the Channel-20 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH20_SUSP): Enable Write to respective CH20_SUSP bit ■ 0x0 (DISABLE_WR_CH20_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true

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

Bits	Name	Memory Access	Description
50	CH19_SUSP_WE	W	This bit is used as a write enable to the Channel-19 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH19_SUSP): Enable Write to respective CH19_SUSP bit ■ 0x0 (DISABLE_WR_CH19_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true
49	CH18_SUSP_WE	W	This bit is used as a write enable to the Channel-18 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH18_SUSP): Enable Write to respective CH18_SUSP bit ■ 0x0 (DISABLE_WR_CH18_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true
48	CH17_SUSP_WE	W	This bit is used as a write enable to the Channel-17 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH17_SUSP): Enable Write to respective CH17_SUSP bit ■ 0x0 (DISABLE_WR_CH17_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true

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

Bits	Name	Memory Access	Description
47	CH32_SUSP	R/W	Channel-32 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH32_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH32_SUSP bit to 1 and polls CH32_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH32_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH32_SUSP bit to 0, after DW_axi_dmac sets CH32_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH32_SUSP): Request to Suspended Channel-32 ■ 0x0 (DISABLE_CH32_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true

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

Bits	Name	Memory Access	Description
46	CH31_SUSP	R/W	Channel-31 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH31_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH31_SUSP bit to 1 and polls CH31_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH31_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH31_SUSP bit to 0, after DW_axi_dmac sets CH31_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH31_SUSP): Request to Suspended Channel-31 ■ 0x0 (DISABLE_CH31_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true

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

Bits	Name	Memory Access	Description
45	CH30_SUSP	R/W	Channel-30 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH30_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH30_SUSP bit to 1 and polls CH30_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH30_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH30_SUSP bit to 0, after DW_axi_dmac sets CH30_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH30_SUSP): Request to Suspended Channel-30 ■ 0x0 (DISABLE_CH30_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true

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

Bits	Name	Memory Access	Description
44	CH29_SUSP	R/W	Channel-29 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH29_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH29_SUSP bit to 1 and polls CH29_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH29_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH29_SUSP bit to 0, after DW_axi_dmac sets CH29_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH29_SUSP): Request to Suspended Channel-29 ■ 0x0 (DISABLE_CH29_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true

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

Bits	Name	Memory Access	Description
43	CH28_SUSP	R/W	Channel-28 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH28_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH28_SUSP bit to 1 and polls CH28_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH28_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH28_SUSP bit to 0, after DW_axi_dmac sets CH28_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH28_SUSP): Request to Suspended Channel-28 ■ 0x0 (DISABLE_CH28_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true

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

Bits	Name	Memory Access	Description
42	CH27_SUSP	R/W	Channel-27 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH27_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH27_SUSP bit to 1 and polls CH27_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH27_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH27_SUSP bit to 0, after DW_axi_dmac sets CH27_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH27_SUSP): Request to Suspended Channel-27 ■ 0x0 (DISABLE_CH27_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true

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

Bits	Name	Memory Access	Description
41	CH26_SUSP	R/W	Channel-26 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH26_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH26_SUSP bit to 1 and polls CH26_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH26_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH26_SUSP bit to 0, after DW_axi_dmac sets CH26_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH26_SUSP): Request to Suspended Channel-26 ■ 0x0 (DISABLE_CH26_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true

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

Bits	Name	Memory Access	Description
40	CH25_SUSP	R/W	Channel-25 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH25_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH25_SUSP bit to 1 and polls CH25_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH25_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH25_SUSP bit to 0, after DW_axi_dmac sets CH25_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH25_SUSP): Request to Suspended Channel-25 ■ 0x0 (DISABLE_CH25_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true

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

Bits	Name	Memory Access	Description
39	CH24_SUSP	R/W	Channel-24 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH24_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH24_SUSP bit to 1 and polls CH24_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH24_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH24_SUSP bit to 0, after DW_axi_dmac sets CH24_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH24_SUSP): Request to Suspended Channel-24 ■ 0x0 (DISABLE_CH24_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true

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

Bits	Name	Memory Access	Description
38	CH23_SUSP	R/W	Channel-23 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH23_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH23_SUSP bit to 1 and polls CH23_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH23_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH23_SUSP bit to 0, after DW_axi_dmac sets CH23_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH23_SUSP): Request to Suspended Channel-23 ■ 0x0 (DISABLE_CH23_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true

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

Bits	Name	Memory Access	Description
37	CH22_SUSP	R/W	Channel-22 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH22_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH22_SUSP bit to 1 and polls CH22_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH22_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH22_SUSP bit to 0, after DW_axi_dmac sets CH22_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH22_SUSP): Request to Suspended Channel-22 ■ 0x0 (DISABLE_CH22_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true

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

Bits	Name	Memory Access	Description
36	CH21_SUSP	R/W	Channel-21 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH21_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH21_SUSP bit to 1 and polls CH21_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH21_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH21_SUSP bit to 0, after DW_axi_dmac sets CH21_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH21_SUSP): Request to Suspended Channel-21 ■ 0x0 (DISABLE_CH21_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true

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

Bits	Name	Memory Access	Description
35	CH20_SUSP	R/W	Channel-20 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH20_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH20_SUSP bit to 1 and polls CH20_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH20_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH20_SUSP bit to 0, after DW_axi_dmac sets CH20_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH20_SUSP): Request to Suspended Channel-20 ■ 0x0 (DISABLE_CH20_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true

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

Bits	Name	Memory Access	Description
34	CH19_SUSP	R/W	Channel-19 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH19_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH19_SUSP bit to 1 and polls CH19_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH19_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH19_SUSP bit to 0, after DW_axi_dmac sets CH19_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH19_SUSP): Request to Suspended Channel-19 ■ 0x0 (DISABLE_CH19_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true

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

Bits	Name	Memory Access	Description
33	CH18_SUSP	R/W	Channel-18 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH18_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH18_SUSP bit to 1 and polls CH18_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH18_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH18_SUSP bit to 0, after DW_axi_dmac sets CH18_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH18_SUSP): Request to Suspended Channel-18 ■ 0x0 (DISABLE_CH18_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true

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

Bits	Name	Memory Access	Description
32	CH17_SUSP	R/W	Channel-17 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH17_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH17_SUSP bit to 1 and polls CH17_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH17_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH17_SUSP bit to 0, after DW_axi_dmac sets CH17_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH17_SUSP): Request to Suspended Channel-17 ■ 0x0 (DISABLE_CH17_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true
31	CH16_SUSP_WE	W	This bit is used as a write enable to the Channel-16 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH16_SUSP): Enable Write to respective CH16_SUSP bit ■ 0x0 (DISABLE_WR_CH16_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true

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

Bits	Name	Memory Access	Description
30	CH15_SUSP_WE	W	This bit is used as a write enable to the Channel-15 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH15_SUSP): Enable Write to respective CH15_SUSP bit ■ 0x0 (DISABLE_WR_CH15_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true
29	CH14_SUSP_WE	W	This bit is used as a write enable to the Channel-14 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH14_SUSP): Enable Write to respective CH14_SUSP bit ■ 0x0 (DISABLE_WR_CH14_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true
28	CH13_SUSP_WE	W	This bit is used as a write enable to the Channel-13 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH13_SUSP): Enable Write to respective CH13_SUSP bit ■ 0x0 (DISABLE_WR_CH13_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true
27	CH12_SUSP_WE	W	This bit is used as a write enable to the Channel-12 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH12_SUSP): Enable Write to respective CH12_SUSP bit ■ 0x0 (DISABLE_WR_CH12_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true

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

Bits	Name	Memory Access	Description
26	CH11_SUSP_WE	W	This bit is used as a write enable to the Channel-11 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH11_SUSP): Enable Write to respective CH11_SUSP bit ■ 0x0 (DISABLE_WR_CH11_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true
25	CH10_SUSP_WE	W	This bit is used as a write enable to the Channel-10 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH10_SUSP): Enable Write to respective CH10_SUSP bit ■ 0x0 (DISABLE_WR_CH10_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true
24	CH9_SUSP_WE	W	This bit is used as a write enable to the Channel-9 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH9_SUSP): Enable Write to respective CH9_SUSP bit ■ 0x0 (DISABLE_WR_CH9_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true
23	CH8_SUSP_WE	W	This bit is used as a write enable to the Channel-8 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH8_SUSP): Enable Write to respective CH8_SUSP bit ■ 0x0 (DISABLE_WR_CH8_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

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

Bits	Name	Memory Access	Description
22	CH7_SUSP_WE	W	This bit is used as a write enable to the Channel-7 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH7_SUSP): Enable Write to respective CH7_SUSP bit ■ 0x0 (DISABLE_WR_CH7_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true
21	CH6_SUSP_WE	W	This bit is used as a write enable to the Channel-6 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH6_SUSP): Enable Write to respective CH6_SUSP bit ■ 0x0 (DISABLE_WR_CH6_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true
20	CH5_SUSP_WE	W	This bit is used as a write enable to the Channel-5 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH5_SUSP): Enable Write to respective CH5_SUSP bit ■ 0x0 (DISABLE_WR_CH5_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true
19	CH4_SUSP_WE	W	This bit is used as a write enable to the Channel-4 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH4_SUSP): Enable Write to respective CH4_SUSP bit ■ 0x0 (DISABLE_WR_CH4_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true

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

Bits	Name	Memory Access	Description
18	CH3_SUSP_WE	W	This bit is used as a write enable to the Channel-3 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH3_SUSP): Enable Write to respective CH3_SUSP bit ■ 0x0 (DISABLE_WR_CH3_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true
17	CH2_SUSP_WE	W	This bit is used as a write enable to the Channel-2 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH2_SUSP): Enable Write to respective CH2_SUSP bit ■ 0x0 (DISABLE_WR_CH2_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true
16	CH1_SUSP_WE	W	This bit is used as a write enable to the Channel-1 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH1_SUSP): Enable Write to respective CH1_SUSP bit ■ 0x0 (DISABLE_WR_CH1_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true

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

Bits	Name	Memory Access	Description
15	CH16_SUSP	R/W	Channel-16 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH16_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH16_SUSP bit to 1 and polls CH16_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH16_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH16_SUSP bit to 0, after DW_axi_dmac sets CH16_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH16_SUSP): Request to Suspended Channel-16 ■ 0x0 (DISABLE_CH16_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true

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

Bits	Name	Memory Access	Description
14	CH15_SUSP	R/W	Channel-15 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH15_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH15_SUSP bit to 1 and polls CH15_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH15_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH15_SUSP bit to 0, after DW_axi_dmac sets CH15_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH15_SUSP): Request to Suspended Channel-15 ■ 0x0 (DISABLE_CH15_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true

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

Bits	Name	Memory Access	Description
13	CH14_SUSP	R/W	Channel-14 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH14_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH14_SUSP bit to 1 and polls CH14_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH14_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH14_SUSP bit to 0, after DW_axi_dmac sets CH14_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH14_SUSP): Request to Suspended Channel-14 ■ 0x0 (DISABLE_CH14_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true

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

Bits	Name	Memory Access	Description
12	CH13_SUSP	R/W	Channel-13 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH13_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH13_SUSP bit to 1 and polls CH13_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH13_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH13_SUSP bit to 0, after DW_axi_dmac sets CH13_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH13_SUSP): Request to Suspended Channel-13 ■ 0x0 (DISABLE_CH13_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true

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

Bits	Name	Memory Access	Description
11	CH12_SUSP	R/W	Channel-12 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH12_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH12_SUSP bit to 1 and polls CH12_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH12_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH12_SUSP bit to 0, after DW_axi_dmac sets CH12_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH12_SUSP): Request to Suspended Channel-12 ■ 0x0 (DISABLE_CH12_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true

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

Bits	Name	Memory Access	Description
10	CH11_SUSP	R/W	Channel-11 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH11_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH11_SUSP bit to 1 and polls CH11_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH11_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH11_SUSP bit to 0, after DW_axi_dmac sets CH11_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH11_SUSP): Request to Suspended Channel-11 ■ 0x0 (DISABLE_CH11_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true

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

Bits	Name	Memory Access	Description
9	CH10_SUSP	R/W	Channel-10 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH10_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH10_SUSP bit to 1 and polls CH10_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH10_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH10_SUSP bit to 0, after DW_axi_dmac sets CH10_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH10_SUSP): Request to Suspended Channel-10 ■ 0x0 (DISABLE_CH10_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true

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

Bits	Name	Memory Access	Description
8	CH9_SUSP	R/W	Channel-9 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH9_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH9_SUSP bit to 1 and polls CH9_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH9_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH9_SUSP bit to 0, after DW_axi_dmac sets CH9_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH9_SUSP): Request to Suspended Channel-9 ■ 0x0 (DISABLE_CH9_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true

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

Bits	Name	Memory Access	Description
7	CH8_SUSP	R/W	Channel-8 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH8_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH8_SUSP bit to 1 and polls CH8_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH8_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH8_SUSP bit to 0, after DW_axi_dmac sets CH8_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH8_SUSP): Request to Suspended Channel-8 ■ 0x0 (DISABLE_CH8_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true

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

Bits	Name	Memory Access	Description
6	CH7_SUSP	R/W	Channel-7 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH7_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH7_SUSP bit to 1 and polls CH7_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH7_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH7_SUSP bit to 0, after DW_axi_dmac sets CH7_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH7_SUSP): Request to Suspended Channel-7 ■ 0x0 (DISABLE_CH7_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true

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

Bits	Name	Memory Access	Description
5	CH6_SUSP	R/W	Channel-6 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH6_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH6_SUSP bit to 1 and polls CH6_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH6_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH6_SUSP bit to 0, after DW_axi_dmac sets CH6_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH6_SUSP): Request to Suspended Channel-6 ■ 0x0 (DISABLE_CH6_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true

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

Bits	Name	Memory Access	Description
4	CH5_SUSP	R/W	Channel-5 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH5_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH5_SUSP bit to 1 and polls CH5_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH5_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH5_SUSP bit to 0, after DW_axi_dmac sets CH5_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH5_SUSP): Request to Suspended Channel-5 ■ 0x0 (DISABLE_CH5_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true

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

Bits	Name	Memory Access	Description
3	CH4_SUSP	R/W	Channel-4 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH4_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH4_SUSP bit to 1 and polls CH4_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH4_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH4_SUSP bit to 0, after DW_axi_dmac sets CH4_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH4_SUSP): Request to Suspended Channel-4 ■ 0x0 (DISABLE_CH4_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true

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

Bits	Name	Memory Access	Description
2	CH3_SUSP	R/W	Channel-3 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH3_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH3_SUSP bit to 1 and polls CH3_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH3_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH3_SUSP bit to 0, after DW_axi_dmac sets CH3_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH3_SUSP): Request to Suspended Channel-3 ■ 0x0 (DISABLE_CH3_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true

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

Bits	Name	Memory Access	Description
1	CH2_SUSP	R/W	Channel-2 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH2_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH2_SUSP bit to 1 and polls CH2_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH2_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH2_SUSP bit to 0, after DW_axi_dmac sets CH2_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH2_SUSP): Request to Suspended Channel-2 ■ 0x0 (DISABLE_CH2_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true

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

Bits	Name	Memory Access	Description
0	CH1_SUSP	R/W	Channel-1 Suspend Request. Software sets this bit to 1 to request channel suspend. If this bit is set to 1, DW_axi_dmac suspends all DMA data transfers from the source gracefully until this bit is cleared. There is no guarantee that the current dma transaction will complete. This bit can also be used in conjunction with CH1_Status.CH_SUSPENDED to cleanly disable the channel without losing any data. In this case, software first sets CH1_SUSP bit to 1 and polls CH1_Status.CH_SUSPENDED till it is set to 1. Software can then clear CH1_EN bit to 0 to disable the channel. ■ 0: No Channel Suspend Request. ■ 1: Request for Channel Suspend. Software can clear CH1_SUSP bit to 0, after DW_axi_dmac sets CH1_Status.CH_SUSPENDED bit to 1, to exit the channel suspend mode. **Note**: CH_SUSP is cleared when channel is disabled. **Values:** ■ 0x1 (ENABLE_CH1_SUSP): Request to Suspended Channel-1 ■ 0x0 (DISABLE_CH1_SUSP): No Channel Suspend Request **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true

5.1.7 DMAC_CHABORTREG

- **Description:** This is DW_axi_dmac Channel Abort Register. The channel abort bit, DMAC_ChAbortReg.CH_ABORT, is written only if the corresponding channel write enable bit, DMAC_ChAbortReg.CH_ABORT_WE, is asserted on the same slave interface write transfer.
- **Size:** 64 bits
- **Offset:** 0x28
- **Exists:** DMAX_NUM_CHANNELS > 8

Table 5-12 Fields for Register: DMAC_CHABORTREG

Bits	Name	Memory Access	Description
63	CH32_ABORT_WE	* Varies	This bit is used to write enable the Channel-32 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH32_ABORT): Enable Write to CH32_ABORT bit ■ 0x0 (DISABLE_WR_CH32_ABORT): Disable Write to CH32_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
62	CH31_ABORT_WE	* Varies	This bit is used to write enable the Channel-31 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH31_ABORT): Enable Write to CH31_ABORT bit ■ 0x0 (DISABLE_WR_CH31_ABORT): Disable Write to CH31_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
61	CH30_ABORT_WE	* Varies	This bit is used to write enable the Channel-30 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH30_ABORT): Enable Write to CH30_ABORT bit ■ 0x0 (DISABLE_WR_CH30_ABORT): Disable Write to CH30_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
60	CH29_ABORT_WE	* Varies	This bit is used to write enable the Channel-29 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH29_ABORT): Enable Write to CH29_ABORT bit ■ 0x0 (DISABLE_WR_CH29_ABORT): Disable Write to CH29_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
59	CH28_ABORT_WE	* Varies	This bit is used to write enable the Channel-28 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH28_ABORT): Enable Write to CH28_ABORT bit ■ 0x0 (DISABLE_WR_CH28_ABORT): Disable Write to CH28_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
58	CH27_ABORT_WE	* Varies	This bit is used to write enable the Channel-27 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH27_ABORT): Enable Write to CH27_ABORT bit ■ 0x0 (DISABLE_WR_CH27_ABORT): Disable Write to CH27_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
57	CH26_ABORT_WE	* Varies	This bit is used to write enable the Channel-26 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH26_ABORT): Enable Write to CH26_ABORT bit ■ 0x0 (DISABLE_WR_CH26_ABORT): Disable Write to CH26_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
56	CH25_ABORT_WE	* Varies	This bit is used to write enable the Channel-25 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH25_ABORT): Enable Write to CH25_ABORT bit ■ 0x0 (DISABLE_WR_CH25_ABORT): Disable Write to CH25_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
55	CH24_ABORT_WE	* Varies	This bit is used to write enable the Channel-24 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH24_ABORT): Enable Write to CH24_ABORT bit ■ 0x0 (DISABLE_WR_CH24_ABORT): Disable Write to CH24_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
54	CH23_ABORT_WE	* Varies	This bit is used to write enable the Channel-23 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH23_ABORT): Enable Write to CH23_ABORT bit ■ 0x0 (DISABLE_WR_CH23_ABORT): Disable Write to CH23_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
53	CH22_ABORT_WE	* Varies	This bit is used to write enable the Channel-22 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH22_ABORT): Enable Write to CH22_ABORT bit ■ 0x0 (DISABLE_WR_CH22_ABORT): Disable Write to CH22_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
52	CH21_ABORT_WE	* Varies	This bit is used to write enable the Channel-21 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH21_ABORT): Enable Write to CH21_ABORT bit ■ 0x0 (DISABLE_WR_CH21_ABORT): Disable Write to CH21_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
51	CH20_ABORT_WE	* Varies	This bit is used to write enable the Channel-20 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH20_ABORT): Enable Write to CH20_ABORT bit ■ 0x0 (DISABLE_WR_CH20_ABORT): Disable Write to CH20_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
50	CH19_ABORT_WE	* Varies	This bit is used to write enable the Channel-19 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH19_ABORT): Enable Write to CH19_ABORT bit ■ 0x0 (DISABLE_WR_CH19_ABORT): Disable Write to CH19_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
49	CH18_ABORT_WE	* Varies	This bit is used to write enable the Channel-18 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH18_ABORT): Enable Write to CH18_ABORT bit ■ 0x0 (DISABLE_WR_CH18_ABORT): Disable Write to CH18_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
48	CH17_ABORT_WE	* Varies	This bit is used to write enable the Channel-17 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH17_ABORT): Enable Write to CH17_ABORT bit ■ 0x0 (DISABLE_WR_CH17_ABORT): Disable Write to CH17_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
47	CH32_ABORT	* Varies	Channel-32 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH32_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH32_ABORT): Request for Channel-32 Abort ■ 0x0 (DISABLE_CH32_ABORT): No Request for Channel-32 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
46	CH31_ABORT	* Varies	Channel-31 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH31_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH31_ABORT): Request for Channel-31 Abort ■ 0x0 (DISABLE_CH31_ABORT): No Request for Channel-31 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
45	CH30_ABORT	* Varies	Channel-30 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH30_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH30_ABORT): Request for Channel-30 Abort ■ 0x0 (DISABLE_CH30_ABORT): No Request for Channel-30 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
44	CH29_ABORT	* Varies	Channel-29 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH29_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH29_ABORT): Request for Channel-29 Abort ■ 0x0 (DISABLE_CH29_ABORT): No Request for Channel-29 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
43	CH28_ABORT	* Varies	Channel-28 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH28_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH28_ABORT): Request for Channel-28 Abort ■ 0x0 (DISABLE_CH28_ABORT): No Request for Channel-28 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
42	CH27_ABORT	* Varies	Channel-27 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH27_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH27_ABORT): Request for Channel-27 Abort ■ 0x0 (DISABLE_CH27_ABORT): No Request for Channel-27 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
41	CH26_ABORT	* Varies	Channel-26 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH26_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH26_ABORT): Request for Channel-26 Abort ■ 0x0 (DISABLE_CH26_ABORT): No Request for Channel-26 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
40	CH25_ABORT	* Varies	Channel-25 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH25_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH25_ABORT): Request for Channel-25 Abort ■ 0x0 (DISABLE_CH25_ABORT): No Request for Channel-25 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
39	CH24_ABORT	* Varies	Channel-24 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH24_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH24_ABORT): Request for Channel-24 Abort ■ 0x0 (DISABLE_CH24_ABORT): No Request for Channel-24 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
38	CH23_ABORT	* Varies	Channel-23 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH23_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH23_ABORT): Request for Channel-23 Abort ■ 0x0 (DISABLE_CH23_ABORT): No Request for Channel-23 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
37	CH22_ABORT	* Varies	Channel-22 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH22_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH22_ABORT): Request for Channel-22 Abort ■ 0x0 (DISABLE_CH22_ABORT): No Request for Channel-22 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
36	CH21_ABORT	* Varies	Channel-21 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH21_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH21_ABORT): Request for Channel-21 Abort ■ 0x0 (DISABLE_CH21_ABORT): No Request for Channel-21 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
35	CH20_ABORT	* Varies	Channel-20 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH20_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH20_ABORT): Request for Channel-20 Abort ■ 0x0 (DISABLE_CH20_ABORT): No Request for Channel-20 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
34	CH19_ABORT	* Varies	Channel-19 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH19_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH19_ABORT): Request for Channel-19 Abort ■ 0x0 (DISABLE_CH19_ABORT): No Request for Channel-19 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
33	CH18_ABORT	* Varies	Channel-18 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH18_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH18_ABORT): Request for Channel-18 Abort ■ 0x0 (DISABLE_CH18_ABORT): No Request for Channel-18 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
32	CH17_ABORT	* Varies	Channel-17 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH17_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH17_ABORT): Request for Channel-17 Abort ■ 0x0 (DISABLE_CH17_ABORT): No Request for Channel-17 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 16 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}
31	CH16_ABORT_WE	* Varies	This bit is used to write enable the Channel-16 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH16_ABORT): Enable Write to CH16_ABORT bit ■ 0x0 (DISABLE_WR_CH16_ABORT): Disable Write to CH16_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
30	CH15_ABORT_WE	* Varies	This bit is used to write enable the Channel-15 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH15_ABORT): Enable Write to CH15_ABORT bit ■ 0x0 (DISABLE_WR_CH15_ABORT): Disable Write to CH15_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
29	CH14_ABORT_WE	* Varies	This bit is used to write enable the Channel-14 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH14_ABORT): Enable Write to CH14_ABORT bit ■ 0x0 (DISABLE_WR_CH14_ABORT): Disable Write to CH14_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
28	CH13_ABORT_WE	* Varies	This bit is used to write enable the Channel-13 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH13_ABORT): Enable Write to CH13_ABORT bit ■ 0x0 (DISABLE_WR_CH13_ABORT): Disable Write to CH13_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
27	CH12_ABORT_WE	* Varies	This bit is used to write enable the Channel-12 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH12_ABORT): Enable Write to CH12_ABORT bit ■ 0x0 (DISABLE_WR_CH12_ABORT): Disable Write to CH12_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
26	CH11_ABORT_WE	* Varies	This bit is used to write enable the Channel-11 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH11_ABORT): Enable Write to CH11_ABORT bit ■ 0x0 (DISABLE_WR_CH11_ABORT): Disable Write to CH11_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
25	CH10_ABORT_WE	* Varies	This bit is used to write enable the Channel-10 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH10_ABORT): Enable Write to CH10_ABORT bit ■ 0x0 (DISABLE_WR_CH10_ABORT): Disable Write to CH10_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
24	CH9_ABORT_WE	* Varies	This bit is used to write enable the Channel-9 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH9_ABORT): Enable Write to CH9_ABORT bit ■ 0x0 (DISABLE_WR_CH9_ABORT): Disable Write to CH9_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
23	CH8_ABORT_WE	* Varies	This bit is used to write enable the Channel-8 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH8_ABORT): Enable Write to CH8_ABORT bit ■ 0x0 (DISABLE_WR_CH8_ABORT): Disable Write to CH8_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
22	CH7_ABORT_WE	* Varies	This bit is used to write enable the Channel-7 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH7_ABORT): Enable Write to CH7_ABORT bit ■ 0x0 (DISABLE_WR_CH7_ABORT): Disable Write to CH7_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
21	CH6_ABORT_WE	* Varies	This bit is used to write enable the Channel-6 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH6_ABORT): Enable Write to CH6_ABORT bit ■ 0x0 (DISABLE_WR_CH6_ABORT): Disable Write to CH6_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
20	CH5_ABORT_WE	* Varies	This bit is used to write enable the Channel-5 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH5_ABORT): Enable Write to CH5_ABORT bit ■ 0x0 (DISABLE_WR_CH5_ABORT): Disable Write to CH5_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
19	CH4_ABORT_WE	* Varies	This bit is used to write enable the Channel-4 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH4_ABORT): Enable Write to CH4_ABORT bit ■ 0x0 (DISABLE_WR_CH4_ABORT): Disable Write to CH4_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
18	CH3_ABORT_WE	* Varies	This bit is used to write enable the Channel-3 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH3_ABORT): Enable Write to CH3_ABORT bit ■ 0x0 (DISABLE_WR_CH3_ABORT): Disable Write to CH3_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
17	CH2_ABORT_WE	* Varies	This bit is used to write enable the Channel-2 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH2_ABORT): Enable Write to CH2_ABORT bit ■ 0x0 (DISABLE_WR_CH2_ABORT): Disable Write to CH2_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}
16	CH1_ABORT_WE	* Varies	This bit is used to write enable the Channel-1 Abort bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH1_ABORT): Enable Write to CH1_ABORT bit ■ 0x0 (DISABLE_WR_CH1_ABORT): Disable Write to CH1_ABORT bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "write-only" : "read-only"}

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

Bits	Name	Memory Access	Description
15	CH16_ABORT	* Varies	Channel-16 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH16_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH16_ABORT): Request for Channel-16 Abort ■ 0x0 (DISABLE_CH16_ABORT): No Request for Channel-16 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
14	CH15_ABORT	* Varies	Channel-15 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH15_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH15_ABORT): Request for Channel-15 Abort ■ 0x0 (DISABLE_CH15_ABORT): No Request for Channel-15 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
13	CH14_ABORT	* Varies	Channel-14 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH14_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH14_ABORT): Request for Channel-14 Abort ■ 0x0 (DISABLE_CH14_ABORT): No Request for Channel-14 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
12	CH13_ABORT	* Varies	Channel-13 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH13_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH13_ABORT): Request for Channel-13 Abort ■ 0x0 (DISABLE_CH13_ABORT): No Request for Channel-13 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
11	CH12_ABORT	* Varies	Channel-12 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH12_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH12_ABORT): Request for Channel-12 Abort ■ 0x0 (DISABLE_CH12_ABORT): No Request for Channel-12 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
10	CH11_ABORT	* Varies	Channel-11 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH11_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH11_ABORT): Request for Channel-11 Abort ■ 0x0 (DISABLE_CH11_ABORT): No Request for Channel-11 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
9	CH10_ABORT	* Varies	Channel-10 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH10_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH10_ABORT): Request for Channel-10 Abort ■ 0x0 (DISABLE_CH10_ABORT): No Request for Channel-10 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
8	CH9_ABORT	* Varies	Channel-9 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH9_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH9_ABORT): Request for Channel-9 Abort ■ 0x0 (DISABLE_CH9_ABORT): No Request for Channel-9 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
7	CH8_ABORT	* Varies	Channel-8 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH8_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH8_ABORT): Request for Channel-8 Abort ■ 0x0 (DISABLE_CH8_ABORT): No Request for Channel-8 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
6	CH7_ABORT	* Varies	Channel-7 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH7_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH7_ABORT): Request for Channel-7 Abort ■ 0x0 (DISABLE_CH7_ABORT): No Request for Channel-7 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
5	CH6_ABORT	* Varies	Channel-6 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH6_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH6_ABORT): Request for Channel-6 Abort ■ 0x0 (DISABLE_CH6_ABORT): No Request for Channel-6 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
4	CH5_ABORT	* Varies	Channel-5 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH5_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH5_ABORT): Request for Channel-5 Abort ■ 0x0 (DISABLE_CH5_ABORT): No Request for Channel-5 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
3	CH4_ABORT	* Varies	Channel-4 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH4_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH4_ABORT): Request for Channel-4 Abort ■ 0x0 (DISABLE_CH4_ABORT): No Request for Channel-4 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
2	CH3_ABORT	* Varies	Channel-3 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH3_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH3_ABORT): Request for Channel-3 Abort ■ 0x0 (DISABLE_CH3_ABORT): No Request for Channel-3 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
1	CH2_ABORT	* Varies	Channel-2 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH2_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH2_ABORT): Request for Channel-2 Abort ■ 0x0 (DISABLE_CH2_ABORT): No Request for Channel-2 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

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

Bits	Name	Memory Access	Description
0	CH1_ABORT	* Varies	Channel-1 Abort Request. Software sets this bit to 1 to request channel abort. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Aborting the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the master interface are completed.Aborting the channel is not recommended and should be used only in situations where a particular channel hangs due to no response from the corresponding AXI slave interface and software wants to disable the channel without resetting the entire DW_axi_dmac. It is recommended to try channel disabling first and then only opt for channel aborting. ■ 0: No Channel Abort Request. ■ 1: Request for Channel Abort. DW_axi_dmac clears this bit to 0 once the channel is aborted (when it sets CH1_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH1_ABORT): Request for Channel-1 Abort ■ 0x0 (DISABLE_CH1_ABORT): No Request for Channel-1 Abort **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

5.1.8 DMAC_INTSTATUSREG

- **Description:** DMAC Interrupt Status Register captures the combined channel interrupt for each channel and Combined common register block interrupt. This register is present provided number of DMA channels are greater than 8.
- **Size:** 64 bits
- **Offset:** 0x30
- **Exists:** DMAX_NUM_CHANNELS <= 8

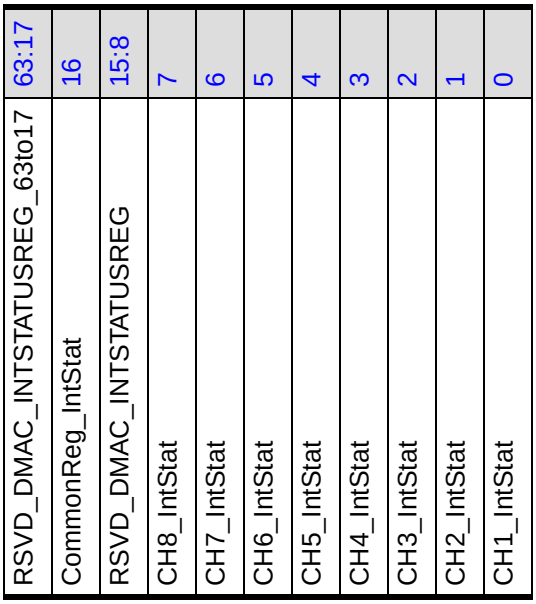


Table 5-13 Fields for Register: DMAC_INTSTATUSREG

Bits	Name	Memory Access	Description
63:17	RSVD_DMAC_INTSTATUSREG_63to17	R	DMAC Interrupt Status Register (bits 63to17) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-13 Fields for Register: DMAC_INTSTATUSREG (Continued)

Bits	Name	Memory Access	Description
16	CommonReg_IntStat	R	Common Register Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Common Register Interrupt is Active ■ 0x0 (INACTIVE): Common Register Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
15:8	RSVD_DMAC_INTSTATUSREG	R	DMAC Interrupt Status Register (bits 15to8) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7	CH8_IntStat	R	Channel 8 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 7 **Volatile:** true
6	CH7_IntStat	R	Channel 7 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 6 **Volatile:** true
5	CH6_IntStat	R	Channel 6 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 5 **Volatile:** true

Table 5-13 Fields for Register: DMAC_INTSTATUSREG (Continued)

Bits	Name	Memory Access	Description
4	CH5_IntStat	R	Channel 5 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 4 **Volatile:** true
3	CH4_IntStat	R	Channel 4 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 3 **Volatile:** true
2	CH3_IntStat	R	Channel 3 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 2 **Volatile:** true
1	CH2_IntStat	R	Channel 2 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 1 **Volatile:** true
0	CH1_IntStat	R	Channel 1 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 0 **Volatile:** true

5.1.9 DMAC_INTSTATUSREG2

- **Description:** DMAC Interrupt Status Register captures the combined channel interrupt for each channel and Combined common register block interrupt. This register is present provided number of DMA channels are less than or equal to 8.
- **Size:** 64 bits
- **Offset:** 0x30
- **Exists:** DMAX_NUM_CHANNELS > 8

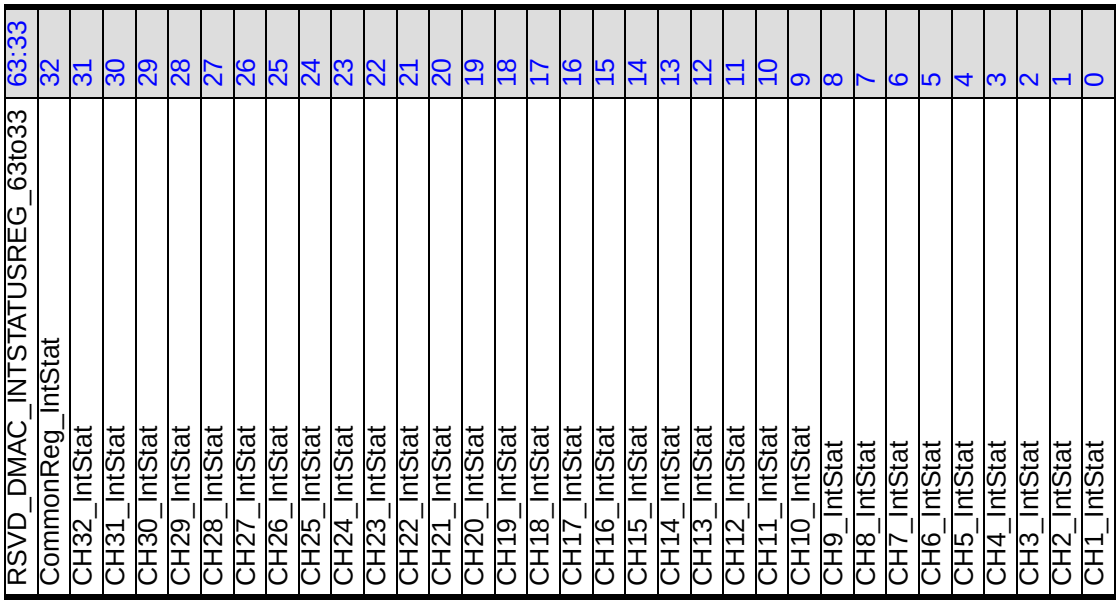


Table 5-14 Fields for Register: DMAC_INTSTATUSREG2

Bits	Name	Memory Access	Description
63:33	RSVD_DMAC_INTSTATUSREG_63to33	R	DMAC Interrupt Status Register (bits 63to33) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
32	CommonReg_IntStat	R	Common Register Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Common Register Interrupt is Active ■ 0x0 (INACTIVE): Common Register Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
31	CH32_IntStat	R	Channel 32 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 31 **Volatile:** true
30	CH31_IntStat	R	Channel 31 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 30 **Volatile:** true
29	CH30_IntStat	R	Channel 30 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 29 **Volatile:** true
28	CH29_IntStat	R	Channel 29 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 28 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
27	CH28_IntStat	R	Channel 28 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 27 **Volatile:** true
26	CH27_IntStat	R	Channel 27 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 26 **Volatile:** true
25	CH26_IntStat	R	Channel 26 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 25 **Volatile:** true
24	CH25_IntStat	R	Channel 25 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 24 **Volatile:** true
23	CH24_IntStat	R	Channel 24 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 23 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
22	CH23_IntStat	R	Channel 23 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 22 **Volatile:** true
21	CH22_IntStat	R	Channel 22 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 21 **Volatile:** true
20	CH21_IntStat	R	Channel 21 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 20 **Volatile:** true
19	CH20_IntStat	R	Channel 20 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 19 **Volatile:** true
18	CH19_IntStat	R	Channel 19 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 18 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
17	CH18_IntStat	R	Channel 18 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 17 **Volatile:** true
16	CH17_IntStat	R	Channel 17 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 16 **Volatile:** true
15	CH16_IntStat	R	Channel 16 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 15 **Volatile:** true
14	CH15_IntStat	R	Channel 15 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 14 **Volatile:** true
13	CH14_IntStat	R	Channel 14 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 13 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
12	CH13_IntStat	R	Channel 13 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 12 **Volatile:** true
11	CH12_IntStat	R	Channel 12 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 11 **Volatile:** true
10	CH11_IntStat	R	Channel 11 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 10 **Volatile:** true
9	CH10_IntStat	R	Channel 10 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 9 **Volatile:** true
8	CH9_IntStat	R	Channel 9 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 8 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
7	CH8_IntStat	R	Channel 8 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 7 **Volatile:** true
6	CH7_IntStat	R	Channel 7 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 6 **Volatile:** true
5	CH6_IntStat	R	Channel 6 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 5 **Volatile:** true
4	CH5_IntStat	R	Channel 5 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 4 **Volatile:** true
3	CH4_IntStat	R	Channel 4 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 3 **Volatile:** true

Table 5-14 Fields for Register: DMAC_INTSTATUSREG2 (Continued)

Bits	Name	Memory Access	Description
2	CH3_IntStat	R	Channel 3 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 2 **Volatile:** true
1	CH2_IntStat	R	Channel 2 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 1 **Volatile:** true
0	CH1_IntStat	R	Channel 1 Interrupt Status Bit. **Values:** ■ 0x1 (ACTIVE): Channel ${ch_num} Interrupt is Active ■ 0x0 (INACTIVE): Channel ${ch_num} Interrupt is Inactive **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS >= 0 **Volatile:** true

5.1.10 DMAC_COMMONREG_INTCLEARREG

- **Description:** Writing 1 to specific field clears the corresponding field in DMAC Common register Interrupt Status Register (DMAC_CommonReg_IntStatusReg).
- **Size:** 64 bits
- **Offset:** 0x38
- **Exists:** Always

Table 5-15 Fields for Register: DMAC_COMMONREG_INTCLEARREG

Bits	Name	Memory Access	Description
63:9	RSVD_DMAC_COMMONREG_INTCLEARREG_63to9	W	DMAC Common Register Interrupt Clear Register (bits 63to9) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-15 Fields for Register: DMAC_COMMONREG_INTCLEARREG (Continued)

Bits	Name	Memory Access	Description
8	Clear_SLVIF_UndefinedReg_DEC_ERR_IntStat	W	Slave Interface Undefined register Decode Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit(SLVIF_UndefinedReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_UndefinedReg_DEC_ERR): Clear the SLVIF_UndefinedReg_DEC_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
7:4	RSVD_DMAC_COMMONREG_INTCLEARREG_7to4	W	DMAC Common Register Interrupt Clear Register (bits 7to4) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
3	Clear_SLVIF_CommonReg_WrOnHold_ERR_IntStat	W	Slave Interface Common Register Write On Hold Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit(SLVIF_CommonReg_WrOnHold_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_WrOnHold_ERR): Clear the SLVIF_CommonReg_WrOnHold_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
2	Clear_SLVIF_CommonReg_RD2WO_ERR_IntStat	W	Slave Interface Common Register Read to Write only Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit(SLVIF_CommonReg_RD2WO_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_RD2WO_ERR): Clear the SLVIF_CommonReg_RD2WO_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-15 Fields for Register: DMAC_COMMONREG_INTCLEARREG (Continued)

Bits	Name	Memory Access	Description
1	Clear_SLVIF_CommonReg_WR2RO_ERR_IntStat	W	Slave Interface Common Register Write to Read only Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit(SLVIF_CommonReg_WR2RO_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_WR2RO_ERR): Clear the SLVIF_CommonReg_WR2RO_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
0	Clear_SLVIF_CommonReg_DEC_ERR_IntStat	W	Slave Interface Common Register Decode Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit (SLVIF_CommonReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_DEC_ERR): Clear the SLVIF_CommonReg_DEC_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

5.1.11 DMAC_COMMONREG_INTSTATUS_ENABLEREG

- **Description:** Writing 1 to specific field enables the corresponding interrupt status generation in DMAC Common register Interrupt Status Register (DMAC_CommonReg_IntStatusReg).
- **Size:** 64 bits
- **Offset:** 0x40
- **Exists:** Always

Table 5-16 Fields for Register: DMAC_COMMONREG_INTSTATUS_ENABLEREG

Bits	Name	Memory Access	Description
63:9	RSVD_DMAC_COMMONREG_INTSTATUS_ENABLEREG_63to9	R	DMAC Common Register Interrupt Status Enable Register (bits 63to9) Reserved bits - Read Only **Value After Reset:** 0x7fffffffffffff **Exists:** Always

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

Bits	Name	Memory Access	Description
8	Enable_SLVIF_UndefinedReg_DEC_ERR_IntStat	R/W	Slave Interface Undefined register Decode Error Interrupt Status enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_UndefinedReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_UndefinedReg_DEC_ERR): SLVIF_UndefinedReg_DEC_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_UndefinedReg_DEC_ERR): SLVIF_UndefinedReg_DEC_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always
7:4	RSVD_DMAC_COMMONREG_INTSTATUS_ENABLEREG_7to4	R	DMAC Common Register Interrupt Status Enable Register (bits 7to4) Reserved bits - Read Only **Value After Reset:** 0xf **Exists:** Always
3	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntStat	R/W	Slave Interface Common Register Write On Hold Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_WrOnHold_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WrOnHold_ERR): SLVIF_CommonReg_WrOnHold_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_WrOnHold_ERR): SLVIF_CommonReg_WrOnHold_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always

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

Bits	Name	Memory Access	Description
2	Enable_SLVIF_CommonReg_RD2WO_ERR_IntStat	R/W	Slave Interface Common Register Read to Write only Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_RD2WO_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_RD2WO_ERR): SLVIF_CommonReg_RD2WO_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_RD2WO_ERR): SLVIF_CommonReg_RD2WO_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always
1	Enable_SLVIF_CommonReg_WR2RO_ERR_IntStat	R/W	Slave Interface Common Register Write to Read only Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_WR2RO_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WR2RO_ERR): SLVIF_CommonReg_WR2RO_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_WR2RO_ERR): SLVIF_CommonReg_WR2RO_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always

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

Bits	Name	Memory Access	Description
0	Enable_SLVIF_CommonReg_DEC_ERR_IntStat	R/W	Slave Interface Common Register Decode Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_DEC_ERR): SLVIF_CommonReg_DEC_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_DEC_ERR): SLVIF_CommonReg_DEC_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always

5.1.12 DMAC_COMMONREG_INTSIGNAL_ENABLEREG

- **Description:** Writing 1 to specific field will propagate the corresponding interrupt status in DMAC Common register Interrupt Status Register (DMAC_CommonReg_IntStatusReg) to generate an port level interrupt.
- **Size:** 64 bits
- **Offset:** 0x48
- **Exists:** Always

Table 5-17 Fields for Register: DMAC_COMMONREG_INTSIGNAL_ENABLEREG

Bits	Name	Memory Access	Description
63:9	RSVD_DMAC_COMMONREG_INTSIGNAL_ENABLEREG_63to9	R	DMAC Common Register Interrupt Signal Enable Register (bits 63to9) Reserved bits - Read Only **Value After Reset:** 0x7fffffffffffff **Exists:** Always

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

Bits	Name	Memory Access	Description
8	Enable_SLVIF_UndefinedReg_DEC_ERR_IntSignal	R/W	Slave Interface Undefined register Decode Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit(SLVIF_UndefinedReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_UndefinedReg_DEC_ERR_IntSignal): SLVIF_UndefinedReg_DEC_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_UndefinedReg_DEC_ERR_IntSignal): SLVIF_UndefinedReg_DEC_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always
7:4	RSVD_DMAC_COMMONREG_INTSIGNAL_ENABLEREG_7to4	R	DMAC Common Register Interrupt Signal Enable Register (bits 7to4) Reserved bits - Read Only **Value After Reset:** 0xf **Exists:** Always

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

Bits	Name	Memory Access	Description
3	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntSignal	R/W	Slave Interface Common Register Write On Hold Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit(SLVIF_CommonReg_WrOnHold_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WrOnHold_ERR_IntSignal): SLVIF_CommonReg_WrOnHold_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_WrOnHold_ERR_IntSignal): SLVIF_CommonReg_WrOnHold_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always
2	Enable_SLVIF_CommonReg_RD2WO_ERR_IntSignal	R/W	Slave Interface Common Register Read to Write only Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit (SLVIF_CommonReg_RD2WO_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_RD2WO_ERR_IntSignal): SLVIF_CommonReg_RD2WO_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_RD2WO_ERR_IntSignal): SLVIF_CommonReg_RD2WO_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always

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

Bits	Name	Memory Access	Description
1	Enable_SLVIF_CommonReg_WR2RO_ERR_IntSignal	R/W	Slave Interface Common Register Write to Read only Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit (SLVIF_CommonReg_WR2RO_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WR2RO_ERR_IntSignal): SLVIF_CommonReg_WR2RO_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_WR2RO_ERR_IntSignal): SLVIF_CommonReg_WR2RO_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always
0	Enable_SLVIF_CommonReg_DEC_ERR_IntSignal	R/W	Slave Interface Common Register Decode Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit (SLVIF_CommonReg_DEC_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_DEC_ERR_IntSignal): SLVIF_CommonReg_DEC_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_DEC_ERR_IntSignal): SLVIF_CommonReg_DEC_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always

5.1.13 DMAC_COMMONREG_INTSTATUSREG

- **Description:** This Register captures Slave interface access errors.
  - Decode Error.
  - Write to read only register.
  - Read to write only register.
  - write on hold.
  - undefined address.
- **Size:** 64 bits
- **Offset:** 0x50
- **Exists:** Always

63:9	8	7:4	3	2	1	0
RSVD_DMAC_COMMONREG_INTSTATUSREG_63to9	SLVIF_UndefinedReg_DEC_ERR_IntStat	RSVD_DMAC_COMMONREG_INTSTATUSREG_7to4	SLVIF_CommonReg_WrOnHold_ERR_IntStat	SLVIF_CommonReg_RD2WO_ERR_IntStat	SLVIF_CommonReg_WR2RO_ERR_IntStat	SLVIF_CommonReg_DEC_ERR_IntStat

Table 5-18 Fields for Register: DMAC_COMMONREG_INTSTATUSREG

Bits	Name	Memory Access	Description
63:9	RSVD_DMAC_COMMONREG_INTSTATUSREG_63to9	R	DMAC Common Register Interrupt Signal Enable Register (bits 63to9) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
8	SLVIF_UndefinedReg_DEC_ERR_IntStat	R	Slave Interface Undefined register Decode Error Interrupt Signal Enable Bit. Decode Error generated by DW_axi_dmac during register access. This error occurs if the register access is to undefined address range (>0x8FF if 8 channels are configured, >0x4FF if 4 channels are configured etc.) resulting in error response by DW_axi_dmac slave interface. ■ 0: No Slave Interface Decode Errors. ■ 1: Slave Interface Decode Error detected. Error Interrupt Status is generated if the corresponding Status Enable bit in DMAC_CommonReg_IntStatus_Enable register bit is set to 1. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in DMAC_COMMONREG_INTCLEARREG on enabling the channel (required when the interrupt is not enabled). **Values:** ■ 0x1 (Active_UndefinedReg_DEC_ERR): Slave Interface Decode Error detected ■ 0x0 (Inactive_UndefinedReg_DEC_ERR): No Slave Interface Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7:4	RSVD_DMAC_COMMONREG_INTSTATUSREG_7to4	R	DMAC Common Register Interrupt Status Register (bits 7to4) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
3	SLVIF_CommonReg_WrOnHold_ERR_IntStat	R	Slave Interface Common Register Write On Hold Error Interrupt Status Bit. This error occurs if an illegal write operation is performed on a common register; this happens if a write operation is performed on a common register except DMAC_RESETREG with DMAC_RST field set to 1 when DW_axi_dmac is in Hold mode. ■ 0: No Slave Interface Common Register Write On Hold Errors. ■ 1: Slave Interface Common Register Write On Hold Error detected. Error Interrupt Status is generated if the corresponding Status Enable bit in DMAC_CommonReg_IntStatus_Enable register bit is set to 1. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in DMAC_COMMONREG_INTCLEARREG on enabling the channel (required when the interrupt is not enabled). **Values:** ■ 0x1 (Active_CommonReg_WrOnHold_ERR): Slave Interface Common Register Write On Hold Error detected ■ 0x0 (Inactive_CommonReg_WrOnHold_ERR): No Slave Interface Common Register Write On Hold Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
2	SLVIF_CommonReg_RD2WO_ERR_IntStat	R	Slave Interface Common Register Read to Write only Error Interrupt Status bit. This error occurs if Read operation is performed to a Write Only register in the common register space (0x000 to 0x0FF). ■ 0: No Slave Interface Read to Write Only Errors. ■ 1: Slave Interface Read to Write Only Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_CommonReg_IntStatus_Enable register bit is set to 1. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in DMAC_COMMONREG_INTCLEARREG on enabling the channel (required when the interrupt is not enabled). **Values:** ■ 0x1 (Active_CommonReg_RD2WO_ERR): Slave Interface Read to Write Only Error detected ■ 0x0 (Inactive_CommonReg_RD2WO_ERR): No Slave Interface Read to Write Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	SLVIF_CommonReg_WR2RO_ERR_IntStat	R	Slave Interface Common Register Write to Read Only Error Interrupt Status bit. This error occurs if write operation is performed to a Read Only register in the common register space (0x000 to 0x0FF). ■ 0: No Slave Interface Write to Read Only Errors. ■ 1: Slave Interface Write to Read Only Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_CommonReg_IntStatus_Enable register bit is set to 1. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in DMAC_COMMONREG_INTCLEARREG on enabling the channel (required when the interrupt is not enabled). **Values:** ■ 0x1 (Active_CommonReg_WR2RO_ERR): No Slave Interface Write to Read Only Errors ■ 0x0 (Inactive_CommonReg_WR2RO_ERR): Slave Interface Write to Read Only Error detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
0	SLVIF_CommonReg_DEC_ERR_IntStat	R	Slave Interface Common Register Decode Error Interrupt Status Bit. Decode Error generated by DW_axi_dmac during register access. This error occurs if the register access is to an invalid address in the common register space (0x000 to 0x0FF) resulting in error response by DW_axi_dmac slave interface. ■ 0: No Slave Interface Decode Errors. ■ 1: Slave Interface Decode Error detected. The Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_CommonReg_IntStatus_Enable register bit is set to 1. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in DMAC_COMMONREG_INTCLEARREG on enabling the channel (required when the interrupt is not enabled). **Values:** ■ 0x1 (Active_CommonReg_DEC_ERR): Slave Interface Decode Error detected ■ 0x0 (Inactive_CommonReg_DEC_ERR): No Slave Interface Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.14 DMAC_RESETREG

- **Description:** This register is used to initiate the Software Reset to DW_axi_dmac.
- **Size:** 64 bits
- **Offset:** 0x58
- **Exists:** Always

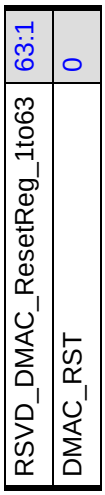


Table 5-19 Fields for Register: DMAC_RESETREG

Bits	Name	Memory Access	Description
63:1	RSVD_DMAC_ResetReg_1to63	R	DMAC_ResetReg (bits 1to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	DMAC_RST	R/W	DMAC Reset Request bit Software writes 1 to this bit to reset the DW_axi_dmac and polls this bit to see it as 0. DW_axi_dmac resets all the modules except the slave bus interface module and clears this bit to 0. **NOTE**: Software is not allowed to write 0 to this bit. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.1.15 DMAC_LOWPOWER_CFGREG

- **Description:** This register contains the fields that configures the Context Sensitive Low Power feature. This register should be programmed prior to enabling the channel.
- **Size:** 64 bits
- **Offset:** 0x60
- **Exists:** Always

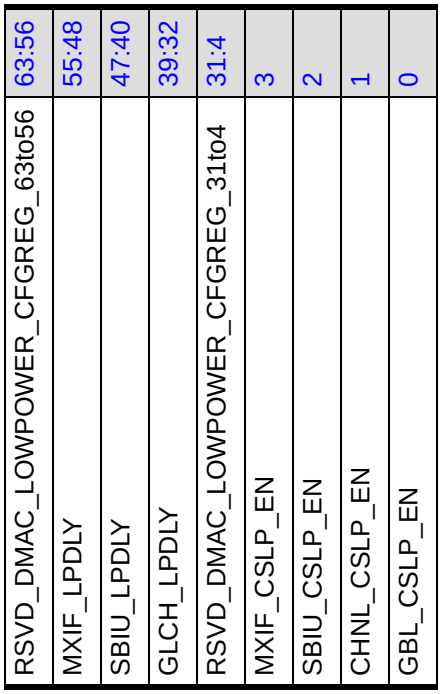


Table 5-20 Fields for Register: DMAC_LOWPOWER_CFGREG

Bits	Name	Memory Access	Description
63:56	RSVD_DMAC_LOWPOWER_CFGREG_63to56	R	DMAC_LOWPOWER_CFGREG (bits 56to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-20 Fields for Register: DMAC_LOWPOWER_CFGREG (Continued)

Bits	Name	Memory Access	Description
55:48	MXIF_LPDLY	R/W	Defines the load value to be programmed into the AXI Master Interface low power delay counter. The programmed value must be greater than or equal to 0x4. If value programmed is less than 0x4, then the register value is reset to DMAX_MXIF_LPDLY. The maximum value programmed into this register field is limited to (2**DMAX_MXIF_LPDLY_WIDTH)-1, otherwise the upper bits (8-DMAX_MXIF_LPDLY_WIDTH) of this field is reset to 0x0. **Value After Reset:** "(DMAX_MXIF_CSLP_EN==1) ? DMAX_MXIF_LPDLY : 0x0" **Exists:** Always **Volatile:** true
47:40	SBIU_LPDLY	R/W	Defines the load value to be programmed into the SBIU low power delay counter. The programmed value must be greater than or equal to 0x4. If value programmed is less than 0x4, then the register value is reset to DMAX_SBIU_LPDLY. The maximum value programmed into this register field is limited to (2**DMAX_SBIU_LPDLY_WIDTH)-1, otherwise the upper bits (8-DMAX_SBIU_LPDLY_WIDTH) of this field is reset to 0x0. **Value After Reset:** "(DMAX_SBIU_CSLP_EN==1) ? DMAX_SBIU_LPDLY : 0x0" **Exists:** Always **Volatile:** true
39:32	GLCH_LPDLY	R/W	Defines the load value to be programmed into the Global and DMA Channel low power delay counter. The programmed value must be greater than or equal to 0x4. If value programmed is less than 0x4, then the register value is reset to DMAX_GLCH_LPDLY. The maximum value programmed into this register field is limited to (2**DMAX_GLCH_LPDLY_WIDTH)-1, otherwise the upper bits (8-DMAX_GLCH_LPDLY_WIDTH) of this field is reset to 0x0. **Value After Reset:** "(DMAX_CSLP_EN==1) ? DMAX_GLCH_LPDLY : 0x0" **Exists:** Always **Volatile:** true
31:4	RSVD_DMAC_LOWPOWER_CFGREG_31to4	R	DMAC_LOWPOWER_CFGREG (bits 4to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-20 Fields for Register: DMAC_LOWPOWER_CFGREG (Continued)

Bits	Name	Memory Access	Description
3	MXIF_CSLP_EN	R/W	AXI Master Interface Context Sensitive Low Power feature enable. **Values:** ■ 0x0 (MXIF_CSLP_DISABLE): AXI Master Interface Context Sensitive Low Power feature is disabled ■ 0x1 (MXIF_CSLP_ENABLE): AXI Master Interface Context Sensitive Low Power feature is enabled **Value After Reset:** "(DMAX_MXIF_CSLP_EN==1) ? 0x1 : 0x0" **Exists:** Always **Volatile:** true
2	SBIU_CSLP_EN	R/W	SBIU Context Sensitive Low Power feature enable. **Values:** ■ 0x0 (SBIU_CSLP_DISABLE): SBIU Context Sensitive Low Power feature is disabled ■ 0x1 (SBIU_CSLP_ENABLE): SBIU Context Sensitive Low Power feature is enabled **Value After Reset:** "(DMAX_SBIU_CSLP_EN==1) ? 0x1 : 0x0" **Exists:** Always **Volatile:** true
1	CHNL_CSLP_EN	R/W	DMA Channel Context Sensitive Low Power feature enable. **Values:** ■ 0x0 (CHNL_CSLP_DISABLE): DMA Channel Context Sensitive Low Power feature is disabled ■ 0x1 (CHNL_CSLP_ENABLE): DMA Channel Context Sensitive Low Power feature is enabled **Value After Reset:** "(DMAX_CHNL_CSLP_EN==1) ? 0x1 : 0x0" **Exists:** Always **Volatile:** true
0	GBL_CSLP_EN	R/W	Global Context Sensitive Low Power feature enable. **Values:** ■ 0x0 (GBL_CSLP_DISABLE): Global Context Sensitive Low Power feature is disabled ■ 0x1 (GBL_CSLP_ENABLE): Global Context Sensitive Low Power feature is enabled **Value After Reset:** "(DMAX_CSLP_EN==1) ? 0x1 : 0x0" **Exists:** Always **Volatile:** true

5.2 DW_axi_dmac_mem_map/Channelx_Registers_Address_Block Registers

DW_axi_dmac Channel x register address block Follow the link for the register to see a detailed description of the register.

Table 5-21 DW_axi_dmac_mem_map/Channelx_Registers_Address_Block Registers

Register	Offset	Description
CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS) on page 341	0x100 + (x-1)*0x100	The starting source address is programmed by software before the DMA channel is enabled, or by an...
CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS) on page 342	0x108 + (x-1)*0x100	The starting destination address is programmed by the software before the DMA channel is enabled,...
CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS) on page 343	0x110 + (x-1)*0x100	When DW_axi_dmac is the flow controller, the DMAC uses this register before the channel is enabled...
CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) on page 345	0x118 + (x-1)*0x100	This register contains fields that control the DMA transfer. This register should be programmed...
CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 359	0x120 + (x-1)*0x100	This register contains fields that configure the DMA transfer. This register should be programmed...
CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) on page 371	0x120 + (x-1)*0x100	This register contains fields that configure the DMA transfer. This register should be programmed...
CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS) on page 383	0x128 + (x-1)*0x100	This is the Linked List Pointer register. This register must be programmed to point to the first...
CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 385	0x130 + (x-1)*0x100	Channelx Status Register contains fields that indicate the status of DMA transfers for...
CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 387	0x138 + (x-1)*0x100	Channelx Software handshake Source Register.
CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 391	0x140 + (x-1)*0x100	Channelx Software handshake Destination Register.
CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 396	0x148 + (x-1)*0x100	Channelx Block Transfer Resume Request Register. This register is used during Linked List or Shadow...

Table 5-21 DW_axi_dmac_mem_map/Channelx_Registers_Address_Block Registers

Register	Offset	Description
CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 398	0x150 + (x-1)*0x100	Channelx AXI ID Register. This register is allowed to be updated only when the channel is disabled,...
CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 401	0x158 + (x-1)*0x100	Channelx AXI QOS Register. This register is allowed to be updated only when the channel is disabled,...
CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) on page 403	0x160 + (x-1)*0x100	Channelx Source Status Register. After each block transfer completes, hardware can retrieve the...
CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) on page 405	0x168 + (x-1)*0x100	Channelx Destination Status Register. After each block transfer completes, hardware can retrieve...
CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS) on page 407	0x170 + (x-1)*0x100	Channelx Source Status Fetch Register. After completion of each block transfer, hardware can retrieve...
CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS) on page 408	0x178 + (x-1)*0x100	Channelx Destination Status Fetch Register. After completion of each block transfer, hardware can...
CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 409	0x180 + (x-1)*0x100	Writing 1 to specific field enables the corresponding interrupt status generation in Channelx Interrupt...
CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) on page 423	0x188 + (x-1)*0x100	Channelx Interrupt Status Register captures the Channelx specific interrupts
CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 441	0x190 + (x-1)*0x100	This register contains fields that are used to enable the generation of port level interrupt at...
CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) on page 455	0x198 + (x-1)*0x100	Writing 1 to specific field will clear the corresponding field in Channelx Interrupt Status...

5.2.1 CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** The starting source address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the source address of the current AXI transfer.
- **Size:** 64 bits
- **Offset:** 0x100 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

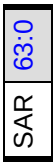


Table 5-22 Fields for Register: CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:0	SAR	R/W	Current Source Address of DMA transfer. Updated after each source transfer. The SINC fields in the CHx_CTL register determines whether the address increments or is left unchanged on every source transfer throughout the block transfer. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.2 CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** The starting destination address is programmed by the software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the destination address of the current AXI transfer.
- **Size:** 64 bits
- **Offset:** 0x108 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

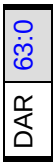


Table 5-23 Fields for Register: CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:0	DAR	R/W	Current Destination Address of DMA transfer. Updated after each destination transfer. The DINC fields in the CHx_CTL register determines whether the address increments or is left unchanged on every destination transfer throughout the block transfer. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.3 CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** When DW_axi_dmac is the flow controller, the DMAC uses this register before the channel is enabled for block-size.
- **Size:** 64 bits
- **Offset:** 0x110 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

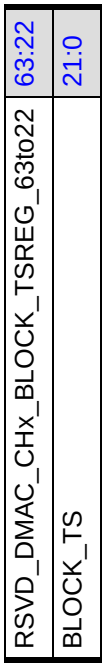


Table 5-24 Fields for Register: CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:22	RSVD_DMAC_CHx_BLOCK_TS REG_63to22	R	DMAC Channelx Block Transfer Size Register (bits 63to22) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-24 Fields for Register: CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
21:0	BLOCK_TS	R/W	Block Transfer Size. The number programmed into BLOCK_TS field indicates the total number of data of width CHx_CTL.SRC_TR_WIDTH to be transferred in a DMA block transfer. Block Transfer Size = BLOCK_TS+1 When the transfer starts, the read-back value is the total number of data items already read from the source peripheral, regardless of who is the flow controller. When the source or destination peripheral is assigned as the flow controller, the value before the transfer starts saturates at DMAX_CHx_MAX_BLK_SIZE, but the actual block size can be greater. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.4 CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** This register contains fields that control the DMA transfer. This register should be programmed prior to enabling the channel except for LLI-based multi-block transfer. When LLI-based multi-block transfer is enabled, the CHx_CTL register is loaded from the corresponding location of the LLI and it can be varied on a block-by-block basis within a DMA transfer. The software is not allowed to directly update this register through DW_axi_dmac slave interface. Any write to this register during LLI based multi-block transfer is ignored.
- **Size:** 64 bits
- **Offset:** 0x118 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63	62	61:59	58	57	56	55:48	47	46:39	38	37:35	34:32	31	30	29:26	25:22	21:18	17:14	13:11	10:8	7	6	5	4	3	2	1	0
SHADOWREG_OR_LLI_VALID	SHADOWREG_OR_LLI_LAST	RSVD_DMAC_CHx_CTL_59to61	IOC_BlkTfr	DST_STAT_EN	SRC_STAT_EN	AWLEN	AWLEN_EN	ARLEN	ARLEN_EN	AW_PROT	AR_PROT	RSVD_DMAC_CHx_CTL_31	NonPosted_LastWrite_En	AW_CACHE	AR_CACHE	DST_MSIZE	SRC_MSIZE	DST_TR_WIDTH	SRC_TR_WIDTH	RSVD_DMAC_CHx_CTL_7	DINC	RSVD_DMAC_CHx_CTL_5	SINC	RSVD_DMAC_CHx_CTL_3	DMS	RSVD_DMAC_CHx_CTL_1	SMS

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63	SHADOWREG_OR_LLI_VALID	R/W	Shadow Register content/Linked List Item valid. Indicates whether the content of shadow register or the linked list item fetched from the memory is valid. ■ 0: Shadow Register content/LLI is invalid. ■ 1: Last Shadow Register/LLI is valid. **LLI based multi-block transfer**: The CHx_CTL register is loaded from the LLI. Hence, the software is not allowed to directly update this register through the DW_axi_dmac slave interface. This field can be used to dynamically extend the LLI by the software. On noticing this bit as 0, DW_axi_dmac discards the LLI and generates the ShadowReg_Or_LII_Invalid_ERR Interrupt if the corresponding channel error interrupt mask bit is set to 0. In the case of LLI pre-fetching, the ShadowReg_Or_LLI_Invalid_ERR interrupt is not generated even if the ShadowReg_Or_LLI_Valid bit is seen to be 0 for the pre-fetched LLI. In this case, DW_axi_dmac attempts the LLI fetch operation again after completing the current block transfer and generates the ShadowReg_Or_LII_Invalid_ERR interrupt only if ShadowReg_Or_LII_Valid bit is still seen to be 0.

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
			This error condition causes the DW_axi_dmac to halt the corresponding channel gracefully. DW_axi_dmac waits until software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability before attempting another LLI read operation. This bit is cleared to 0 and written back to the corresponding LLI location after block transfer completion when LLI write-back option is enabled. Hence, for LLI-based multi-block transfers, the software might manipulate/redefine any descriptor with the ShadowReg_Or_LII_Valid bit set to 0 if LLI write-back option is enabled. **Shadow Reg based multi-block transfer**: On noticing this bit as 0 during shadow register fetch phase, DW_axi_dmac discards the Shadow Register contents and generates ShadowReg_Or_LLI_Invlid_ERR Interrupt. In this case, the software has to write (any value) to CHx_BLK_TFR_ResumeReqReg after updating the shadow registers and after setting ShadowReg_Or_LLI_Valid bit to 1 to indicate to DW_axi_dmac that shadow register contents are valid and the next block transfer can be resumed. DW_axi_dmac clears this bit to 0 after copying the shadow register contents. Software can reprogram the shadow registers only if ShadowReg_Or_LLI_Valid bit is 0. Software needs to read this register in block completion interrupt service routine (if interrupt is enabled)/continuously poll this register (if interrupt is not enabled) to make sure that this bit is 0 before updating the shadow registers. If shadow-register-based multi-block transfer is enabled and software attempts to write to the shadow register when ShadowReg_Or_LLI_Valid bit is 1, DW_axi_dmac generates SLVIF_ShadowReg_WrOnValid_ERR interrupt. **Values:** ■ 0x1 (VALID): Indicates shadowreg/LLI content is Valid ■ 0x0 (INVALID): Indicates shadowreg/LLI content is Invalid **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
62	SHADOWREG_OR_LLI_LAST	R/W	Last Shadow Register/Linked List Item. Indicates whether shadow register content or the linked list item fetched from the memory is the last one or not. ■ 0: Not last Shadow Register/LLI ■ 1: Last Shadow Register/LLI **LLI based multi-block transfer**: DW_axi_dmac uses this bit to decide if another LLI fetch is needed in the current DMA transfer. ■ If this bit is 0, DW_axi_dmac fetches the next LLI from the address pointed out by LLP field in the current LLI. ■ If this bit is 1, DW_axi_dmac understands that current block is the final block in the dma transfer and ends the dma transfer once the AMBA transfer corresponding to the current block completes. **Shadow Reg based multi-block transfer**: DW_axi_dmac uses this bit to decide if another Shadow Register fetch is needed in the current DMA transfer. ■ If this bit is 0, DW_axi_dmac understands that there are one or more blocks to be transferred in the current block and hence one or more shadow register set contents will be valid and needs to be fetched. ■ If this bit is 1, DW_axi_dmac understands that current block is the final block in the dma transfer and ends the dma transfer once the AMBA transfer corresponding to the current block completes. **Values:** ■ 0x1 (LAST_ITEM): Indicates shadowreg/LLI content is the last one ■ 0x0 (NOT_LAST_ITEM): Indicates shadowreg/LLI content is not the last one **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
61:59	RSVD_DMAC_CHx_CTL_59to61	R	DMAC Channelx Control Transfer Register (bits 59to61) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
58	IOC_BlkTfr	R/W	Interrupt On completion of Block Transfer This bit is used to control the block transfer completion interrupt generation on a block by block basis for shadow register or linked list based multi-block transfers. Writing 1 to this register field enables CHx_IntStatusReg.BLOCK_TFR_DONE_IntStat field if this interrupt generation is enabled in CHx_IntStatus_EnableReg register and the external interrupt output is is asserted if this interrupt generation is enabled in CHx_IntSignal_EnableReg register. **Note**: If a linked-list or shadow-register-based multi-block transfer is not used for both source and destination (for instance if source and destination use contiguous address or auto-reload-based multi-block transfer), the value of this field cannot be modified per block. Additionally, the value programmed before the channel is enabled is used for all the blocks in the DMA transfer. **Values:** ■ 0x1 (Enable_BLKTFR_INTR): Enables CHx_IntStatusReg.BLOCK_TFR_DONE_IntStat field ■ 0x0 (DISABLE_BLKTFR_INTR): Disables CHx_IntStatusReg.BLOCK_TFR_DONE_IntStat field **Value After Reset:** 0x0 **Exists:** Always
57	DST_STAT_EN	{(DMAX_CH(x)_DST_STAT_EN == 1) ? "read-write" : "read-only"}	Destination Status Enable Enable the logic to fetch status from destination peripheral of channel x pointed to by the content of CHx_DSTATAR register and stores it in CHx_DSTAT register. This value is written back to the CHx_DSTAT location of linked list at end of each block transfer if DMAX_CHx_LLI_WB_EN is set to 1 and if linked list based multi-block transfer is used by either source or destination peripheral. **Values:** ■ 0x1 (Enable_STAT_FETCH): Enables status fetch for Destination and store the value in CH1_DSTAT register ■ 0x0 (NO_STAT_FETCH): No status fetch for Destination device **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
56	SRC_STAT_EN	{(DMAX_CH(x)_SRC_STAT_EN == 1) ? "read-write" : "read-only"}	Source Status Enable Enable the logic to fetch status from source peripheral of channel x pointed to by the content of CHx_SSTATAR register and stores it in CHx_SSTAT register. This value is written back to the CHx_SSTAT location of linked list at end of each block transfer if DMAX_CHx_LLI_WB_EN is set to 1 and if linked list based multi-block transfer is used by either source or destination peripheral. **Values:** ■ 0x1 (Enable_STAT_FETCH): Enables status fetch for Source and store the value in CH1_SSTAT register ■ 0x0 (NO_STAT_FETCH): No status fetch for Source device **Value After Reset:** 0x0 **Exists:** Always
55:48	AWLEN	R/W	Destination Burst Length AXI Burst length used for destination data transfer. The specified burst length is used for destination data transfer till the extent possible; remaining transfers use maximum possible value that is less than or equal to DMAX_CHx_MAX_AMBA_BURST_LENGTH. The maximum value of AWLEN is limited by DMAX_CHx_MAX_AMBA_BURST_LENGTH. **Note**: The AWLEN setting may not be honored towards end-to-block transfers, the end of a transaction (only applicable to non-memory peripharals), and during 4K boundary crossings. **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
47	AWLEN_EN	R/W	Destination Burst Length Enable If this bit is set to 1, DW_axi_dmac uses the value of CHx_CTL.AWLEN as AXI Burst length for destination data transfer till the extent possible; remaining transfers use maximum possible burst length. If this bit is set to 0, DW_axi_dmac uses any possible value which is less than or equal to DMAX_CHx_MAX_AMBA_BURST_LENGTH as AXI Burst length for destination data transfer. **Values:** ■ 0x1 (Enable): AXI Burst Length is CH1_CTL.AWLEN (till the extent possible) for Destination data transfers ■ 0x0 (Disable): AXI Burst Length is any possible value <= DMAX_CH1_MAX_AMBA_BURST_LENGTH for Destination data transfers **Value After Reset:** 0x0 **Exists:** Always
46:39	ARLEN	R/W	Source Burst Length AXI Burst length used for source data transfer. The specified burst length is used for source data transfer till the extent possible; remaining transfers use maximum possible value that is less than or equal to DMAX_CHx_MAX_AMBA_BURST_LENGTH. The maximum value of ARLEN is limited by DMAX_CHx_MAX_AMBA_BURST_LENGTH **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
38	ARLEN_EN	R/W	Source Burst Length Enable If this bit is set to 1, DW_axi_dmac uses the value of CHx_CTL.ARLEN as AXI Burst length for source data transfer till the extent possible; remaining transfers use maximum possible burst length. If this bit is set to 0, DW_axi_dmac uses any possible value that is less than or equal to DMAX_CHx_MAX_AMBA_BURST_LENGTH as AXI Burst length for source data transfer. **Values:** ■ 0x1 (Enable): AXI Burst Length is CH1_CTL.ARLEN (till the extent possible) for Source data transfers ■ 0x0 (Disable): AXI Burst Length is any possible value <= DMAX_CH1_MAX_AMBA_BURST_LENGTH for Source data transfers **Value After Reset:** 0x0 **Exists:** Always
37:35	AW_PROT	R/W	AXI 'aw_prot' signal **Value After Reset:** 0x0 **Exists:** Always
34:32	AR_PROT	R/W	AXI 'ar_prot' signal **Value After Reset:** 0x0 **Exists:** Always
31	RSVD_DMAC_CHx_CTL_31	R	DMAC Channelx Control Transfer Register bit31 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
30	NonPosted_LastWrite_En	R/W	Non Posted Last Write Enable This bit decides whether posted writes can be used throughout the block transfer. ■ 0: Posted writes may be used throughout the block transfer. ■ 1: Posted writes may be used till the end of the block (inside a block) and the last write in the block must be non-posted. This is to synchronize block completion interrupt generation to the last write data reaching the end memory/peripheral. **Values:** ■ 0x1 (Enable): Last write in the block must be non-posted ■ 0x0 (Disable): Posted writes may be used throughout the block transfer **Value After Reset:** 0x0 **Exists:** Always
29:26	AW_CACHE	R/W	AXI 'aw_cache' signal **Value After Reset:** 0x0 **Exists:** Always
25:22	AR_CACHE	R/W	AXI 'ar_cache' signal **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
21:18	DST_MSIZE	R/W	Destination Burst Transaction Length. Number of data items, each of width CHx_CTL.DST_TR_WIDTH, to be written to the destination every time a destination burst transaction request is made from the corresponding hardware or software handshaking interface.**Note**: This Value is not related to the AXI awlen signal. **Values:** ■ 0x0 (DATA_ITEM_1): 1 Data Item read from Destination in the burst transaction ■ 0x1 (DATA_ITEMS_4): 4 Data Item read from Destination in the burst transaction ■ 0x2 (DATA_ITEMS_8): 8 Data Item read from Destination in the burst transaction ■ 0x3 (DATA_ITEMS_16): 16 Data Item read from Destination in the burst transaction ■ 0x4 (DATA_ITEMS_32): 32 Data Item read from Destination in the burst transaction ■ 0x5 (DATA_ITEMS_64): 64 Data Item read from Destination in the burst transaction ■ 0x6 (DATA_ITEMS_128): 128 Data Item read from Destination in the burst transaction ■ 0x7 (DATA_ITEMS_256): 256 Data Item read from Destination in the burst transaction ■ 0x8 (DATA_ITEMS_512): 512 Data Item read from Destination in the burst transaction ■ 0x9 (DATA_ITEMS_1024): 1024 Data Item read from Destination in the burst transaction **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
17:14	SRC_MSIZE	R/W	Source Burst Transaction Length. Number of data items, each of width CHx_CTL.SRC_TR_WIDTH, to be read from the source every time a source burst transaction request is made from the corresponding hardware or software handshaking interface. The maximum value of DST_MSIZE is limited by DMAX_CHx_MAX_MSIZE. **Note**: This Value is not related to the AXI arlen signal. **Values:** ■ 0x0 (DATA_ITEM_1): 1 Data Item read from Source in the burst transaction ■ 0x1 (DATA_ITEMS_4): 4 Data Item read from Source in the burst transaction ■ 0x2 (DATA_ITEMS_8): 8 Data Item read from Source in the burst transaction ■ 0x3 (DATA_ITEMS_16): 16 Data Item read from Source in the burst transaction ■ 0x4 (DATA_ITEMS_32): 32 Data Item read from Source in the burst transaction ■ 0x5 (DATA_ITEMS_64): 64 Data Item read from Source in the burst transaction ■ 0x6 (DATA_ITEMS_128): 128 Data Item read from Source in the burst transaction ■ 0x7 (DATA_ITEMS_256): 256 Data Item read from Source in the burst transaction ■ 0x8 (DATA_ITEMS_512): 512 Data Item read from Source in the burst transaction ■ 0x9 (DATA_ITEMS_1024): 1024 Data Item read from Source in the burst transaction **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
13:11	DST_TR_WIDTH	{(DMAX_CH(x)_DTW == 0) ? "read-write" : "read-only"}	Destination Transfer Width. Mapped to AXI bus awsize, this value must be less than or equal to DMAX_M_DATA_WIDTH. **Values:** ■ 0x0 (BITS_8): Destination Transfer Width is 8 bits ■ 0x1 (BITS_16): Destination Transfer Width is 16 bits ■ 0x2 (BITS_32): Destination Transfer Width is 32 bits ■ 0x3 (BITS_64): Destination Transfer Width is 64 bits ■ 0x4 (BITS_128): Destination Transfer Width is 128 bits ■ 0x5 (BITS_256): Destination Transfer Width is 256 bits ■ 0x6 (BITS_512): Destination Transfer Width is 512 bits **Value After Reset:** {(DMAX_CH(x)_DTW_ENC < 7) ? DMAX_CH(x)_DTW_ENC : 2} **Exists:** Always
10:8	SRC_TR_WIDTH	{(DMAX_CH(x)_STW == 0) ? "read-write" : "read-only"}	Source Transfer Width. Mapped to AXI bus arsize, this value must be less than or equal to DMAX_M_DATA_WIDTH. **Values:** ■ 0x0 (BITS_8): Source Transfer Width is 8 bits ■ 0x1 (BITS_16): Source Transfer Width is 16 bits ■ 0x2 (BITS_32): Source Transfer Width is 32 bits ■ 0x3 (BITS_64): Source Transfer Width is 64 bits ■ 0x4 (BITS_128): Source Transfer Width is 128 bits ■ 0x5 (BITS_256): Source Transfer Width is 256 bits ■ 0x6 (BITS_512): Source Transfer Width is 512 bits **Value After Reset:** {(DMAX_CH(x)_STW_ENC < 7) ? DMAX_CH(x)_STW_ENC : 2} **Exists:** Always
7	RSVD_DMAC_CHx_CTL_7	R	DMAC Channelx Control Transfer Register bit7 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
6	DINC	R/W	Destination Address Increment. Indicates whether to increment the destination address on every destination transfer. If the device is writing data from a source peripheral FIFO with a fixed address, then set this field to 'No change'. ■ 0: Increment ■ 1: No Change **NOTE**: Increment aligns the address to the next CHx_CTL.DST_TR_WIDTH boundary. **Values:** ■ 0x0 (INCREMENTAL): Destination address incremented on every source transfer ■ 0x1 (FIXED): Destination address is fixed **Value After Reset:** 0x0 **Exists:** Always
5	RSVD_DMAC_CHx_CTL_5	R	DMAC Channelx Control Transfer Register bit5 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
4	SINC	R/W	Source Address Increment. Indicates whether to increment the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to 'No change'. ■ 0: Increment ■ 1: No Change **NOTE**: Increment aligns the address to the next CHx_CTL.SRC_TR_WIDTH boundary. **Values:** ■ 0x0 (INCREMENTAL): Source address incremented on every source transfer ■ 0x1 (FIXED): Source address is fixed **Value After Reset:** 0x0 **Exists:** Always
3	RSVD_DMAC_CHx_CTL_3	R	DMAC Channelx Control Transfer Register bit3 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-25 Fields for Register: CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
2	DMS	{(DMAX_CH(x)_DMS == 2) ? "read-write" : "read-only"}	Destination Master Select. Identifies the Master Interface layer from which the destination device (peripheral or memory) is accessed. ■ 0: AXI master 1 ■ 1: AXI Master 2 **Values:** ■ 0x1 (MASTER2_INTF): Destination device on Master-2 interface layer ■ 0x0 (MASTER1_INTF): Destination device on Master-1 interface layer **Value After Reset:** {(DMAX_CH(x)_DMS < 2) ? DMAX_CH(x)_DMS : 0} **Exists:** Always
1	RSVD_DMAC_CHx_CTL_1	R	DMAC Channelx Control Transfer Register bit1 Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	SMS	{(DMAX_CH(x)_SMS == 2) ? "read-write" : "read-only"}	Source Master Select. Identifies the Master Interface layer from which the source device (peripheral or memory) is accessed. ■ 0: AXI master 1 ■ 1: AXI Master 2 **Values:** ■ 0x1 (MASTER2_INTF): Source device on Master-2 interface layer ■ 0x0 (MASTER1_INTF): Source device on Master-1 interface layer **Value After Reset:** {(DMAX_CH(x)_SMS < 2) ? DMAX_CH(x)_SMS : 0} **Exists:** Always

5.2.5 CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** This register contains fields that configure the DMA transfer. This register should be programmed prior to enabling the channel.

  Bits [63:32] of the channel configuration register remains fixed for all blocks of a multi-block transfer and can be programmed only when channel is disabled.

  Bits [3:0] of the channel configuration register can be programmed even when channel is enabled.

  Software clears these bits to end the multi-block transfers. For Contiguous-Address and Auto-Reloading-based multi-block transfers (if neither source nor destination peripheral uses Shadow-Register or Linked-List-based multi-block transfers), if the corresponding multi-block type selection bits namely CHx_CFG.SRC_MLTBLK_TYPE and / or CHx_CFG.DST_MLTBLK_TYPE bits are seen to be 2'b00 at the end of a block transfer, the DW_axi_dmac understands that the previous block was the final block in the transfer and completes the DMA transfer operation.
- **Size:** 64 bits
- **Offset:** 0x120 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x && DMAX_NUM_CHANNELS <= 8 && DMAX_NUM_HS_IF <= 16

63	62:59	58:55	54:53	52	51:49	48	47:y	x:44	43	42:y	x:39	38	37	36	35	34:32	31:4	3:2	1:0
RSVD_DMAC_CHx_CFG_63	DST_OSR_LMT	SRC_OSR_LMT	LOCK_CH_L	LOCK_CH	CH_PRIOR	RSVD_DMAC_CHx_CFG_48	RSVD_DMAC_CHx_CFG_47_44	DST_PER	RSVD_DMAC_CHx_CFG_43	RSVD_DMAC_CHx_CFG_42_39	SRC_PER	DST_HWHS_POL	SRC_HWHS_POL	HS_SEL_DST	HS_SEL_SRC	TT_FC	RSVD_DMAC_CHx_CFG_4to31	DST_MULTBLK_TYPE	SRC_MULTBLK_TYPE

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63	RSVD_DMAC_CHx_CFG_63	R	DMAC Channelx Transfer Configuration Register (63bit) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
62:59	DST_OSR_LMT	R/W	Destination Outstanding Request Limit ■ Maximum outstanding request supported is 16. ■ Source Outstanding Request Limit = DST_OSR_LMT + 1 **Value After Reset:** 0x0 **Exists:** Always
58:55	SRC_OSR_LMT	R/W	Source Outstanding Request Limit ■ Maximum outstanding request supported is 16. ■ Source Outstanding Request Limit = SRC_OSR_LMT + 1 **Value After Reset:** 0x0 **Exists:** Always
54:53	LOCK_CH_L	{(DMAX_CH(x)_LOCK_EN == 1) ? "read-write" : "read-only"}	Channel Lock Level This bit indicates the duration over which CHx_CFG.LOCK_CH bit applies. ■ 00: Over complete DMA transfer ■ 01: Over DMA block transfer ■ 1x: Reserved This field does not exist if the configuration parameter DMAX_CHx_LOCK_EN is set to False; in that case, the read-back value is always 0. **Values:** ■ 0x0 (DMA_transfer_CH_LOCK): Duration of the Channel locking is for the entire DMA transfer ■ 0x1 (BLOCK_TRANFER_CH_LOCK): Duration of the Channel locking is for the current block transfer **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
52	LOCK_CH	{(DMAX_CH(x)_LOCK_EN == 1) ? "read-write" : "read-only"}	Channel Lock bit When the channel is granted control of the master bus interface and if the CHx_CFG.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CHx_CFG.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CHx_CFG.LOCK_CH_L. This field does not exist if the configuration parameter DMAX_CHx_LOCK_EN is set to False; in this case, the read-back value is always 0. Locking the channel locks AXI Read Address, Write Address and Write Data channels on the corresponding master interface. **Note**: Channel locking feature is supported only for memory-to-memory transfer at Block Transfer and DMA Transfer levels. Hardware does not check for the validity of channel locking setting, hence the software must take care of enabling the channel locking only for memory-to-memory transfers at Block Transfer or DMA Transfer levels. Illegal programming of channel locking might result in unpredictable behavior. **Values:** ■ 0x0 (NO_CHANNEL_LOCK): Channel is not locked during the transfers ■ 0x1 (CHANNEL_LOCK): Channel is locked and granted exclusive access to the Master Bus Interface **Value After Reset:** 0x0 **Exists:** Always
51:49	CH_PRIOR	R/W	Channel Priority A priority of DMAX_NUM_CHANNELS-1 is the highest priority, and 0 is the lowest. This field must be programmed within the following range: 0: DMAX_NUM_CHANNELS-1 A programmed value outside this range will cause erroneous behavior. **Value After Reset:** (DMAX_NUM_CHANNELS -(x)) **Exists:** Always
48	RSVD_DMAC_CHx_CFG_48	R	DMAC Channelx Transfer Configuration Register (48bit) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
47:y	RSVD_DMAC_CHx_CFG_47_44	R	DMAC Channelx Transfer Configuration Register (bits (LOG2_DMAX_NUM_HS_IF+44) to 47) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 44
x:44	DST_PER	* Varies	Assigns a hardware handshaking interface (0 - DMAX_NUM_HS_IF-1) to the destination of Channelx if the CHx_CFG.HS_SEL_DST field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface.**Note**: For correct DW_axi_dmac operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. This field does not exist if the configuration parameter DMAX_NUM_HS_IF is set to 0. x = 44 if DMAC_NUM_HS_IF is 1 x = ceil(log2(DMAC_NUM_HS_IF)) + 43 if DMAC_NUM_HS_IF is greater than 1 Bits 47: (x+1) do not exist and return 0 on a read. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** LOG2_DMAX_NUM_HS_IF + 43 **Memory Access:** {(DMAX_NUM_HS_IF == 0) ? "read-only" : "read-write"}
43	RSVD_DMAC_CHx_CFG_43	R	DMAC Channelx Transfer Configuration Register (43bit) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always
42:y	RSVD_DMAC_CHx_CFG_42_39	R	DMAC Channelx Transfer Configuration Register (bits (LOG2_DMAX_NUM_HS_IF+39) to 42) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 39

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
x:39	SRC_PER	* Varies	Assigns a hardware handshaking interface (0 - DMAX_NUM_HS_IF-1) to the source of Channelx if the CHx_CFG.HS_SEL_SRC field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface.**Note**: For correct DW_axi_dmac operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. This field does not exist if the configuration parameter DMAX_NUM_HS_IF is set to 0. x = 39 if DMAC_NUM_HS_IF is 1 x = ceil(log2(DMAC_NUM_HS_IF) + 38 if DMAC_NUM_HS_IF is greater than 1. Bits 42: (x+1) do not exist and return 0 on a read. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** LOG2_DMAX_NUM_HS_IF + 38 **Memory Access:** {(DMAX_NUM_HS_IF == 0) ? "read-only" : "read-write"}
38	DST_HWHS_POL	R	Destination Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Values:** ■ 0x0 (ACTIVE_HIGH): Polarity of the Handshaking Interface used for the Destination peripheral is Active High ■ 0x1 (ACTIVE_LOW): Polarity of the Handshaking Interface used for the Destination peripheral is Active Low **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
37	SRC_HWHS_POL	R	Source Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Values:** ■ 0x0 (ACTIVE_HIGH): Polarity of the Handshaking Interface used for the Source peripheral is Active High ■ 0x1 (ACTIVE_LOW): Polarity of the Handshaking Interface used for the Source peripheral is Active Low **Value After Reset:** 0x0 **Exists:** Always
36	HS_SEL_DST	R/W	Destination Software or Hardware Handshaking Select. This register selects which of the handshaking interfaces (hardware or software) is active for destination requests on this channel. ■ 0: Hardware handshaking interface. Software-initiated transaction requests are ignored. ■ 1: Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored. **Values:** ■ 0x0 (HARDWARE_HS): Hardware Handshaking Interface is used for the Destination peripheral ■ 0x1 (SOFTWARE_HS): Software Handshaking Interface is used for the Destination peripheral **Value After Reset:** 0x1 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
35	HS_SEL_SRC	R/W	Source Software or Hardware Handshaking Select. This register selects which of the handshaking interfaces (hardware or software) is active for source requests on this channel. ■ 0: Hardware handshaking interface. Software-initiated transaction requests are ignored. ■ 1: Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored. **Values:** ■ 0x0 (HARDWARE_HS): Hardware Handshaking Interface is used for the Source peripheral ■ 0x1 (SOFTWARE_HS): Software Handshaking Interface is used for the Source peripheral **Value After Reset:** 0x1 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
34:32	TT_FC	{(DMAX_CH(x)_TT_FC == 8) ? "read-write" : "read-only"}	Transfer Type and Flow Control. The following transfer types are supported. ■ Memory to Memory ■ Memory to Peripheral ■ Peripheral to Memory ■ Peripheral to Peripheral Flow Control can be assigned to the DW_axi_dmac, the source peripheral, or hte destination peripheral. **Values:** ■ 0x0 (MEM_TO_MEM_DMAC): Transfer Type is memory to memory and Flow Controller is DW_axi_dmac ■ 0x1 (MEM_TO_PER_DMAC): Transfer Type is memory to peripheral and Flow Controller is DW_axi_dmac ■ 0x2 (PER_TO_MEM_DMAC): Transfer Type is peripheral to memory and Flow Controller is DW_axi_dmac ■ 0x3 (PER_TO_PER_DMAC): Transfer Type is peripheral to peripheral and Flow Controller is DW_axi_dmac ■ 0x4 (PER_TO_MEM_SRC): Transfer Type is peripheral to Memory and Flow Controller is Source peripheral ■ 0x5 (PER_TO_PER_SRC): Transfer Type is peripheral to peripheral and Flow Controller is Source peripheral ■ 0x6 (MEM_TO_PER_DST): Transfer Type is memory to peripheral and Flow Controller is Destination peripheral ■ 0x7 (PER_TO_PER_DST): Transfer Type is peripheral to peripheral and Flow Controller is Destination peripheral **Value After Reset:** {(DMAX_CH(x)_TT_FC < 8) ? DMAX_CH(x)_TT_FC : 3} **Exists:** Always
31:4	RSVD_DMAC_CHx_CFG_4to31	R	DMAC Channelx Transfer Configuration Register (bits 4to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
3:2	DST_MULTBLK_TYPE	{(DMAX_CH(x)_MULTI_BLK_EN == 1) && ((DMAX_CH(x)_MULTI_BLK_TYPE == 0) || (DMAX_CH(x)_MULTI_BLK_TYPE == 1) || (DMAX_CH(x)_MULTI_BLK_TYPE == 4)) ? "read-write" : "read-only"}	Destination Multi Block Transfer Type. These bits define the type of multi-block transfer used for destination peripheral. ■ 00: Contiguous ■ 01: Reload ■ 10: Shadow Register ■ 11: Linked List If the type selected is Contiguous, the CHx_DAR register is loaded with the value of the end source address of previous block + 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Reload, the CHx_DAR register is reloaded from the initial value of DAR at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Shadow Register, the CHx_DAR register is loaded from the content of its shadow register if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Linked List, the CHx_DAR register is loaded from the Linked List if CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. CHx_CTL and CHx_BLOCK_TS registers are loaded from their initial values or from the contents of their shadow registers (if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1) or from the linked list (if CTL.ShadowReg_Or_LLI_Valid bit is set to 1) at the end of every block for multi-block transfers based on the multi-block transfer type programmed for source and destination peripherals. Contiguous transfer on both source and destination peripheral is not a valid multi-block transfer configuration.

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
			This field does not exist if the configuration parameter DMAX_CHx_MULTI_BLK_EN is not selected; in that case, the read-back value is always 0. **Values:** ■ 0x0 (CONTINGUOUS): Contiguous Multiblock Type used for Destination Transfer ■ 0x1 (RELOAD): Reload Multiblock Type used for Destination Transfer ■ 0x2 (SHADOW_REGISTER): Shadow Register based Multiblock Type used for Destination Transfer ■ 0x3 (LINKED_LIST): Linked List based Multiblock Type used for Destination Transfer **Value After Reset:** {(DMAX_CH(x)_MULTI_BLK_TYPE == 0) ? 0 : DMAX_CH(x)_DST_MULTI_BLK_TYPE} **Exists:** Always

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
1:0	SRC_MULTBLK_TYPE	{(DMAX_CH(x)_MULTI_BLK_EN == 1) && ((DMAX_CH(x)_MULTI_BLK_TYPE == 0) || (DMAX_CH(x)_MULTI_BLK_TYPE == 3) || (DMAX_CH(x)_MULTI_BLK_TYPE == 4)) ? "read-write" : "read-only"}	Source Multi Block Transfer Type. These bits define the type of multi-block transfer used for source peripheral. ■ 00: Contiguous ■ 01: Reload ■ 10: Shadow Register ■ 11: Linked List If the type selected is Contiguous, the CHx_SAR register is loaded with the value of the end source address of previous block + 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Reload, the CHx_SAR register is reloaded from the initial value of SAR at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Shadow Register, the CHx_SAR register is loaded from the content of its shadow register if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Linked List, the CHx_SAR register is loaded from the Linked List if CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. CHx_CTL and CHx_BLOCK_TS registers are loaded from their initial values or from the contents of their shadow registers (if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1) or from the linked list (if CTL.ShadowReg_Or_LLI_Valid bit is set to 1) at the end of every block for multi-block transfers based on the multi-block transfer type programmed for source and destination peripherals. Contiguous transfer on both source and destination peripheral is not a valid multi-block transfer configuration.

Table 5-26 Fields for Register: CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
			This field does not exist if the configuration parameter DMAX_CHx_MULTI_BLK_EN is not selected; in that case, the read-back value is always 0. **Values:** ■ 0x0 (CONTINGUOUS): Contiguous Multiblock Type used for Source Transfer ■ 0x1 (RELOAD): Reload Multiblock Type used for Source Transfer ■ 0x2 (SHADOW_REGISTER): Shadow Register based Multiblock Type used for Source Transfer ■ 0x3 (LINKED_LIST): Linked List based Multiblock Type used for Source Transfer **Value After Reset:** {(DMAX_CH(x)_MULTI_BLK_TYPE == 0) ? 0 : DMAX_CH(x)_SRC_MULTI_BLK_TYPE} **Exists:** Always

5.2.6 CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** This register contains fields that configure the DMA transfer. This register should be programmed prior to enabling the channel.

  Bits [63:32] of the channel configuration register remains fixed for all blocks of a multi-block transfer and can be programmed only when channel is disabled.

  Bits [3:0] of the channel configuration register can be programmed even when channel is enabled.

  Software clears these bits to end the multi-block transfers. For Contiguous-Address and Auto-Reloading-based multi-block transfers (if neither source nor destination peripheral uses Shadow-Register or Linked-List-based multi-block transfers), if the corresponding multi-block type selection bits namely CHx_CFG.SRC_MLTBLK_TYPE and / or CHx_CFG.DST_MLTBLK_TYPE bits are seen to be 2'b00 at the end of a block transfer, the DW_axi_dmac understands that the previous block was the final block in the transfer and completes the DMA transfer operation.
- **Size:** 64 bits
- **Offset:** 0x120 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= 1 && (DMAX_NUM_CHANNELS > 8 | | DMAX_NUM_HS_IF > 16)

63	62:59	58:55	54:53	52	51:47	46:39	38	37	36	35	34:32	31:18	17	16:y	x:11	10	9:y	x:4	3:2	1:0
RSVD_DMAC_CHx_CFG_63	DST_OSR_LMT	SRC_OSR_LMT	LOCK_CH_L	LOCK_CH	CH_PRIOR	RSVD_DMAC_CHx_CFG_39to46	DST_HWHS_POL	SRC_HWHS_POL	HS_SEL_DST	HS_SEL_SRC	TT_FC	RSVD_DMAC_CHx_CFG_18to31	RSVD_DMAC_CHx_CFG_17	RSVD_DMAC_CHx_CFG_16_11	DST_PER	RSVD_DMAC_CHx_CFG_10	RSVD_DMAC_CHx_CFG_9_4	SRC_PER	DST_MULTBLK_TYPE	SRC_MULTBLK_TYPE

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63	RSVD_DMAC_CHx_CFG_63	R	DMAC Channelx Transfer Configuration Register (63bit) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
62:59	DST_OSR_LMT	R/W	Destination Outstanding Request Limit ■ Maximum outstanding request supported is 16. ■ Source Outstanding Request Limit = DST_OSR_LMT + 1 **Value After Reset:** 0x0 **Exists:** Always
58:55	SRC_OSR_LMT	R/W	Source Outstanding Request Limit ■ Maximum outstanding request supported is 16. ■ Source Outstanding Request Limit = SRC_OSR_LMT + 1 **Value After Reset:** 0x0 **Exists:** Always
54:53	LOCK_CH_L	{(DMAX_CH(x)_LOCK_EN == 1) ? "read-write" : "read-only"}	Channel Lock Level This bit indicates the duration over which CHx_CFG.LOCK_CH bit applies. ■ 00: Over complete DMA transfer ■ 01: Over DMA block transfer ■ 1x: Reserved This field does not exist if the configuration parameter DMAX_CHx_LOCK_EN is set to False; in that case, the read-back value is always 0. **Values:** ■ 0x0 (DMA_transfer_CH_LOCK): Duration of the Channel locking is for the entire DMA transfer ■ 0x1 (BLOCK_TRANFER_CH_LOCK): Duration of the Channel locking is for the current block transfer **Value After Reset:** 0x0 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
52	LOCK_CH	{(DMAX_CH(x)_LOCK_EN == 1) ? "read-write" : "read-only"}	Channel Lock bit When the channel is granted control of the master bus interface and if the CHx_CFG.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CHx_CFG.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CHx_CFG.LOCK_CH_L. This field does not exist if the configuration parameter DMAX_CHx_LOCK_EN is set to False; in this case, the read-back value is always 0. Locking the channel locks AXI Read Address, Write Address and Write Data channels on the corresponding master interface. **Note**: Channel locking feature is supported only for memory-to-memory transfer at Block Transfer and DMA Transfer levels. Hardware does not check for the validity of channel locking setting, hence the software must take care of enabling the channel locking only for memory-to-memory transfers at Block Transfer or DMA Transfer levels. Illegal programming of channel locking might result in unpredictable behavior. **Values:** ■ 0x0 (NO_CHANNEL_LOCK): Channel is not locked during the transfers ■ 0x0 (CHANNEL_LOCK): Channel is locked and granted exclusive access to the Master Bus Interface **Value After Reset:** 0x0 **Exists:** Always
51:47	CH_PRIOR	R/W	Channel Priority A priority of DMAX_NUM_CHANNELS-1 is the highest priority, and 0 is the lowest. This field must be programmed within the following range: 0: DMAX_NUM_CHANNELS-1 A programmed value outside this range will cause erroneous behavior. **Value After Reset:** (DMAX_NUM_CHANNELS -(x)) **Exists:** Always
46:39	RSVD_DMAC_CHx_CFG_39to46	R	DMAC Channelx Transfer Configuration Register (bits 39to46) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
38	DST_HWHS_POL	R	Destination Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Values:** ■ 0x0 (ACTIVE_HIGH): Polarity of the Handshaking Interface used for the Destination peripheral is Active High ■ 0x1 (ACTIVE_LOW): Polarity of the Handshaking Interface used for the Destination peripheral is Active Low **Value After Reset:** 0x0 **Exists:** Always
37	SRC_HWHS_POL	R	Source Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Values:** ■ 0x0 (ACTIVE_HIGH): Polarity of the Handshaking Interface used for the Source peripheral is Active High ■ 0x1 (ACTIVE_LOW): Polarity of the Handshaking Interface used for the Source peripheral is Active Low **Value After Reset:** 0x0 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
36	HS_SEL_DST	R/W	Destination Software or Hardware Handshaking Select. This register selects which of the handshaking interfaces (hardware or software) is active for destination requests on this channel. ■ 0: Hardware handshaking interface. Software-initiated transaction requests are ignored. ■ 1: Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored. **Values:** ■ 0x0 (HARDWARE_HS): Hardware Handshaking Interface is used for the Destination peripheral ■ 0x1 (SOFTWARE_HS): Software Handshaking Interface is used for the Destination peripheral **Value After Reset:** 0x1 **Exists:** Always
35	HS_SEL_SRC	R/W	Source Software or Hardware Handshaking Select. This register selects which of the handshaking interfaces (hardware or software) is active for source requests on this channel. ■ 0: Hardware handshaking interface. Software-initiated transaction requests are ignored. ■ 1: Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored. **Values:** ■ 0x0 (HARDWARE_HS): Hardware Handshaking Interface is used for the Source peripheral ■ 0x1 (SOFTWARE_HS): Software Handshaking Interface is used for the Source peripheral **Value After Reset:** 0x1 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
34:32	TT_FC	{(DMAX_CH(x)_TT_FC == 8) ? "read-write" : "read-only"}	Transfer Type and Flow Control. The following transfer types are supported. ■ Memory to Memory ■ Memory to Peripheral ■ Peripheral to Memory ■ Peripheral to Peripheral Flow Control can be assigned to the DW_axi_dmac, the source peripheral, or hte destination peripheral. **Values:** ■ 0x0 (MEM_TO_MEM_DMAC): Transfer Type is memory to memory and Flow Controller is DW_axi_dmac ■ 0x1 (MEM_TO_PER_DMAC): Transfer Type is memory to peripheral and Flow Controller is DW_axi_dmac ■ 0x2 (PER_TO_MEM_DMAC): Transfer Type is peripheral to memory and Flow Controller is DW_axi_dmac ■ 0x3 (PER_TO_PER_DMAC): Transfer Type is peripheral to peripheral and Flow Controller is DW_axi_dmac ■ 0x4 (PER_TO_MEM_SRC): Transfer Type is peripheral to Memory and Flow Controller is Source peripheral ■ 0x5 (PER_TO_PER_SRC): Transfer Type is peripheral to peripheral and Flow Controller is Source peripheral ■ 0x6 (MEM_TO_PER_DST): Transfer Type is memory to peripheral and Flow Controller is Destination peripheral ■ 0x7 (PER_TO_PER_DST): Transfer Type is peripheral to peripheral and Flow Controller is Destination peripheral **Value After Reset:** {(DMAX_CH(x)_TT_FC < 8) ? DMAX_CH(x)_TT_FC : 3} **Exists:** Always
31:18	RSVD_DMAC_CHx_CFG_18to31	R	DMAC Channelx Transfer Configuration Register (bits 18to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
17	RSVD_DMAC_CHx_CFG_17	R	DMAC Channelx Transfer Configuration Register (bit 17) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
16:y	RSVD_DMAC_CHx_CFG_16_11	R	DMAC Channelx Transfer Configuration Register (bits (LOG2_DMAX_NUM_HS_IF+11) to 16) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 11
x:11	DST_PER	* Varies	Assigns a hardware handshaking interface (0 - DMAX_NUM_HS_IF-1) to the destination of Channelx if the CHx_CFG.HS_SEL_DST field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface.**Note**: For correct DW_axi_dmac operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. This field does not exist if the configuration parameter DMAX_NUM_HS_IF is set to 0. x = 11 if DMAC_NUM_HS_IF is 1 x = ceil(log2(DMAC_NUM_HS_IF)) + 10 if DMAC_NUM_HS_IF is greater than 1 Bits 16: (x+1) do not exist and return 0 on a read. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** LOG2_DMAX_NUM_HS_IF + 10 **Memory Access:** {(DMAX_NUM_HS_IF == 0) ? "read-only" : "read-write"}
10	RSVD_DMAC_CHx_CFG_10	R	DMAC Channelx Transfer Configuration Register (bit 10) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always
9:y	RSVD_DMAC_CHx_CFG_9_4	R	DMAC Channelx Transfer Configuration Register (bits (LOG2_DMAX_NUM_HS_IF+4) to 9) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 4

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
x:4	SRC_PER	* Varies	Assigns a hardware handshaking interface (0 - DMAX_NUM_HS_IF-1) to the source of Channelx if the CHx_CFG.HS_SEL_SRC field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface.**Note**: For correct DW_axi_dmac operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. This field does not exist if the configuration parameter DMAX_NUM_HS_IF is set to 0. x = 4 if DMAC_NUM_HS_IF is 1 x = ceil(log2(DMAC_NUM_HS_IF) + 3 if DMAC_NUM_HS_IF is greater than 1. Bits 9: (x+1) do not exist and return 0 on a read. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** LOG2_DMAX_NUM_HS_IF + 3 **Memory Access:** {(DMAX_NUM_HS_IF == 0) ? "read-only" : "read-write"}

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
3:2	DST_MULTBLK_TYPE	{(DMAX_CH(x)_MULTI_BLK_EN == 1) && ((DMAX_CH(x)_MULTI_BLK_TYPE == 0) || (DMAX_CH(x)_MULTI_BLK_TYPE == 1) || (DMAX_CH(x)_MULTI_BLK_TYPE == 4)) ? "read-write" : "read-only"}	Destination Multi Block Transfer Type. These bits define the type of multi-block transfer used for destination peripheral. ■ 00: Contiguous ■ 01: Reload ■ 10: Shadow Register ■ 11: Linked List If the type selected is Contiguous, the CHx_DAR register is loaded with the value of the end source address of previous block + 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Reload, the CHx_DAR register is reloaded from the initial value of DAR at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Shadow Register, the CHx_DAR register is loaded from the content of its shadow register if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Linked List, the CHx_DAR register is loaded from the Linked List if CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. CHx_CTL and CHx_BLOCK_TS registers are loaded from their initial values or from the contents of their shadow registers (if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1) or from the linked list (if CTL.ShadowReg_Or_LLI_Valid bit is set to 1) at the end of every block for multi-block transfers based on the multi-block transfer type programmed for source and destination peripherals. Contiguous transfer on both source and destination peripheral is not a valid multi-block transfer configuration.

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
			This field does not exist if the configuration parameter DMAX_CHx_MULTI_BLK_EN is not selected; in that case, the read-back value is always 0. **Values:** ■ 0x0 (CONTINGUOUS): Contiguous Multiblock Type used for Destination Transfer ■ 0x1 (RELOAD): Reload Multiblock Type used for Destination Transfer ■ 0x2 (SHADOW_REGISTER): Shadow Register based Multiblock Type used for Destination Transfer ■ 0x3 (LINKED_LIST): Linked List based Multiblock Type used for Destination Transfer **Value After Reset:** {(DMAX_CH(x)_MULTI_BLK_TYPE == 0) ? 0 : DMAX_CH(x)_DST_MULTI_BLK_TYPE} **Exists:** Always

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
1:0	SRC_MULTBLK_TYPE	{(DMAX_CH(x)_MULTI_BLK_EN == 1) && ((DMAX_CH(x)_MULTI_BLK_TYPE == 0) || (DMAX_CH(x)_MULTI_BLK_TYPE == 3) || (DMAX_CH(x)_MULTI_BLK_TYPE == 4)) ? "read-write" : "read-only"}	Source Multi Block Transfer Type. These bits define the type of multi-block transfer used for source peripheral. ■ 00: Contiguous ■ 01: Reload ■ 10: Shadow Register ■ 11: Linked List If the type selected is Contiguous, the CHx_SAR register is loaded with the value of the end source address of previous block + 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Reload, the CHx_SAR register is reloaded from the initial value of SAR at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Shadow Register, the CHx_SAR register is loaded from the content of its shadow register if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. If the type selected is Linked List, the CHx_SAR register is loaded from the Linked List if CTL.ShadowReg_Or_LLI_Valid bit is set to 1 at the end of every block for multi-block transfers. A new block transfer is then initiated. CHx_CTL and CHx_BLOCK_TS registers are loaded from their initial values or from the contents of their shadow registers (if CHx_CTL.ShadowReg_Or_LLI_Valid bit is set to 1) or from the linked list (if CTL.ShadowReg_Or_LLI_Valid bit is set to 1) at the end of every block for multi-block transfers based on the multi-block transfer type programmed for source and destination peripherals. Contiguous transfer on both source and destination peripheral is not a valid multi-block transfer configuration.

Table 5-27 Fields for Register: CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
			This field does not exist if the configuration parameter DMAX_CHx_MULTI_BLK_EN is not selected; in that case, the read-back value is always 0. **Values:** ■ 0x0 (CONTINGUOUS): Contiguous Multiblock Type used for Source Transfer ■ 0x1 (RELOAD): Reload Multiblock Type used for Source Transfer ■ 0x2 (SHADOW_REGISTER): Shadow Register based Multiblock Type used for Source Transfer ■ 0x3 (LINKED_LIST): Linked List based Multiblock Type used for Source Transfer **Value After Reset:** {(DMAX_CH(x)_MULTI_BLK_TYPE == 0) ? 0 : DMAX_CH(x)_SRC_MULTI_BLK_TYPE} **Exists:** Always

5.2.7 CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** This is the Linked List Pointer register. This register must be programmed to point to the first Linked List Item (LLI) in memory prior to enabling the channel if linked-list-based block chaining is enabled. This register is updated with new value of linked list pointer during the LLI update stage of dma transfer.
- **Size:** 64 bits
- **Offset:** 0x128 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

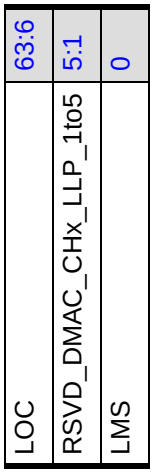


Table 5-28 Fields for Register: CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:6	LOC	R/W	Starting Address Memory of LLI block Starting Address In Memory of next LLI if block chaining is enabled. The six LSBs of the starting address are not stored because the address is assumed to be aligned to a 64-byte boundary. LLI access always uses the burst size (arsize/awsize) that is same as the data bus width and cannot be changed or programmed to anything other than this. Burst length (awlen/arlen) is chosen based on the data bus width so that the access does not cross one complete LLI structure of 64 bytes. DW_axi_dmac will fetch the entire LLI (40 bytes) in one AXI burst if the burst length is not limited by other settings. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-28 Fields for Register: CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
5:1	RSVD_DMAC_CHx_LLP_1to5	R	DMAC Channelx Linked List Pointer Register (bits 1to5) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	LMS	{(DMAX_CH(x)_LMS == 2) ? "read-write" : "read-only"}	LLI master Select This bit identifies the AXI layer/interface where the memory device that stores the next linked list item resides. ■ 0: AXI Master 1 ■ 1: AXI Master 2 This field does not exist if the configuration parameter DMAX_CHx_LMS is not set to NO_HARDCODE. In this case, the read-back value is always the hardcoded value. The maximum value of this field that can be read back is 'DMAX_NUM_MASTER_IF-1'. **Values:** ■ 0x0 (MASTER1_INTF): next Linked List item resides on AXI Master1 interface ■ 0x1 (MASTER2_INTF): next Linked List item resides on AXI Master2 interface **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.8 CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Status Register contains fields that indicate the status of DMA transfers for Channelx.
- **Size:** 64 bits
- **Offset:** 0x130 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

Table 5-29 Fields for Register: CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:47	RSVD_DMAC_CHx_STATUSREG_47to63	R	DMAC Channelx Status Register (bits 47to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-29 Fields for Register: CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
46:32	DATA_LEFT_IN_FIFO	R	Data Left in FIFO. This bit indicates the total number of data left in DW_axi_dmac channel FIFO after completing the current block transfer. The width of the data in channel FIFO is equal to CHx_CTL.SRC_TR_WIDTH. For normal block transfer completion without errors, Data_Left_In_FIFO = 0. If any error occurs during the dma transfer, the block transfer might be terminated early and in such a case, Data_Left_In_FIFO indicates the data remaining in channel FIFO which could not be transferred to destination peripheral. This field is cleared to zero on enabling the channel. **Note**: If CHx_CTL.DST_TR_WIDTH > CHx_CTL.SRC_TR_WIDTH, there may be residual data left in the FIFO which is not enough to form one CHx_CTL.SRC_TR_WIDTH of data and Data_Left_In_FIFO will return 0 in this case. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
31:22	RSVD_DMAC_CHx_STATUSREG_22to31	R	DMAC Channelx Status Register (bits 22to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
21:0	CMPLTD_BLK_TFR_SIZE	R	Completed Block Transfer Size. This bit indicates the total number of data of width CHx_CTL.SRC_TR_WIDTH transferred for the previous block transfer. For normal block transfer completion without any errors, this value will be equal to the value programmed in BLOCK_TS field of CHx_BLOCK_TS register. If any error occurs during the dma transfer, the block transfer might be terminated early and in such a case, this value indicates the actual data transferred without error in the current block. This field is cleared to zero on enabling the channel. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.9 CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Software handshake Source Register.
- **Size:** 64 bits
- **Offset:** 0x138 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

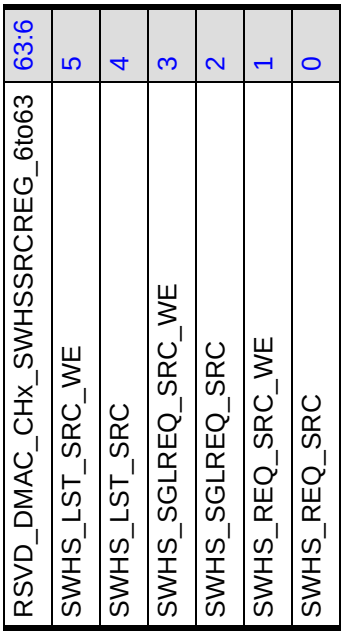


Table 5-30 Fields for Register: CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:6	RSVD_DMAC_CHx_SWHSSRCREG_6to63	R	DMAC Channelx Software Handshake Source Register (bits 6to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
5	SWHS_LST_SRC_WE	W	Write Enable bit for Software Handshake Last Request for Channel Source. **Values:** ■ 0x1 (ENABLE_SWHS_LAST_SRC): Enables write to the SWHS_LAST_SRC bit ■ 0x0 (DISABLE_SWHS_LAST_SRC): Disables write to the SWHS_LAST_SRC bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
4	SWHS_LST_SRC	R/W	Software Handshake Last Request for Channel Source. This bit is used to request LAST dma source data transfer if software handshaking method is selected for the source of the corresponding channel. This bit is ignored if software handshaking is not enabled for the source of the Channelx or if the source of Channelx is not the flow controller. CHx_SWHSSrcReg.SWHS_Req_Src bit must be set to 1 for DW_axi_dmac to treat it as a valid software handshaking request. If CHx_SWHSSrcReg.SWHS_SglReq_Src is set to 1, the LAST request is for SINGLE dma transaction (AXI burst length = 1), else the request is treated as a BURST transaction request. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSSrcReg.SWHS_Ack_Src bit and sees it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_Lst_Src bit is written only if the corresponding write enable bit, SWHS_Lst_Src_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSSrcReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_LAST_SRC): Source peripheral indication to dmac that the current transfer is the last transfer ■ 0x0 (INACTIVE_SWHS_LAST_SRC): Source peripheral indication that the curent transfer is not the last transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
3	SWHS_SGLREQ_SRC_WE	W	Write Enable bit for Software Handshake Single Request for Channel Source. **Values:** ■ 0x1 (ENABLE_SWHS_SGLREQ_SRC): Enables write to the SWHS_SGLREQ_SRC bit ■ 0x0 (DISABLE_SWHS_SGLREQ_SRC): Disables write to the SWHS_SGLREQ_SRC bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	SWHS_SGLREQ_SRC	R/W	Software Handshake Single Request for Channel Source. This bit is used to request SINGLE (AXI burst length = 1) dma source data transfer if software handshaking method is selected for the source of the corresponding channel. This bit is ignored if software handshaking is not enabled for the source of the Channelx. The functionality of this field depends on whether the peripheral is the flow controller. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSSrcReg.SWHS_Ack_Src bit and sees it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_SglReq_Src bit is written only if the corresponding write enable bit, SWHS_SglReq_Src_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSSrcReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_SGLREQ_SRC): Source peripheral request for a single dma transfer ■ 0x0 (INACTIVE_SWHS_SGLREQ_SRC): Source peripheral is not requesting for a single transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

Bits	Name	Memory Access	Description
1	SWHS_REQ_SRC_WE	W	Write Enable bit for Software Handshake Request for Channel Source. **Note**: This bit always returns 0 on a read back. **Values:** ■ 0x1 (ENABLE_SWHS_REQ_SRC): Enables write to the SWHS_REQ_SRC bit ■ 0x0 (DISABLE_SWHS_REQ_SRC): Disables write to the SWHS_REQ_SRC bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	SWHS_REQ_SRC	R/W	Software Handshake Request for Channel Source. This bit is used to request dma source data transfer if software handshaking method is selected for the source of the corresponding channel. This bit is ignored if software handshaking is not enabled for the source of the Channelx. The functionality of this field depends on whether the peripheral is the flow controller or not. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSSrcReg.SWHS_Ack_Src bit and sees it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_Req_Src bit is written only if the corresponding write enable bit, SWHS_Req_Src_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSSrcReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_REQ_SRC): Source peripheral request for a dma transfer ■ 0x0 (INACTIVE_SWHS_REQ_SRC): Source peripheral is not requesting for a burst transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.10 CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Software handshake Destination Register.
- **Size:** 64 bits
- **Offset:** 0x140 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

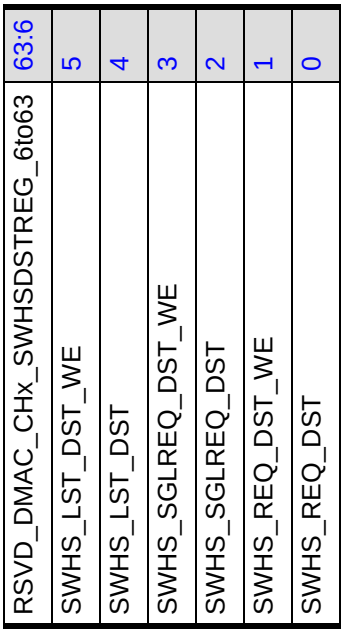


Table 5-31 Fields for Register: CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:6	RSVD_DMAC_CHx_SWHSDSTREG_6to63	R	DMAC Channelx Software Handshake Destination Register (bits 6to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-31 Fields for Register: CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
5	SWHS_LST_DST_WE	W	Write Enable bit for Software Handshake Last Request for Channel Destination. **Note**: This bit always returns 0 on a read back. **Values:** ■ 0x1 (ENABLE_SWHS_LAST_DST): Enables write to the SWHS_LAST_DST bit ■ 0x0 (DISABLE_SWHS_LAST_DST): Disables write to the SWHS_LAST_DST bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-31 Fields for Register: CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
4	SWHS_LST_DST	R/W	Software Handshake Last Request for Channel Destination. This bit is used to request LAST dma destination data transfer if software handshaking method is selected for the destination of the corresponding channel. This bit is ignored if software handshaking is not enabled for the destination of the Channelx or if the destination of Channelx is not the flow controller. CHx_SWHSDstReg.SWHS_Req_Dst bit must be set to 1 for DW_axi_dmac to treat it as a valid software handshaking request. If CHx_SWHSDstReg.SWHS_SglReq_Dst is set to 1, the LAST request is for SINGLE dma transaction (AXI burst length = 1), else the request is treated as a BURST transaction request. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSDstReg.SWHS_Ack_Dst bit and sets it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_Lst_Src bit is written only if the corresponding write enable bit, SWHS_Lst_Src_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSDstReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_LAST_DST): Destination peripheral indication to dmac that the current transfer is the last transfer ■ 0x0 (INACTIVE_SWHS_LAST_DST): Destination peripheral indication that the current transfer is not the last transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-31 Fields for Register: CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
3	SWHS_SGLREQ_DST_WE	W	Write Enable bit for Software Handshake Single Request for Channel Destination. **Note**: This bit always returns 0 on a read block. **Values:** ■ 0x1 (ENABLE_SWHS_SGLREQ_DST): Enables write to the SWHS_SGLREQ_DST bit ■ 0x0 (DISABLE_SWHS_SGLREQ_DST): Disables write to the SWHS_SGLREQ_DST bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	SWHS_SGLREQ_DST	R/W	Software Handshake Single Request for Channel Destination. This bit is used to request SINGLE (AXI burst length = 1) dma destination data transfer if software handshaking method is selected for the destination of the corresponding channel. This bit is ignored if software handshaking is not enabled for the destination of the Channelx. The functionality of this field depends on whether the peripheral is the flow controller. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSDstReg.SWHS_Ack_Dst bit and sees it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_SglReq_Dst bit is written only if the corresponding write enable bit, SWHS_SglReq_Dst_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSDstReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_SGLREQ_DST): Destination peripheral request for a single dma transfer ■ 0x0 (INACTIVE_SWHS_SGLREQ_DST): Destination peripheral is not requesting for a single transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-31 Fields for Register: CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
1	SWHS_REQ_DST_WE	W	Write Enable bit for Software Handshake Request for Channel Destination. **Note**: This bit always returns 0 on a read block. **Values:** ■ 0x1 (ENABLE_SWHS_REQ_DST): Enables write to the SWHS_REQ_DST bit ■ 0x0 (DISABLE_SWHS_REQ_DST): Disables write to the SWHS_REQ_DST bit **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	SWHS_REQ_DST	R/W	Software Handshake Request for Channel Destination. This bit is used to request dma destination data transfer if software handshaking method is selected for the destination of the corresponding channel. This bit is ignored if software handshaking is not enabled for the source of the Channelx. The functionality of this field depends on whether the peripheral is the flow controller. DW_axi_dmac clears this bit to 0 once software reads CHx_SWHSDstReg.SWHS_Ack_Dst bit and sees it as 1. Software can only set this bit to 1; it is not allowed to clear this bit to 0; only DW_axi_dmac can clear this bit. **Note**: SWHS_Req_Dst bit is written only if the corresponding write enable bit, SWHS_Req_Dst_WE is asserted on the same register write operation and if the Channelx is enabled in the DMAC_ChEnReg register. This allows software to set a bit in the CHx_SWHSDstReg register without performing a read-modified write operation. **Values:** ■ 0x1 (ACTIVE_SWHS_REQ_DST): Destination peripheral request for a dma transfer ■ 0x0 (INACTIVE_SWHS_REQ_DST): Destination peripheral is not requesting for a burst transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.11 CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Block Transfer Resume Request Register. This register is used during Linked List or Shadow Register based multi-block transfer.
  - For Linked-List-based multi-block transfer, ShadowReg_Or_LLI_Valid bit in LLI.CHx_CTL indicates whether the linked list item fetched from the memory is valid (0: LLI is invalid, 1: LLI is valid). On noticing this bit as 0, DW_axi_dmac discards the LLI and generates ShadowReg_Or_LLI_Invalid_ERR Interrupt if the corresponding channel error interrupt mask bit is set to 0. This error condition causes the DW_axi_dmac to halt the corresponding channel gracefully. DW_axi_dmac waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability, before attempting another LLI read operation.
  - For Shadow-Register-based multi-block transfer, ShadowReg_Or_LLI_Valid bit in CHx_CTL register indicates whether the shadow register contents are valid (0: Shadow Register contents are invalid, 1: Shadow Register contents are valid). On noticing this bit as 0 during shadow register fetch phase, DW_axi_dmac discards the Shadow Register contents and generates ShadowReg_Or_LLI_Invalid_ERR Interrupt. DW_axi_dmac waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid shadow register availability, before attempting another shadow register fetch operation and continue the next block transfer.
- **Size:** 64 bits
- **Offset:** 0x148 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

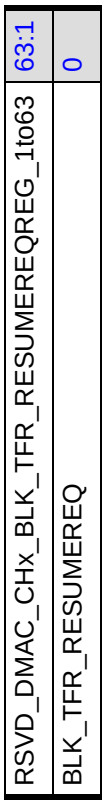


Table 5-32 Fields for Register: CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:1	RSVD_DMAC_CHx_BLK_TFR_RESUMEREQREG_1to63	W	DMAC Channelx Block Transfer Resume Request Register (bits 1to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
0	BLK_TFR_RESUMEREQ	W	Block Transfer Resume Request during Linked-List or Shadow-Register-based multi-block transfer. **Values:** ■ 0x0 (INACTIVE_BLK_TFR_RESUMEREQ): No request to resume the block transfer ■ 0x1 (ACTIVE_BLK_TFR_RESUMEREQ): Request for resuming the block transfer **Value After Reset:** 0x0 **Exists:** Always

5.2.12 CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx AXI ID Register. This register is allowed to be updated only when the channel is disabled, which means that it remains fixed for the entire DMA transfer.

  Note: The presence of this register is determined by the DMAC_M_ID_WIDTH and DMAC_NUM_CHANNELS configuration parameters.

  - If LLI is enabled for any of the channel, then the register is present only when:

  DMAX_M_ID_WIDTH - (log2(DMAC_NUM_CHANNELS) +1) > 0

  - Otherwise:

  DMAX_M_ID_WIDTH - log2(DMAC_NUM_CHANNELS) > 0

- **Size:** 64 bits
- **Offset:** 0x150 + (x-1)*0x100
- **Exists:** (DMAX_NUM_CHANNELS >= x) && (DMAX_AXI_ID_SUFFIX_WIDTH != 0)

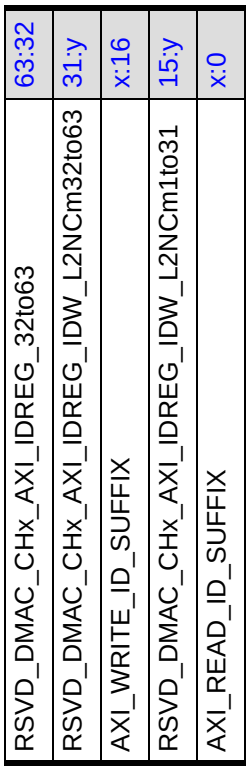


Table 5-33 Fields for Register: CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_AXI_IDREG_32to63	R	DMAC Channelx AXI ID Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
31:y	RSVD_DMAC_CHx_AXI_IDREG_IDW_L2NCm32to63	R	DMAC Channelx AXI ID Register (bits (IDW-L2NC-1)to32) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** DMAX_AXI_ID_SUFFIX_WIDTH + 16
x:16	AXI_WRITE_ID_SUFFIX	R/W	AXI Write ID Suffix. These bits form part of the AWID output of AXI3/AXI4 master interface. IDW = DMAX_M_ID_WIDTH L2NC = log2(DMAX_NUM_CHANNELS) The upper L2NC+1 bits of awidN is derived from the channel number which is currently accessing the master interface. This varies for LLI fetch and source data transfer. For source data transfer, awidN for channel1 4'b0000, awidN for channel8 4'b0111 and so on. For LLI fetch access, awidN for channel1 4'b1000, awidN for channel8 4'b1111 and so on. Lower bits are same as the value programmed in CHx_AXI_IDReg.AXI_Write_ID_Suffix filed. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** DMAX_AXI_ID_SUFFIX_WIDTH + 15
15:y	RSVD_DMAC_CHx_AXI_IDREG_IDW_L2NCm1to31	R	DMAC Channelx AXI ID Register (bits (IDW-L2NC-1)to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** DMAX_AXI_ID_SUFFIX_WIDTH

Table 5-33 Fields for Register: CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
x:0	AXI_READ_ID_SUFFIX	R/W	AXI Read ID Suffix These bits form part of the ARID output of AXI3/AXI4 master interface. IDW = DMAX_M_ID_WIDTH L2NC = log2(DMAX_NUM_CHANNELS) The upper L2NC+1 bits of aridN is derived from the channel number which is currently accessing the master interface. This varies for LLI fetch and source data transfer. For source data transfer, aridN for channel1 4'b0000, aridN for channel8 4'b0111 and so on. For LLI fetch access, aridN for channel1 4'b1000, aridN for channel8 4'b1111 and so on. Lower bits are same as the value programmed in CHx_AXI_IDReg.AXI_Read_ID_Suffix filed. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** DMAX_AXI_ID_SUFFIX_WIDTH - 1

5.2.13 CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx AXI QOS Register. This register is allowed to be updated only when the channel is disabled, which means that it remains fixed for the entire DMA transfer.
- **Size:** 64 bits
- **Offset:** 0x158 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

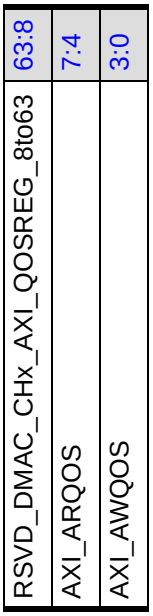


Table 5-34 Fields for Register: CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:8	RSVD_DMAC_CHx_AXI_QOSREG_8to63	R	DMAC Channelx AXI QOS Register (bits 8to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
7:4	AXI_ARQOS	* Varies	AXI ARQOS. These bits form the arqos output of AXI4 master interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(DMAX_HAS_QOS == 1) ? "read-write" : "read-only"}

Table 5-34 Fields for Register: CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
3:0	AXI_AWQOS	* Varies	AXI AWQOS. These bits form the awqos output of AXI4 master interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(DMAX_HAS_QOS == 1) ? "read-write" : "read-only"}

5.2.14 CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Source Status Register. After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the contents of the CHx_SSTATAR register. This status information is then stored in the CHx_SSTAT register and written out to the CHx_SSTAT register location of the LLI before the start of the next block.

  Source status write-back to the CHx_SSTAT register location of the LLI is performed only if DMAX_CHx_LLI_WB_EN = 1 and linked-list-based multi-block transfer is enabled for either source or destination peripheral of the channel.

  This register does not exist if DMAC_CHx_SRC_STAT_EN is set to False; in this case, the read-back value is always 0.
- **Size:** 64 bits
- **Offset:** 0x160 + (x-1)*0x100
- **Exists:** (DMAX_NUM_CHANNELS >= x) && (DMAX_CH1_SRC_STAT_EN == 1)

Table 5-35 Fields for Register: CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_SSTAT_32to63	R	DMAC Channelx Source Status Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-35 Fields for Register: CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
31:0	SSTAT	R	Source Status Source status information retrieved by hardware from the address pointed to by the contents of the CHx_SSTATAR register. Source peripheral should update the source status information, if any, at the location pointed to by CHx_SSTATAR to utilize this feature.This status is not related to any internal status of DW_axi_dmac. This status is not related to any internal status of DW_axi_dmac. **Value After Reset:** 0x0 **Exists:** Always

5.2.15 CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Destination Status Register. After each block transfer completes, hardware can retrieve the destination status information from the address pointed to by the contents of the CHx_DSTATAR register. This status information is then stored in the CHx_DSTAT register and written out to the CHx_DSTAT register location of the LLI before the start of the next block. Destination status write-back to the CHx_DSTAT register location of the LLI is performed only if DMAX_CHx_LLI_WB_EN = 1 and linked list based multi-block transfer is enabled for either source or destination peripheral of the channel.
- **Size:** 64 bits
- **Offset:** 0x168 + (x-1)*0x100
- **Exists:** (DMAX_NUM_CHANNELS >= x) && (DMAX_CH1_DST_STAT_EN == 1)

Table 5-36 Fields for Register: CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_DSTAT_32to63	R	DMAC Channelx Destination Status Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-36 Fields for Register: CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
31:0	DSTAT	R	Destination Status Destination status information retrieved by hardware from the address pointed to by the contents of the CHx_DSTATAR register. Destination peripheral should update the destination status information, if any, at the location pointed to by CHx_DSTATAR to utilize this feature. This status is not related to any internal status of DW_axi_dmac. **Value After Reset:** 0x0 **Exists:** Always

5.2.16 CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Source Status Fetch Register. After completion of each block transfer, hardware can retrieve the source status information from the user-defined address to which the contents of the CHx_SSTATAR register point. You can select any location in system memory that provides a 64-bit value to indicate the status of the source transfer.

  This register does not exist if DMAC_CHx_SRC_STAT_EN is set to False; in this case, the read-back value is always 0.
- **Size:** 64 bits
- **Offset:** 0x170 + (x-1)*0x100
- **Exists:** (DMAX_NUM_CHANNELS >= x) && (DMAX_CH1_SRC_STAT_EN == 1)

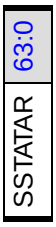


Table 5-37 Fields for Register: CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:0	SSTATAR	R/W	Source Status Fetch Address Pointer from where hardware can fetch the source status information, which is registered in the CHx_SSTAT register and written out to the CHx_SSTAT register location of the LLI before the start of the next block if DMAX_CHx_LLI_WB_EN = 1 and linked list based multi-block transfer is enabled for either source or destination peripheral of the channel. Source peripheral should update the source status information, if any, at the location pointed to by CHx_SSTATAR to utilize this feature. This status is not related to any internal status of DW_axi_dmac. **Value After Reset:** 0x0 **Exists:** Always

5.2.17 CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Destination Status Fetch Register. After completion of each block transfer, hardware can retrieve the destination status information from the user-defined address to which the contents of the CHx_DSTATAR register points. You can select any location in system memory that would provide a 64-bit value to indicate the status of the destination transfer.

  This register does not exist if DMAC_CHx_SRC_STAT_EN is set to False; in this case, the read-back value is always 0.
- **Size:** 64 bits
- **Offset:** 0x178 + (x-1)*0x100
- **Exists:** (DMAX_NUM_CHANNELS >= x) && (DMAX_CH1_DST_STAT_EN == 1)

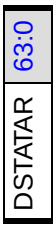


Table 5-38 Fields for Register: CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:0	DSTATAR	R/W	Destination Status Fetch Address Pointer from where hardware can fetch the Destination status information, which is registered in the CHx_DSTAT register and written out to the CHx_DSTAT register location of the LLI before the start of the next block if DMAX_CHx_LLI_WB_EN = 1 and linked list based multi-block transfer is enabled for either source or destination peripheral of the channel. Destination peripheral should update the destination status information, if any, at the location pointed to by CHx_DSTATAR to utilize this feature. This status is not related to any internal status of DW_axi_dmac. **Value After Reset:** 0x0 **Exists:** Always

5.2.18 CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Writing 1 to specific field enables the corresponding interrupt status generation in Channelx Interrupt Status Register(CH1_IntStatusReg).
- **Size:** 64 bits
- **Offset:** 0x180 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= 1

63:32	31	30	29	28	27	26:22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_32to63	Enable_CH_ABORTED_IntStat	Enable_CH_DISABLED_IntStat	Enable_CH_SUSPENDED_IntStat	Enable_CH_SRC_SUSPENDED_IntStat	Enable_CH_LOCK_CLEARED_IntStat	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to26	Enable_SLVIF_WRONHOLD_ERR_IntStat	Enable_SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	Enable_SLVIF_WRONCHEN_ERR_IntStat	Enable_SLVIF_RD2RWO_ERR_IntStat	Enable_SLVIF_WR2RO_ERR_IntStat	Enable_SLVIF_DEC_ERR_IntStat	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_15	Enable_SLVIF_MULTIBLKTYPE_ERR_IntStat	Enable_SHADOWREG_OR_LLI_INVALID_ERR_IntStat	Enable_LLI_WR_SLV_ERR_IntStat	Enable_LLI_RD_SLV_ERR_IntStat	Enable_LLI_WR_DEC_ERR_IntStat	Enable_LLI_RD_DEC_ERR_IntStat	Enable_DST_SLV_ERR_IntStat	Enable_SRC_SLV_ERR_IntStat	Enable_DST_DEC_ERR_IntStat	Enable_SRC_DEC_ERR_IntStat	Enable_DST_TRANSCOMP_IntStat	Enable_SRC_TRANSCOMP_IntStat	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_2	Enable_DMA_TFR_DONE_IntStat	Enable_BLOCK_TFR_DONE_IntStat

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_32to63	R	DMAC Channelx Interrupt Status Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0xffffffff **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
31	Enable_CH_ABORTED_IntStat	R/W	Channel Aborted Status Enable. ■ 0: Disable the generation of Channel Aborted Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel Aborted Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_ABORTED): Enable the generation of Channel Aborted Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_ABORTED): Disable the generation of Channel Aborted Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
30	Enable_CH_DISABLED_IntStat	R/W	Channel Disabled Status Enable. ■ 0: Disable the generation of Channel Disabled Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel Disabled Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_DISABLED): Enable the generation of Channel Disabled Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_DISABLED): Disable the generation of Channel Disabled Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
29	Enable_CH_SUSPENDED_IntStat	R/W	Channel Suspended Status Enable. ■ 0: Disable the generation of Channel Suspended Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel Suspended Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_SUSPENDED): Enable the generation of Channel Suspended Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_SUSPENDED): Disable the generation of Channel Suspended Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
28	Enable_CH_SRC_SUSPENDED_IntStat	R/W	Channel Source Suspended Status Enable. ■ 0: Disable the generation of Channel Source Suspended Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel Source Suspended Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_SRC_SUSPENDED): Enable the generation of Channel Source Suspended Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_SRC_SUSPENDED): Disable the generation of Channel Source Suspended Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
27	Enable_CH_LOCK_CLEARED_IntStat	R/W	Channel Lock Cleared Status Enable. ■ 0: Disable the generation of Channel LOCK CLEARED Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel LOCK CLEARED Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_LOCK_CLEARED): Enable the generation of Channel LOCK CLEARED Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_LOCK_CLEARED): Disable the generation of Channel LOCK CLEARED Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
26:22	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to26	R	DMAC Channelx Interrupt Status Register (bits 22to26) Reserved bits - Read Only **Value After Reset:** 0x1f **Exists:** Always
21	Enable_SLVIF_WRONHOLD_ERR_IntStat	R/W	Slave Interface Write On Hold Error Status Enable. ■ 0: Disable the generation of Slave Interface Write On Hold Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Write On Hold Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WRONHOLD_ERR): Enable the generation of Slave Interface Write On Hold Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WRONHOLD_ERR): Disable the generation of Slave Interface Write On Hold Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
20	Enable_SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	R/W	Shadow Register Write On Valid Error Status Enable. ■ 0: Disable the generation of Shadow Register Write On Valid Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Shadow register Write On Valid Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR): Enable the generation of Shadow register Write On Valid Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_SHADOWREG_WRON_VALID_ERR): Disable the generation of Shadow Register Write On Valid Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
19	Enable_SLVIF_WRONCHEN_ERR_IntStat	R/W	Slave Interface Write On Channel Enabled Error Status Enable. ■ 0: Disable the generation of Slave Interface Write On Channel enabled Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Write On Channel enabled Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WRONCHEN_ERR): Enable the generation of Slave Interface Write On Channel enabled Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WRONCHEN_ERR): Disable the generation of Slave Interface Write On Channel enabled Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
18	Enable_SLVIF_RD2RWO_ERR_IntStat	R/W	Slave Interface Read to write Only Error Status Enable. ■ 0: Disable the generation of Slave Interface Read to Write only Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Read to Write Only Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_RD2RWO_ERR): Enable the generation of Slave Interface Read to Write Only Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_RD2RWO_ERR): Disable the generation of Slave Interface Read to Write only Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
17	Enable_SLVIF_WR2RO_ERR_IntStat	R/W	Slave Interface Write to Read Only Error Status Enable. ■ 0: Disable the generation of Slave Interface Write to Read only Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Write to Read Only Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WR2RO_ERR): Enable the generation of Slave Interface Write to Read Only Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WR2RO_ERR): Disable the generation of Slave Interface Write to Read only Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
16	Enable_SLVIF_DEC_ERR_IntStat	R/W	Slave Interface Decode Error Status Enable. ■ 0: Disable the generation of Slave Interface Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_DEC_ERR): Enable the generation of Slave Interface Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_DEC_ERR): Disable the generation of Slave Interface Decode Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
15	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_15	R	DMAC Channelx Interrupt Status Register (bit 15) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always
14	Enable_SLVIF_MULTIBLKTYPE_ERR_IntStat	R/W	Slave Interface Multi Block type Error Status Enable. ■ 0: Disable the generation of Slave Interface Multi Block type Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Slave Interface Multi Block type Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_MULTIBLKTYPE_ERR): Enable the generation of Slave Interface Multi Block type Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_MULTIBLKTYPE_ERR): Disable the generation of Slave Interface Multi Block type Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
13	Enable_SHADOWREG_OR_LLI_INVALID_ERR_IntStat	R/W	Shadow register or LLI Invalid Error Status Enable. ■ 0: Disable the generation of Shadow Register or LLI Invalid Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Shadow Register or LLI Invalid Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SHADOWREG_OR_LLI_INVALID_ERR): Enable the generation of Shadow Register or LLI Invalid Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SHADOWREG_OR_LLI_INVALID_ERR): Disable the generation of Shadow Register or LLI Invalid Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
12	Enable_LLI_WR_SLV_ERR_IntStat	R/W	LLI WRITE Slave Error Status Enable. ■ 0: Disable the generation of LLI WRITE Slave Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI WRITE Slave Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_WR_SLV_ERR): Enable the generation of LLI WRITE Slave Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_WR_SLV_ERR): Disable the generation of LLI WRITE Slave Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
11	Enable_LLI_RD_SLV_ERR_IntStat	R/W	LLI Read Slave Error Status Enable. ■ 0: Disable the generation of LLI Read Slave Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI Read Slave Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_RD_SLV_ERR): Enable the generation of LLI Read Slave Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_RD_SLV_ERR): Disable the generation of LLI Read Slave Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
10	Enable_LLI_WR_DEC_ERR_IntStat	R/W	LLI WRITE Decode Error Status Enable. ■ 0: Disable the generation of LLI WRITE Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI WRITE Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_WR_DEC_ERR): Enable the generation of LLI WRITE Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_WR_DEC_ERR): Disable the generation of LLI WRITE Decode Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
9	Enable_LLI_RD_DEC_ERR_IntStat	R/W	LLI Read Decode Error Status Enable. ■ 0: Disable the generation of LLI Read Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI Read Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_RD_DEC_ERR): Enable the generation of LLI Read Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_RD_DEC_ERR): Disable the generation of LLI Read Decode Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
8	Enable_DST_SLV_ERR_IntStat	R/W	Destination Slave Error Status Enable. ■ 0: Disable the generation of Destination Slave Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Destination Slave Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_DST_SLV_ERR): Enable the generation of Destination Slave Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_DST_SLV_ERR): Disable the generation of Destination Slave Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
7	Enable_SRC_SLV_ERR_IntStat	R/W	Source Slave Error Status Enable. ■ 0: Disable the generation of Source Slave Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Source Slave Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SRC_SLV_ERR): Enable the generation of Source Slave Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SRC_SLV_ERR): Disable the generation of Source Slave Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
6	Enable_DST_DEC_ERR_IntStat	R/W	Destination Decode Error Status Enable. ■ 0: Disable the generation of Destination Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Destination Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_DST_DEC_ERR): Enable the generation of Destination Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_DST_DEC_ERR): Disable the generation of Destination Decode Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
5	Enable_SRC_DEC_ERR_IntStat	R/W	Source Decode Error Status Enable. ■ 0: Disable the generation of Source Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Source Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SRC_DEC_ERR): Enable the generation of Source Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SRC_DEC_ERR): Disable the generation of Source Decode Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
4	Enable_DST_TRANSCOMP_IntStat	R/W	Destination Transaction Completed Status Enable. ■ 0: Disable the generation of Destination Transaction complete Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Destination Transaction complete Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_DST_TRANSCOMP): Enable the generation of Destination Transaction complete Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_DST_TRANSCOMP): Disable the generation of Destination Transaction complete Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
3	Enable_SRC_TRANSCOMP_IntStat	R/W	Source Transaction Completed Status Enable. ■ 0: Disable the generation of Source Transaction Complete Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Source Transaction Complete Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SRC_TRANSCOMP): Enable the generation of Source Transaction Complete Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SRC_TRANSCOMP): Disable the generation of Source Transaction Complete Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
2	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_2	R	DMAC Channelx Interrupt Status Register (bit 2) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always
1	Enable_DMA_TFR_DONE_IntStat	R/W	DMA Transfer Done Interrupt Status Enable. ■ 0: Disable the generation of DMA Transfer Done Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of DMA Transfer Done Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_DMA_TFR_DONE): Enable the generation of DMA Transfer Done Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_DMA_TFR_DONE): Disable the generation of DMA Transfer Done Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Table 5-39 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
0	Enable_BLOCK_TFR_DONE_IntStat	R/W	Block Transfer Done Interrupt Status Enable. ■ 0: Disable the generation of Block Transfer Done Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Block Transfer Done Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_BLOCK_TFR_DONE): Enable the generation of Block Transfer Done Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_BLOCK_TFR_DONE): Disable the generation of Block Transfer Done Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

5.2.19 CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Channelx Interrupt Status Register captures the Channelx specific interrupts
- **Size:** 64 bits
- **Offset:** 0x188 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:32	31	30	29	28	27	26:22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_DMAC_CHx_INTSTATUSREG_32to63	CH_ABORTED_IntStat	CH_DISABLED_IntStat	CH_SUSPENDED_IntStat	CH_SRC_SUSPENDED_IntStat	CH_LOCK_CLEARED_IntStat	RSVD_DMAC_CHx_INTSTATUSREG_22to26	SLVIF_WRONHOLD_ERR_IntStat	SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	SLVIF_WRONCHEN_ERR_IntStat	SLVIF_RD2RWO_ERR_IntStat	SLVIF_WR2RO_ERR_IntStat	SLVIF_DEC_ERR_IntStat	RSVD_DMAC_CHx_INTSTATUSREG_15	SLVIF_MULTIBLKTYPE_ERR_IntStat	SHADOWREG_OR_LLI_INVALID_ERR_IntStat	LLI_WR_SLV_ERR_IntStat	LLI_RD_SLV_ERR_IntStat	LLI_WR_DEC_ERR_IntStat	LLI_RD_DEC_ERR_IntStat	DST_SLV_ERR_IntStat	SRC_SLV_ERR_IntStat	DST_DEC_ERR_IntStat	SRC_DEC_ERR_IntStat	DST_TRANSCOMP_IntStat	SRC_TRANSCOMP_IntStat	RSVD_DMAC_CHx_INTSTATUSREG_2	DMA_TFR_DONE_IntStat	BLOCK_TFR_DONE_IntStat

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_INTSTATUS REG_32to63	R	DMAC Channelx Specific Interrupt Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
31	CH_ABORTED_IntStat	R	Channel Aborted. This indicates to the software that the corresponding channel in DW_axi_dmac is aborted. ■ 0: Channel is not aborted ■ 1: Channel is aborted Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_CH_ABORTED): Channel is aborted ■ 0x0 (INACTIVE_CH_ABORTED): Channel is not aborted **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
30	CH_DISABLED_IntStat	R	Channel Disabled. This indicates to the software that the corresponding channel in DW_axi_dmac is disabled. ■ 0: Channel is not disabled. ■ 1: Channel is disabled. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_CH_DISABLED): Channel is disabled ■ 0x0 (INACTIVE_CH_DISABLED): Channel is not disabled **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
29	CH_SUSPENDED_IntStat	R	Channel Suspended. This indicates to the software that the corresponding channel in DW_axi_dmac is suspended. ■ 0: Channel is not suspended. ■ 1: Channel is suspended. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_CH_SUSPENDED): Channel is suspended ■ 0x0 (INACTIVE_CH_SUSPENDED): Channel is not suspended **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
28	CH_SRC_SUSPENDED_IntStat	R	Channel Source Suspended. This indicates to the software that the corresponding channel source data transfer in DW_axi_dmac is suspended. ■ 0: Channel source is not suspended ■ 1: Channel Source is suspended. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_CH_SRC_SUSPENDED): Channel Source is suspended ■ 0x0 (INACTIVE_CH_SRC_SUSPENDED): Channel source is not suspended **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
27	CH_LOCK_CLEARED_IntStat	R	Channel Lock Cleared. This indicates to the software that the locking of the corresponding channel in DW_axi_dmac is cleared. ■ 0: Channel locking is not cleared. ■ 1: Channel locking is cleared. Channel locking is cleared by DW_axi_dmac during the following situations: ■ Channel locking is cleared and the channel locking settings in CHx_CFG register is reset if DW_axi_dmac disables the channel upon request from software. ■ Channel locking is cleared and the channel locking settings in CHx_CFG register is reset if DW_axi_dmac disables the channel upon receiving error response on the master interface. This bit is cleared to 0 on enabling the channel. **Values:** ■ 0x1 (ACTIVE_CH_LOCK_CLEARED): Channel Locking is cleared ■ 0x0 (INACTIVE_CH_LOCK_CLEARED): Channel locking is not cleared, if present. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
26:22	RSVD_DMAC_CHx_INTSTATUS REG_22to26	R	DMAC Channelx Specific Interrupt Register (bits 22to26) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
21	SLVIF_WRONHOLD_ERR_IntStat	R	Slave Interface Write On Hold Error. This error occurs if an illegal write operation is performed on a register; this happens if a write operation is performed on a channel register when DW_axi_dmac is in Hold mode. ■ 0: No Slave Interface Write On Hold Errors. ■ 1: Slave Interface Write On Hold Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_WRONHOLD_ERR): Slave Interface Write On Hold Error detected ■ 0x0 (INACTIVE_SLVIF_WRONHOLD_ERR): No Slave Interface Write On Hold Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
20	SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	R	Shadow Register Write On Valid Error. This error occurs if shadow register based multi-block transfer is enabled and software tries to write to the shadow register when CHx_CTL.ShadowReg_Or_LLI_Valid bit is 1. ■ 0: No Slave Interface Shadow Register Write On Valid Errors. ■ 1: Slave Interface Shadow Register Write On Valid Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_SHADOWREG_WRON_VALID_ERR): Slave Interface Shadow Register Write On Valid Error detected ■ 0x0 (INACTIVE_SLVIF_SHADOWREG_WRON_VALID_ERR): No Slave Interface Shadow Register Write On Valid Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
19	SLVIF_WRONCHEN_ERR_IntStat	R	Slave Interface Write On Channel Enabled Error. This error occurs if an illegal write operation is performed on a register; this happens if a write operation is performed on a register when the channel is enabled and if it is not allowed for the corresponding register as per the DW_axi_dmac specification. ■ 0: No Slave Interface Write On Channel Enabled Errors. ■ 1: Slave Interface Write On Channel Enabled Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_WRONCHEN_ERR): Slave Interface Write On Channel Enabled Error detected ■ 0x0 (INACTIVE_SLVIF_WRONCHEN_ERR): No Slave Interface Write On Channel Enabled Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
18	SLVIF_RD2RWO_ERR_IntStat	R	Slave Interface Read to write Only Error. This error occurs if read operation is performed to a Write Only register. ■ 0: No Slave Interface Read to Write Only Errors. ■ 1: Slave Interface Read to Write Only Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_RD2RWO_ERR): Slave Interface Read to Write Only Error detected ■ 0x0 (INACTIVE_SLVIF_RD2RWO_ERR): No Slave Interface Read to Write Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
17	SLVIF_WR2RO_ERR_IntStat	R	Slave Interface Write to Read Only Error. This error occurs if write operation is performed to a Read Only register. ■ 0: No Slave Interface Write to Read Only Errors. ■ 1: Slave Interface Write to Read Only Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_WR2RO_ERR): Slave Interface Write to Read Only Error detected ■ 0x0 (INACTIVE_SLVIF_WR2RO_ERR): No Slave Interface Write to Read Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
16	SLVIF_DEC_ERR_IntStat	R	Slave Interface Decode Error. Decode Error generated by DW_axi_dmac during register access. This error occurs if the register access is to invalid address in Channelx register space resulting in error response by DW_axi_dmac slave interface. ■ 0: No Slave Interface Decode errors. ■ 1: Slave Interface Decode Error detected. Error Interrupt is generated if the corresponding bit in CHxINTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_DEC_ERR): Slave Interface Decode Error detected ■ 0x0 (INACTIVE_SLVIF_DEC_ERR): No Slave Interface Decode errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
15	RSVD_DMAC_CHx_INTSTATUS REG_15	R	DMAC Channelx Specific Interrupt Register (bit 15) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
14	SLVIF_MULTIBLKTYPE_ERR_Int Stat	R	Slave Interface Multi Block type Error. This error occurs if multi-block transfer type programmed in CHx_CFG register (SRC_MLTBLK_TYPE and DST_MLTBLK_TYPE) is invalid. This error condition causes the DW_axi_dmac to halt the corresponding channel gracefully; Error Interrupt is generated if the corresponding channel error interrupt mask bit is set to 0 and the channel waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid multi-block transfer type availability. ■ 0: No Multi-block transfer type Errors. ■ 1: Multi-block transfer type Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SLVIF_MULTIBLKTYPE_ERR): Multi-block transfer type Error detected ■ 0x0 (INACTIVE_SLVIF_MULTIBLKTYPE_ERR): No Multi-block transfer type Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
13	SHADOWREG_OR_LLI_INVALID_ERR_IntStat	R	Shadow register or LLI Invalid Error. This error occurs if CHx_CTL.ShadowReg_Or_LLI_Valid bit is seen to be 0 during DW_axi_dmac Shadow Register / LLI fetch phase. This error condition causes the DW_axi_dmac to halt the corresponding channel gracefully; Error Interrupt is generated if the corresponding channel error interrupt mask bit is set to 0 and the channel waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid Shadow Register availability. In the case of LLI pre-fetching, ShadowReg_Or_LLI_Invalid_ERR Interrupt is not generated even if ShadowReg_Or_LLI_Valid bit is seen to be 0 for the pre-fetched LLI. In this case, DW_axi_dmac re-attempts the LLI fetch operation after completing the current block transfer and generates ShadowReg_Or_LLI_Invalid_ERR Interrupt only if ShadowReg_Or_LLI_Valid bit is still seen to be 0. ■ 0: No Shadow Register / LLI Invalid errors. ■ 1: Shadow Register / LLI Invalid error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SHADOWREG_OR_LLI_INVALID_ERR): Shadow Register / LLI Invalid error detected ■ 0x0 (INACTIVE_SHADOWREG_OR_LLI_INVALID_ERR): No Shadow Register / LLI Invalid errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
12	LLI_WR_SLV_ERR_IntStat	R	LLI WRITE Slave Error. Slave Error detected by Master Interface during LLI write-back operation. This error occurs if the slave interface on which LLI resides issues a Slave Error. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN1 bit which received the error is set to 0. ■ 0: No LLI write Slave Errors. ■ 1: LLI Write SLAVE Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_LLI_WR_SLV): LLI Write SLAVE Error detected ■ 0x0 (INACTIVE_LLI_WR_SLV): No LLI write Slave Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
11	LLI_RD_SLV_ERR_IntStat	R	LLI Read Slave Error. Slave Error detected by Master Interface during LLI read operation. This error occurs if the slave interface on which LLI resides issues a Slave Error. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN1 bit which received the error is set to 0. ■ 0: No LLI Read Slave Errors. ■ 1: LLI read Slave Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_LLI_RD_SLV_ERR): LLI read Slave Error detected ■ 0x0 (INACTIVE_LLI_RD_SLV_ERR): No LLI Read Slave Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
10	LLI_WR_DEC_ERR_IntStat	R	LLI WRITE Decode Error. Decode Error detected by Master Interface during LLI write-back operation. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/slave. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN1 bit which received the error is set to 0. ■ 0: NO LLI Write Decode Errors. ■ 1: LLI write Decode Error detected. Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_LLI_WR_DEC_ERR): LLI write Decode Error detected ■ 0x0 (INACTIVE_LLI_WR_DEC_ERR): NO LLI Write Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
9	LLI_RD_DEC_ERR_IntStat	R	LLI Read Decode Error. Decode Error detected by Master Interface during LLI read operation. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/slave. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN1 bit which received the error is set to 0. ■ 0: NO LLI Read Decode Errors. ■ 1: LLI Read Decode Error detected Error Interrupt is generated if the corresponding bit in CHx_INTSTATUS_ENABLEReg is enabled. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVELLI_RD_DEC_ERR_): LLI Read Decode Error detected ■ 0x0 (INACTIVE_LLI_RD_DEC_ERR): NO LLI Read Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
8	DST_SLV_ERR_IntStat	R	Destination Slave Error. Slave Error detected by Master Interface during destination data transfer. This error occurs if the slave interface to which the data is written issues a Slave Error. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN bit corresponding to the channel which received the error is set to 0. ■ 0: No Destination Slave Errors ■ 1: Destination Slave Errors Detected This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_DST_SLV_ERR): Destination Slave Errors Detected ■ 0x0 (INACTIVE_DST_SLV_ERR): No Destination Slave Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7	SRC_SLV_ERR_IntStat	R	Source Slave Error. Slave Error detected by Master Interface during source data transfer. This error occurs if the slave interface from which the data is read issues a Slave Error. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN bit corresponding to the channel which received the error is set to 0. ■ 0: No Source Slave Errors ■ 1: Source Slave Error Detected This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SRC_SLV_ERR): Source Slave Error Detected ■ 0x0 (INACTIVE_SRC_SLV_ERR): No Source Slave Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
6	DST_DEC_ERR_IntStat	R	Destination Decode Error. Decode Error detected by Master Interface during destination data transfer. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/slave. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN bit corresponding to the channel which received the error is set to 0. ■ 0: No destination Decode Errors. ■ 1: Destination Decode Error Detected This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_DST_DEC_ERR): Destination Decode Error Detected ■ 0x0 (INACTIVE_DST_DEC_ERR): No destination Decode Errors. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
5	SRC_DEC_ERR_IntStat	R	Source Decode Error. Decode Error detected by Master Interface during source data transfer. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/slave. This error condition causes the DW_axi_dmac to disable the corresponding channel gracefully; the DMAC_ChEnReg.CH_EN bit corresponding to the channel which received the error is set to 0. ■ 0: No Source Decode Errors. ■ 1: Source Decode Error detected. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (ACTIVE_SRC_DEC_ERR): Source Decode Error detected ■ 0x0 (INACTIVE_SRC_DEC_ERR): No Source Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
4	DST_TRANSCOMP_IntStat	R	Destination Transaction Completed. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register or on enabling the channel (needed when interrupt is not enabled. **Values:** ■ 0x1 (ACTIVE_DST_TRANSCOMP): Destination transaction is complete ■ 0x0 (INACTIVE_DST_TRANSCOMP): Destination transaction is not complete **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	SRC_TRANSCOMP_IntStat	R	Source Transaction Completed. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register or on enabling the channel (needed when interrupt is not enabled. **Values:** ■ 0x1 (ACTIVE_SRC_TRANSCOMP): Source transaction is complete ■ 0x0 (INACTIVE_SRC_TRANSCOMP): Source transation is not complete **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	RSVD_DMAC_CHx_INTSTATUS REG_2	R	DMAC Channelx Specific Interrupt Register (bit 2) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Table 5-40 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
1	DMA_TFR_DONE_IntStat	R	DMA Transfer Done. This indicates to the software that the DW_axi_dmac has completed the requested DMA transfer. The DW_axi_dmac sets this bit to 1 along with setting CHx_INTSTATUS.BLOCK_TFR_DONE bit to 1 when the last block transfer is completed. ■ 0: DMA Transfer not completed. ■ 1: DMA Transfer Completed This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (DMA_TFR_COMPLETED): DMA Transfer completed ■ 0x0 (DMA_TFR_NOT_COMPLETE): DMA Transfer not complete **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
0	BLOCK_TFR_DONE_IntStat	R	Block Transfer Done. This indicates to the software that the DW_axi_dmac has completed the requested block transfer. The DW_axi_dmac sets this bit to 1 when the transfer is successfully completed. ■ 0: Block Transfer not completed. ■ 1: Block Transfer completed. This bit is cleared to 0 on writing 1 to the corresponding channel interrupt clear bit in CHx_IntClearReg register. **Values:** ■ 0x1 (BLOCK_TFR_COMPLETED): Block Transfer completed ■ 0x0 (BLOCK_TFR_NOT_COMPLETE): Block Transfer not complete **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

5.2.20 CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** This register contains fields that are used to enable the generation of port level interrupt at the channel level.
- **Size:** 64 bits
- **Offset:** 0x190 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:32	31	30	29	28	27	26:22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_32to63	Enable_CH_ABORTED_IntSignal	Enable_CH_DISABLED_IntSignal	Enable_CH_SUSPENDED_IntSignal	Enable_CH_SRC_SUSPENDED_IntSignal	Enable_CH_LOCK_CLEARED_IntSignal	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to26	Enable_SLVIF_WRONHOLD_ERR_IntSignal	Enable_SLVIF_SHADOWREG_WRON_VALID_ERR_IntSignal	Enable_SLVIF_WRONCHEN_ERR_IntSignal	Enable_SLVIF_RD2RWO_ERR_IntSignal	Enable_SLVIF_WR2RO_ERR_IntSignal	Enable_SLVIF_DEC_ERR_IntSignal	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_15	Enable_SLVIF_MULTIBLKTYPE_ERR_IntSignal	Enable_SHADOWREG_OR_LLI_INVALID_ERR_IntSignal	Enable_LLI_WR_SLV_ERR_IntSignal	Enable_LLI_RD_SLV_ERR_IntSignal	Enable_LLI_WR_DEC_ERR_IntSignal	Enable_LLI_RD_DEC_ERR_IntSignal	Enable_DST_SLV_ERR_IntSignal	Enable_SRC_SLV_ERR_IntSignal	Enable_DST_DEC_ERR_IntSignal	Enable_SRC_DEC_ERR_IntSignal	Enable_DST_TRANSCOMP_IntSignal	Enable_SRC_TRANSCOMP_IntSignal	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_2	Enable_DMA_TFR_DONE_IntSignal	Enable_BLOCK_TFR_DONE_IntSignal

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_32to63	R	DMAC Channelx Interrupt Status Enable Register (bits 32to63) Reserved bits - Read Only **Value After Reset:** 0xffffffff **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
31	Enable_CH_ABORTED_IntSignal	R/W	Channel Aborted Signal Enable. ■ 0: Disable the propagation of Channel Aborted Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Aborted Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_ABORTED_IntSignal): Enable the propagation of Channel Aborted Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_ABORTED_IntSignal): Disable the propagation of Channel Aborted Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
30	Enable_CH_DISABLED_IntSignal	R/W	Channel Disabled Signal Enable. ■ 0: Disable the propagation of Channel Disabled Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Disabled Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_DISABLED_IntSignal): Enable the propagation of Channel Disabled Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_DISABLED_IntSignal): Disable the propagation of Channel Disabled Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
29	Enable_CH_SUSPENDED_IntSignal	R/W	Channel Suspended Signal Enable. ■ 0: Disable the propagation of Channel Suspended Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Suspended Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_SUSPENDED_IntSignal): Enable the propagation of Channel Suspended Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_SUSPENDED_IntSignal): Disable the propagation of Channel Suspended Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
28	Enable_CH_SRC_SUSPENDED_IntSignal	R/W	Channel Source Suspended Signal Enable. ■ 0: Disable the propagation of Channel Source Suspended Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Source Suspended Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_SRC_SUSPENDED_IntSignal): Enable the propagation of Channel Source Suspended Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_SRC_SUSPENDED_IntSignal): Disable the propagation of Channel Source Suspended Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
27	Enable_CH_LOCK_CLEARED_IntSignal	R/W	Channel Lock Cleared Signal Enable. ■ 0: Disable the propagation of Channel Lock Cleared Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Lock Cleared Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_LOCK_CLEARED_IntSignal): Enable the propagation of Channel Lock Cleared Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_LOCK_CLEARED_IntSignal): Disable the propagation of Channel Lock Cleared Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
26:22	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to26	R	DMAC Channelx Interrupt Status Enable Register (bits 22to26) Reserved bits - Read Only **Value After Reset:** 0x1f **Exists:** Always
21	Enable_SLVIF_WRONHOLD_ERR_IntSignal	R/W	Slave Interface Write On Hold Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Write On Hold Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Write On Hold Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WRONHOLD_ERR_IntSignal): Enable the propagation of Slave Interface Write On Hold Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WRONHOLD_ERR_IntSignal): Disable the propagation of Slave Interface Write On Hold Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
20	Enable_SLVIF_SHADOWREG_WRON_VALID_ERR_IntSignal	R/W	Shadow Register Write On Valid Error Signal Enable. ■ 0: Disable the propagation of Shadow Register Write On Valid Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Shadow register Write On Valid Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_IntSignal): Enable the propagation of Shadow register Write On Valid Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_IntSignal): Disable the propagation of Shadow Register Write On Valid Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
19	Enable_SLVIF_WRONCHEN_ERR_IntSignal	R/W	Slave Interface Write On Channel Enabled Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Write On Channel enabled Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Write On Channel enabled Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WRONCHEN_ERR_IntSignal): Enable the propagation of Slave Interface Write On Channel enabled Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WRONCHEN_ERR_IntSignal): Disable the propagation of Slave Interface Write On Channel enabled Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
18	Enable_SLVIF_RD2RWO_ERR_IntSignal	R/W	Slave Interface Read to write Only Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Read to Write only Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Read to Write Only Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_RD2RWO_ERR_IntSignal): Enable the propagation of Slave Interface Read to Write Only Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_RD2RWO_ERR_IntSignal): Disable the propagation of Slave Interface Read to Write only Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
17	Enable_SLVIF_WR2RO_ERR_IntSignal	R/W	Slave Interface Write to Read Only Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Write to Read only Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Write to Read Only Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WR2RO_ERR_IntSignal): Enable the propagation of Slave Interface Write to Read Only Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WR2RO_ERR_IntSignal): Disable the propagation of Slave Interface Write to Read only Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
16	Enable_SLVIF_DEC_ERR_IntSignal	R/W	Slave Interface Decode Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_DEC_ERR_IntSignal): Enable the propagation of Slave Interface Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_DEC_ERR_IntSignal): Disable the propagation of Slave Interface Decode Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
15	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_15	R	DMAC Channelx Interrupt Status Enable Register (bit 15) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always
14	Enable_SLVIF_MULTIBLKTYPE_ERR_IntSignal	R/W	Slave Interface Multi Block type Error Signal Enable. ■ 0: Disable the propagation of Slave Interface Multi Block type Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Slave Interface Multi Block type Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_MULTIBLKTYPE_ERR_IntSignal): Enable the propagation of Slave Interface Multi Block type Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_MULTIBLKTYPE_ERR_IntSignal): Disable the propagation of Slave Interface Multi Block type Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
13	Enable_SHADOWREG_OR_LLI_INVALID_ERR_IntSignal	R/W	Shadow register or LLI Invalid Error Signal Enable. ■ 0: Disable the propagation of Shadow Register or LLI Invalid Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Shadow Register or LLI Invalid Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SHADOWREG_OR_LLI_INVALID_ERR_IntSignal): Enable the propagation of Shadow Register or LLI Invalid Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SHADOWREG_OR_LLI_INVALID_ERR_IntSignal): Disable the propagation of Shadow Register or LLI Invalid Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
12	Enable_LLI_WR_SLV_ERR_IntSignal	R/W	LLI WRITE Slave Error Signal Enable. ■ 0: Disable the propagation of LLI WRITE Slave Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI WRITE Slave Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_WR_SLV_ERR_IntSignal): Enable the propagation of LLI WRITE Slave Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_WR_SLV_ERR_IntSignal): Disable the propagation of LLI WRITE Slave Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
11	Enable_LLI_RD_SLV_ERR_IntSignal	R/W	LLI Read Slave Error Signal Enable. ■ 0: Disable the propagation of LLI Read Slave Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI Read Slave Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_RD_SLV_ERR_IntSignal): Enable the propagation of LLI Read Slave Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_RD_SLV_ERR_IntSignal): Disable the propagation of LLI Read Slave Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
10	Enable_LLI_WR_DEC_ERR_IntSignal	R/W	LLI WRITE Decode Error Signal Enable. ■ 0: Disable the propagation of LLI WRITE Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI WRITE Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_WR_DEC_ERR_IntSignal): Enable the propagation of LLI WRITE Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_WR_DEC_ERR_IntSignal): Disable the propagation of LLI WRITE Decode Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
9	Enable_LLI_RD_DEC_ERR_IntSignal	R/W	LLI Read Decode Error Signal Enable. ■ 0: Disable the propagation of LLI Read Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI Read Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_RD_DEC_ERR_IntSignal): Enable the propagation of LLI Read Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_RD_DEC_ERR_IntSignal): Disable the propagation of LLI Read Decode Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
8	Enable_DST_SLV_ERR_IntSignal	R/W	Destination Slave Error Signal Enable. ■ 0: Disable the propagation of Destination Slave Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Destination Slave Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_DST_SLV_ERR_IntSignal): Enable the propagation of Destination Slave Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_DST_SLV_ERR_IntSignal): Disable the propagation of Destination Slave Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
7	Enable_SRC_SLV_ERR_IntSignal	R/W	Source Slave Error Signal Enable. ■ 0: Disable the propagation of Source Slave Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Source Slave Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SRC_SLV_ERR_IntSignal): Enable the propagation of Source Slave Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SRC_SLV_ERR_IntSignal): Disable the propagation of Source Slave Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
6	Enable_DST_DEC_ERR_IntSignal	R/W	Destination Decode Error Signal Enable. ■ 0: Disable the propagation of Destination Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Destination Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_DST_DEC_ERR_IntSignal): Enable the propagation of Destination Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_DST_DEC_ERR_IntSignal): Disable the propagation of Destination Decode Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
5	Enable_SRC_DEC_ERR_IntSignal	R/W	Source Decode Error Signal Enable. ■ 0: Disable the propagation of Source Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Source Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SRC_DEC_ERR_IntSignal): Enable the propagation of Source Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SRC_DEC_ERR_IntSignal): Disable the propagation of Source Decode Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
4	Enable_DST_TRANSCOMP_IntSignal	R/W	Destination Transaction Completed Signal Enable. ■ 0: Disable the propagation of Destination Transaction complete Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Destination Transaction complete Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_DST_TRANSCOMP_IntSignal): Enable the propagation of Destination Transaction complete Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_DST_TRANSCOMP_IntSignal): Disable the propagation of Destination Transaction complete Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
3	Enable_SRC_TRANSCOMP_IntSignal	R/W	Source Transaction Completed Signal Enable. ■ 0: Disable the propagation of Source Transaction Complete Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Source Transaction Complete Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SRC_TRANSCOMP_IntSignal): Enable the propagation of Source Transaction Complete Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SRC_TRANSCOMP_IntSignal): Disable the propagation of Source Transaction Complete Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
2	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_2	R	DMAC Channelx Interrupt Status Enable Register (bit 2) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always
1	Enable_DMA_TFR_DONE_IntSignal	R/W	DMA Transfer Done Interrupt Signal Enable. ■ 0: Disable the propagation of DMA Transfer Done Interrupt to generate a port level interrupt ■ 1: Enable the propagation of DMA Transfer Done Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_DMA_TFR_DONE_IntSignal): Enable the propagation of DMA Transfer Done Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_DMA_TFR_DONE_IntSignal): Disable the propagation of DMA Transfer Done Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Table 5-41 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
0	Enable_BLOCK_TFR_DONE_Int Signal	R/W	Block Transfer Done Interrupt Signal Enable. ■ 0: Disable the propagation of Block Transfer Done Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Block Transfer Done Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_BLOCK_TFR_DONE_IntSignal): Enable the propagation of Block Transfer Done Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_BLOCK_TFR_DONE_IntSignal): Disable the propagation of Block Transfer Done Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

5.2.21 CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Description:** Writing 1 to specific field will clear the corresponding field in Channelx Interrupt Status Register(CHx_IntStatusReg).
- **Size:** 64 bits
- **Offset:** 0x198 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:32	31	30	29	28	27	26:22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD_DMAC_CHx_INTCLEARREG_32to63	Clear_CH_ABORTED_IntStat	Clear_CH_DISABLED_IntStat	Clear_CH_SUSPENDED_IntStat	Clear_CH_SRC_SUSPENDED_IntStat	Clear_CH_LOCK_CLEARED_IntStat	RSVD_DMAC_CHx_INTCLEARREG_22to26	Clear_SLVIF_WRONHOLD_ERR_IntStat	Clear_SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	Clear_SLVIF_WRONCHEN_ERR_IntStat	Clear_SLVIF_RD2RWO_ERR_IntStat	Clear_SLVIF_WR2RO_ERR_IntStat	Clear_SLVIF_DEC_ERR_IntStat	RSVD_DMAC_CHx_INTCLEARREG_15	Clear_SLVIF_MULTIBLKTYPE_ERR_IntStat	Clear_SHADOWREG_OR_LLI_INVALID_ERR_IntStat	Clear_LLI_WR_SLV_ERR_IntStat	Clear_LLI_RD_SLV_ERR_IntStat	Clear_LLI_WR_DEC_ERR_IntStat	Clear_LLI_RD_DEC_ERR_IntStat	Clear_DST_SLV_ERR_IntStat	Clear_SRC_SLV_ERR_IntStat	Clear_DST_DEC_ERR_IntStat	Clear_SRC_DEC_ERR_IntStat	Clear_DST_TRANSCOMP_IntStat	Clear_SRC_TRANSCOMP_IntStat	RSVD_DMAC_CHx_INTCLEARREG_2	Clear_DMA_TFR_DONE_IntStat	Clear_BLOCK_TFR_DONE_IntStat

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_CHx_INTCLEARREG_32to63	W	DMAC Channelx Interrupt Clear Register (bits 32to63) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
31	Clear_CH_ABORTED_IntStat	W	Channel Aborted Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_CH_ABORTED): Clear the CH_ABORTED interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
30	Clear_CH_DISABLED_IntStat	W	Channel Disabled Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_CH_DISABLED): Clear the CH_DISABLED interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
29	Clear_CH_SUSPENDED_IntStat	W	Channel Suspended Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_CH_SUSPENDED): Clear the CH_SUSPENDED interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
28	Clear_CH_SRC_SUSPENDED_IntStat	W	Channel Source Suspended Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_CH_SRC_SUSPENDED): Clear the CH_SRC_SUSPENDED interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
27	Clear_CH_LOCK_CLEARED_IntStat	W	Channel Lock Cleared Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_CH_LOCK_CLEARED): Clear the CH_LOCK_CLEARED interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
26:22	RSVD_DMAC_CHx_INTCLEARREG_22to26	W	DMAC Channelx Interrupt Clear Register (bits 22to26) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always
21	Clear_SLVIF_WRONHOLD_ERR_IntStat	W	Slave Interface Write On Hold Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_WRONHOLD_ERR): Clear the SLVIF_WRONHOLD_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
20	Clear_SLVIF_SHADOWREG_WRON_VALID_ERR_IntStat	W	Shadow Register Write On Valid Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_SHADOWREG_WRON_VALID_ERR): Clear the SLVIF_SHADOWREG_WRON_VALID_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
19	Clear_SLVIF_WRONCHEN_ERR_IntStat	W	Slave Interface Write On Channel Enabled Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_WRONCHEN_ERR): Clear the SLVIF_WRONCHEN_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
18	Clear_SLVIF_RD2RWO_ERR_IntStat	W	Slave Interface Read to write Only Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_RD2RWO_ERR): Clear the SLVIF_RD2RWO_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
17	Clear_SLVIF_WR2RO_ERR_IntStat	W	Slave Interface Write to Read Only Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_WR2RO_ERR): Clear the SLVIF_WR2RO_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
16	Clear_SLVIF_DEC_ERR_IntStat	W	Slave Interface Decode Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_DEC_ERR): Clear the SLVIF_DEC_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
15	RSVD_DMAC_CHx_INTCLEARREG_15	W	DMAC Channelx Interrupt Clear Register (bit 15) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always
14	Clear_SLVIF_MULTIBLKTYPE_ERR_IntStat	W	Slave Interface Multi Block type Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_MULTIBLKTYPE_ERR): Clear the SLVIF_MULTIBLKTYPE_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
13	Clear_SHADOWREG_OR_LLI_INVALID_ERR_IntStat	W	Shadow register or LLI Invalid Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SHADOWREG_OR_LLI_INVALID_ERR): Clear the SHADOWREG_OR_LLI_INVALID_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
12	Clear_LLI_WR_SLV_ERR_IntStat	W	LLI WRITE Slave Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_LLI_WR_SLV_ERR): Clear the LLI_WR_SLV_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
11	Clear_LLI_RD_SLV_ERR_IntStat	W	LLI Read Slave Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_LLI_RD_SLV_ERR): Clear the LLI_RD_SLV_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
10	Clear_LLI_WR_DEC_ERR_IntStat	W	LLI WRITE Decode Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_LLI_WR_DEC_ERR): Clear the LLI_WR_DEC_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
9	Clear_LLI_RD_DEC_ERR_IntStat	W	LLI Read Decode Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_LLI_RD_DEC_ERR): Clear the LLI_RD_DEC_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
8	Clear_DST_SLV_ERR_IntStat	W	Destination Slave Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_DST_SLV_ERR): Clear the DST_SLV_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
7	Clear_SRC_SLV_ERR_IntStat	W	Source Slave Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SRC_SLV_ERR): Clear the SRC_SLV_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
6	Clear_DST_DEC_ERR_IntStat	W	Destination Decode Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_DST_DEC_ERR): Clear the DST_DEC_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
5	Clear_SRC_DEC_ERR_IntStat	W	Source Decode Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SRC_DEC_ERR): Clear the SRC_DEC_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
4	Clear_DST_TRANSCOMP_IntStat	W	Destination Transaction Completed Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_DST_TRANSCOMP): Clear the DST_TRANSCOMP interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
3	Clear_SRC_TRANSCOMP_IntStat	W	Source Transaction Completed Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SRC_TRANSCOMP): Clear the SRC_TRANSCOMP interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
2	RSVD_DMAC_CHx_INTCLEARREG_2	W	DMAC Channelx Interrupt Clear Register (bit 2) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always
1	Clear_DMA_TFR_DONE_IntStat	W	DMA Transfer Done Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_DMA_TFR_DONE): Clear the DMA_TFR_DONE interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Table 5-42 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) (Continued)

Bits	Name	Memory Access	Description
0	Clear_BLOCK_TFR_DONE_IntStat	W	Block Transfer Done Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CH1_INTSTATUSREG **Values:** ■ 0x1 (CLEAR_BLOCK_TFR_DONE): Clear the interrupt in the Interrupt Status Register(CHx_IntStatusReg). Writing a 1 to this register field clears the corresponding bit in the CHx_IntStatusReg register. ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

6 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 6-1 Internal Parameters

Parameter Name	Equals To
DMAX_AXI_ID_SUFFIX_WIDTH	= (DMAX_M_ID_WIDTH - (DMAX_HAS_LLI_PARAM == 1 ? (LOG2_DMAX_NUM_CHANNELS + 1) : LOG2_DMAX_NUM_CHANNELS))
DMAX_AXI_LOCK_WIDTH	=(DMAX_MSTIF_MODE ==0 ? 2: 1)
DMAX_COMP_VER	64'h0000_0000_3130_322a
DMAX_HAS_LLI_PARAM	{[function_of : DMAX_NUM_CHANNELS, DMAX_CH(x)_MULTI_BLK_EN, DMAX_CH(x)_MULTI_BLK_TYPE]}
DMAX_HS_CLKRST_WIDTH	=(DMAX_HS_SAME_ASYNC_CLK ==1 ? 1: DMAX_NUM_HS_IF)
DMAX_NUM_CHANNELS_MUL_3	DMAX_NUM_CHANNELS*3
LOG2_2_DMAX_NUM_CHANNELS	{1 + [function_of: DMAX_NUM_CHANNELS]}
LOG2_3_DMAX_NUM_CHANNELS	{[function_of: DMAX_NUM_CHANNELS_MUL_3]}
LOG2_DMAX_ARB_RD_REQ_WIDTH	= ((DMAX_HAS_LLI_PARAM ==1) ? LOG2_3_DMAX_NUM_CHANNELS : LOG2_2_DMAX_NUM_CHANNELS)

Table 6-1 Internal Parameters (Continued)

Parameter Name	Equals To
LOG2_DMAX_ARB_WR_REQ_WIDTH	= ((DMAX_HAS_LLI_PARAM ==1) ? LOG2_3_DMAX_NUM_CHANNELS : LOG2_2_DMAX_NUM_CHANNELS)
LOG2_DMAX_NUM_CHANNELS	{[function_of: DMAX_NUM_CHANNELS]}
LOG2_DMAX_NUM_HS_IF	{[function_of: DMAX_NUM_HS_IF]}
DMAX_CH(x)_STW_ENC	{[function_of : DMAX_CH(x)_STW"}
DMAX_CH(x)_DTW_ENC	{[function_of : DMAX_CH(x)_DTW"}

7 Programming the DW_axi_dmac

The DW_axi_dmac can be programmed through software registers or the DW_axi_dmac low-level software driver; software registers are described in more detail in “Register Descriptions” on page 159.

7.1 Programming Flow for Shadow-Register-Based Multi-Block Transfer

1. Software reads the DMAC channel enable register (DMAC_ChEnReg) to select an available (unused) channel.
2. Software programs the CHx_CFG register with appropriate values for the DMA transfer.

   The SRC_MLTBLK_TYPE and / or DST_MLTBLK_TYPE bits must be set to 2’b10.

Note

- The CHx_CFG register must be programmed before programming the CHx_SAR, CHx_DAR, CHx_BLOCK_TS, or CHx_CTL registers, as the value of the SRC_MLTBLK_TYPE and/or DST_MLTBLK_TYPE fields are used for accessing the shadow registers.
- If the slave interface data bus width or transfer size is less than 64 bits, CHx_CFG[7:0] should be updated in the first write to the CHx_CFG register.

3. Software programs the CHx_SAR and / or CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers with appropriate values for the first block.

   DW_axi_dmac loads the corresponding shadow registers with these values.

   The CHx_CTL register must be the last register to be programmed with the ShadowReg_Or_LLI_Valid bit set to 1 to indicate that the shadow register contents are valid. If the slave interface data bus width or transfer size is less than 64 bits, CHx_CTL[63:56] must be updated last.
4. Software enables the channel by writing 1 to the appropriate bit location in the DMAC_ChEnReg register.

Note

It is possible to swap the sequence of step 3 and step 4. However, if step 4 is performed before step 3, DW_axi_dmac might generate a ShadowReg_Or_LLI_Invalid_ERR interrupt if the value of the ShadowReg_Or_LLI_Valid bit is 0 during the shadow register fetch phase.

5. DW_axi_dmac initiates the DMA block transfer operation based on the settings for the block transfer.
   a. The block transfer might start immediately or after the hardware or software handshaking request, depending on the value of the TT_FC field in the CHx_CFG register.
   b. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as '0', DW_axi_dmac waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability, before attempting another Shadow Register fetch operation. DW_axi_dmac might generate 'ShadowReg_Or_LLI_Invalid_ERR' Interrupt in this case.
   c. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as '1,' DW_axi_dmac copies the shadow register contents to the registers used for executing the DMA block transfer (CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS and CHx_CTL registers) and clears the ShadowReg_Or_LLI_Valid bit in CHx_CTL and CHx_CTL_ShadowReg registers to 0.
      i. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register as 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of current block transfer.
      ii. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register as 0, it understands that there are one or more blocks to be transferred and checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit again at the end of current block transfer.
6. Software polls the ShadowReg_Or_LLI_Valid bit in the CHx_CTL register till it is 0.
   a. DW_axi_dmac clears this bit to 0 only after copying the shadow register contents to the registers used for executing the DMA block transfer (that is, the CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers).
   b. Software must program the shadow registers with a new set of values only after the ShadowReg_Or_LLI_Valid bit is set to 0.
   c. If software tries to programs the shadow registers when the ShadowReg_Or_LLI_Valid bit is set to 1, DW_axi_dmac ignores this write operation, sets the SLVIF_ShadowReg_WrOnValid_ERR bit of the CHx_IntStatusReg register to 1, and generates an interrupt (if the corresponding interrupt generation is not masked off).
7. Software programs the CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers with appropriate values for the next block.
   a. The CHx_CTL register must be the last register to be programmed with the ShadowReg_Or_LLI_Valid bit set to 1 to indicate that the shadow register contents are valid.
   b. If current block is the final block in the transfer, S/W must set CHx_CTL.ShadowReg_Or_LLI_Last bit to 1.
   c. The DMA block transfer corresponding to the previous shadow register contents may be in progress during this time.
   d. DW_axi_dmac loads the corresponding shadow registers with these new values.
8. DW_axi_dmac initiates the DMA block transfer operation based on the settings for the block transfer.
   a. Based on the settings of TT_FC field in CHx_CFG register, the block transfer might start immediately or after the hardware/software handshaking request.

b. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as 0, DW_axi_dmac waits until software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability, before attempting another Shadow Register fetch operation. DW_axi_dmac might generate ShadowReg_Or_LLI_Invalid_ERR Interrupt in this case.

c. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as 1, DW_axi_dmac copies the shadow register contents to the registers used for executing the DMA block transfer (CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS and CHx_CTL registers) and clears the ShadowReg_Or_LLI_Valid bit in CHx_CTL and CHx_CTL_ShadowReg registers to 0.

d. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register as 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of current block transfer.

e. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register as 0, it understands that there are one or more blocks to be transferred and checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit again at the end of current block transfer.

9. Software waits for the block transfer completion interrupt or polls the block transfer completion indication bit (BLOCK_TFR_DONE) of the CHx_IntStatusReg register until it is set to 1.

10. On block transfer completion:

a. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as 0, DW_axi_dmac waits until software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability, before attempting another Shadow Register fetch operation. DW_axi_dmac might generate a ShadowReg_Or_LLI_Invalid_ERR Interrupt in this case.

b. DW_axi_dmac checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit and if it is seen as 1, DW_axi_dmac copies the shadow register contents to the registers used for executing the DMA block transfer (CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS and CHx_CTL registers) and clears ShadowReg_Or_LLI_Valid bit in CHx_CTL and CHx_CTL_ShadowReg registers to 0.

- If CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register is 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation.
- If CHx_CTL.ShadowReg_Or_LLI_Last bit of the copied Shadow Register is 0, it understands that there are one or more blocks to be transferred and checks CHx_CTL_ShadowReg.ShadowReg_Or_LLI_Valid bit again at the end of current block transfer.

c. If there are one or more blocks to be transferred, software polls CHx_CTL.ShadowReg_Or_LLI_Valid bit until it is seen as 0 and go to step 7.

One read operation is enough as DW_axi_dmac should have already copied the shadow register contents and cleared this bit to 0.

Note

In case when ShadowReg_Or_LLI_Invalid_ERR is generated the recommended flow to resume transfer is:

- Software programs CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS and CHx_CTL registers with appropriate values for the next block.
- Clear the interrupt using interrupt register CHx_IntClearReg.
- Program block resume request CHx_BLK_TFR_ResumeReqReg.

7.2 Programming Flow for Linked-List-Based Multi-Bock Transfer

1. Software reads the DMAC channel enable register (DMAC_ChEnReg) to select an available (unused) channel.
2. Software programs the CHx_CFG register with appropriate values for the DMA transfer.

   The SRC_MLTBLK_TYPE and/or DST_MLTBLK_TYPE bits must be set to 2'b11.
3. Software programs the base address of the first linked list item and the master interface on which the linked list item is available in the CHx_LLP register.
4. Software creates one or more linked list items in system memory. Software can create the entire linked list item in advance or dynamically extend the linked list using the CHx_CTL.ShadowReg_Or_LLI_Valid and CHx_CTL.LLI_Last fields of the LLI.
5. Software enables the channel by writing 1 to the appropriate bit location in DMAC_ChEnReg register.

Note It is possible to swap the sequence of step 4 and step 5. However, if step 5 is performed before step 4, or if the linked list item for the next block transfer is not available in system memory at any time during the multiblock transfer, as indicated by CHx_CTL.ShadowReg_Or_LLI_Valid bit of the fetched LLI being set to 0, DW_axi_dmac might generate a ShadowReg_Or_LLI_Invalid_ERR interrupt.

6. DW_axi_dmac initiates the DMA block transfer operation based on the settings for the block transfer.

   The block transfer might start immediately or after the hardware or software handshaking request, depending on the settings of the TT_FC field in the CHx_CFG register.

   DW_axi_dmac copies the linked list contents to the registers used for executing the DMA block transfer (that is, the CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS, and CHx_CTL registers) and initiates the DMA block transfer.
7. During the linked list fetch phase:
   a. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the fetched LLI as 1, it understands that the current block is the final block in the transfer and completes the DMA transfer operation at the end of current block transfer.
   b. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Last bit of the fetched LLI as 0, it understands that there are one or more blocks to be transferred and goes to step 6.
   c. If DW_axi_dmac sees CHx_CTL.ShadowReg_Or_LLI_Valid bit of the fetched LLI as 0, DW_axi_dmac might generate ShadowReg_Or_LLI_Invalid_ERR Interrupt. DW_axi_dmac waits till software writes (any value) to CHx_BLK_TFR_ResumeReqReg to indicate valid LLI availability, before attempting another LLI read operation.

7.3 Programming Flow for Single Block Transfer

1. Software reads the DMAC Channel Enable Register (DMAC_ChEnReg) to choose a free (unused) channel.

2. Software programs CHx_CFG register with multi-block type value of both source and destination peripheral to be 2'b00.
3. Software programs CHx_SAR and/or CHx_DAR, CHx_BLOCK_TS and CHx_CTL registers with appropriate values for the block.
4. Software enables the channel by writing 1 to the appropriate bit location in DMAC_ChEnReg register.
5. Source and destination requests single or burst DMA transactions to transfer the block of data (assuming non-memory peripherals). The DW_axi_dmac acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. Software waits for the block transfer completion interrupt/polls the block transfer completion indication bit (BLOCK_TFR_DONE) in CHx_IntStatusReg register till the bit is 1.

8 Verification

This chapter provides an overview of the testbench available for DW_axi_dmac verification. Once the DW_axi_dmac has been configured and the verification environment set up, simulations can be run automatically. For information on running simulations for DW_axi_dmac in coreAssembler or coreConsultant, see "Building and Verifying a Component or Subsystem" on page 25.

Note The DW_axi_dmac verification testbench is built with DesignWare Verification IP (VIP). Ensure that you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide.*

8.1 Overview of SV-UVM Tests

The DW_axi_dmac verification testbench (SV-UVM) performs the following tests that have been written to exhaustively verify the functionality:

- A single block transfer
- An LLP block chaining transfer (static and dynamic LLP)
- A reloading transfer
- A shadow transfer

Within each of these transfers, all parameters are randomized. The testbench constantly verifies whether conditions including the following are met:

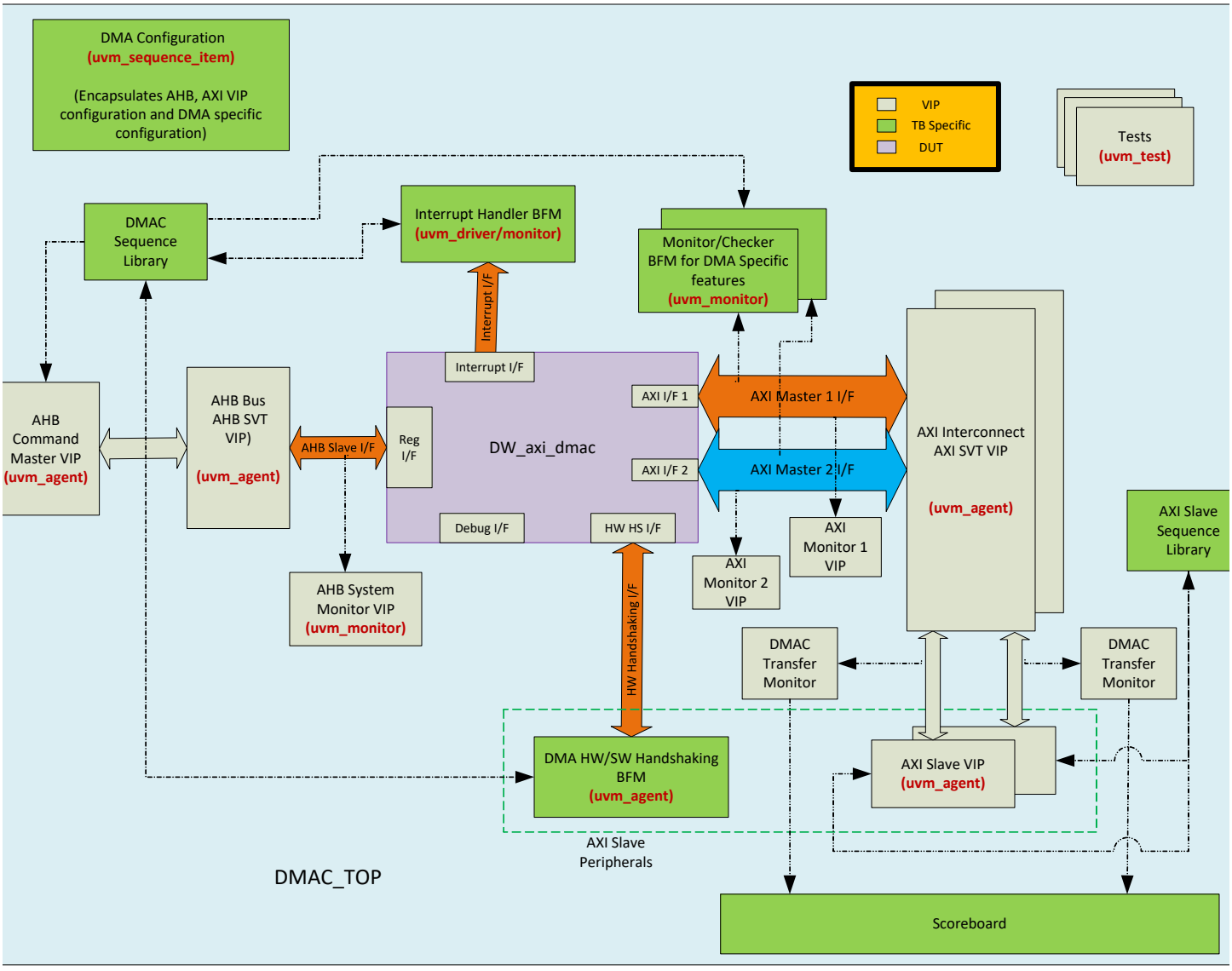
- Transfers are correct
- Registers are updated correctly
- Registers are written out correctly
- Interrupts are set correctly
- Flow control mode is not violated
- Bus and channel locking is correct
- Channel arbitration is correct

8.2 Overview of DW_axi_dmac Testbench

As illustrated in Figure 8-1, the DW_axi_dmac testbench is an SV-UVM testbench that includes:

- Verilog DUT (DW_axi_dmac)
- SV-UVM BFMs (Interrupt handler, hardware/software handshake, Transfer Monitors)
- VIPs (AHB Master, bus, monitor and AXI Slaves, interconnect, and monitors)

Figure 8-1 Verification Testbench Block Diagram



The file, dmac_top.sv, shows the instantiation of the top-level design in a testbench and resides in the "<workspace>/sim/testbench" directory. The testbench tests your configuration specified in the **Specify Configuration** task of coreConsultant and is self-checking. When a coreKit has been configured, the verification environment is stored in "<workspace>/sim". Files in "<workspace>/sim/testbench" form the

actual testbench for DW_axi_dmac. The “<workspace>/sim/<test_name>” directory contains the test_name.sv file.

9 Integration Considerations

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

9.1 Performance

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

9.1.1 Power Consumption, Frequency, and Area Results

The following section provides information about the synthesis results (power consumption, frequency, and area) of the DW_axi_dmac using the industry standard 28nm technology library and how it affects performance.

Table 9-1 provides synthesis results for the configurations where DMAX_CHx_MULTI_BLK_TYPE is set to 0.

Table 9-1 Synthesis Results for DW_axi_dmac Where DMAX_CHx_MULTI_BLK_TYPE = 0

Configuration	Operating Frequency (in MHz)	Gate Count	Leakage Power consumption	Dynamic Power consumption
Number of masters = 1 Number of channels = 1 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core (Default configuration)	Core clock- 400	20084 gates	318 nW	2.219 mW
Number of masters = 1 Number of channels = 8 Number of Handshakes = 16 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock-400 Slave clock-300	125037 gates	1.92 uW	12.3 mW

Configuration	Operating Frequency (in MHz)	Gate Count	Leakage Power consumption	Dynamic Power consumption
Number of masters = 1 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core	Core clock- 400 Slave clock-300	249952 gates	3.87 uW	23.7 mW
Number of masters = 2 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 64 Slave clock asynchronous to core Master clock asynchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock- 400 Slave clock - 300	532841 gates	8.53 uW	62.4 mW
Number of masters = 2 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 64 Low Power Enable = 1 (Global CSLP, Channel CSLP, SBIU CSLP, AXI CSLP) Slave clock asynchronous to core Master clock asynchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock- 400 Slave clock-300	538978 gates	8.75 uW	23.0 mW

The following table provides synthesis results for configurations where DMAX_CHx_MULTI_BLK_TYPE is set to 1.

Table 9-2 Synthesis Results for DW_axi_dmac Where DMAX_CHx_MULTI_BLK_TYPE = 1

Configuration	Operating Frequency (in MHz)	Gate Count	Leakage Power consumption	Dynamic Power consumption
Number of masters = 1 Number of channels = 1 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core (Default configuration)	Core clock- 400	23554 gates	370 nW	2.56 mW
Number of masters = 1 Number of channels = 8 Number of Handshakes = 16 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock-400 Slave clock-300	153083 gates	2.36 uW	15.1 mW

Configuration	Operating Frequency (in MHz)	Gate Count	Leakage Power consumption	Dynamic Power consumption
Number of masters = 1 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 8 Slave clock synchronous to core Master clock synchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock- 400 Slave clock-300	306349 gates	4.74 uW	29.3 mW
Number of masters = 2 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 64 Slave clock asynchronous to core Master clock asynchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock- 400 Slave clock- 300	593210 gates	9.55 uW	68 mW
Number of masters = 2 Number of channels = 16 Number of Handshakes = 32 FIFO depth of each channel = 64 Low Power Enable = 1 (Global CSLP, Channel CSLP, SBIU CSLP, AXI CSLP) Slave clock asynchronous to core Master clock asynchronous to core	Core clock- 400 Master 1 clock- 400 Master 2 clock- 400 Slave clock- 300	600259 gates	9.86 uW	32.1 mW

9.2 4K Boundary Crossing

The AXI protocol requires that any AXI burst does not cross a 4KB address boundary. DW_axi_dmac handles this situation automatically. If a DMA transfer is set up by software such that during the transfer, an AXI transfer crosses a 4KB boundary, DW_axi_dmac will automatically set up the AXI transfers such that the end of the 4KB boundary completes one AXI transfer and the beginning of the 4KB boundary starts another AXI transfer.

9.3 Read Accesses

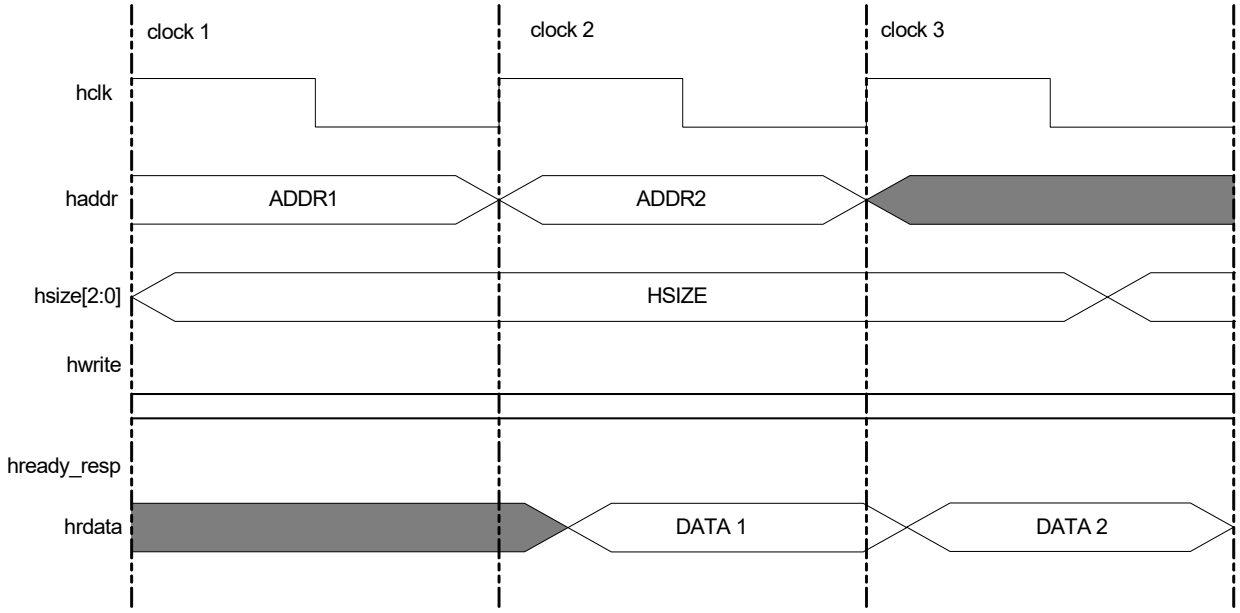
For reads, registers less than the full access width return zeros in the unused upper bits. All registers in DW_axi_dmac are READ and WRITE in DW_axi_dmac core clock domain. Therefore, the time taken for the reads or writes to complete depends on the slave interface clock mode (parameter DMAX_SLVIF_CLOCK_MODE).

9.3.1 Slave Interface Clock is Synchronous to the DW_axi_dmac Core Clock

When a slave interface clock is synchronous to the core clock, synchronization is not required. In this case, an AHB read takes two hclk cycles. The two cycles can be a control and data cycle, respectively. As shown in Figure 9-1, 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 9-1 AHB Read When Slave Clock is Synchronous to DW_axi_dmac Core Clock

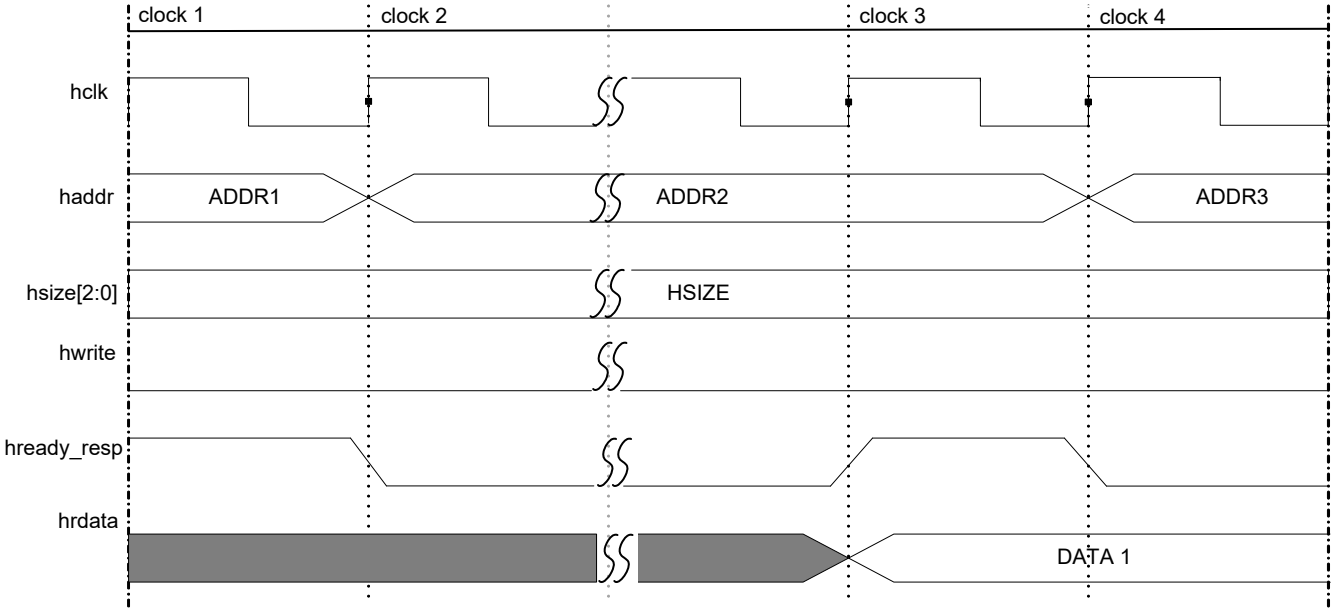


9.3.2 Slave Interface Clock is Asynchronous to DW_axi_dmac Core Clock

When a slave interface clock is asynchronous to DW_axi_dmac core clock, synchronization occurs in between each read operation. Once the read request is made on the slave interface, it is synchronized in DW_axi_dmac core clock domain and then response from register interface is synchronized back in the slave clock domain. Therefore, the data is driven on the slave interface only after the synchronization delay from slave interface clock to DW_axi_dmac core clock and the synchronization delay from DW_axi_dmac core clock to slave interface clock. Until then, hready_resp is driven low on AHB interface. As shown in Figure 9-2, the address and control for the first read is driven on clock 1; on clock 2 controls for next read is driven on the bus but hready_resp is pulled down by DW_axi_dmac. After synchronization delay on clock

3, data for first read is driven on the bus by DW_axi_dmac and in similar manner reads following that happens.

Figure 9-2 AHB Read When Slave Clock is Asynchronous to DW_axi_dmac Core Clock



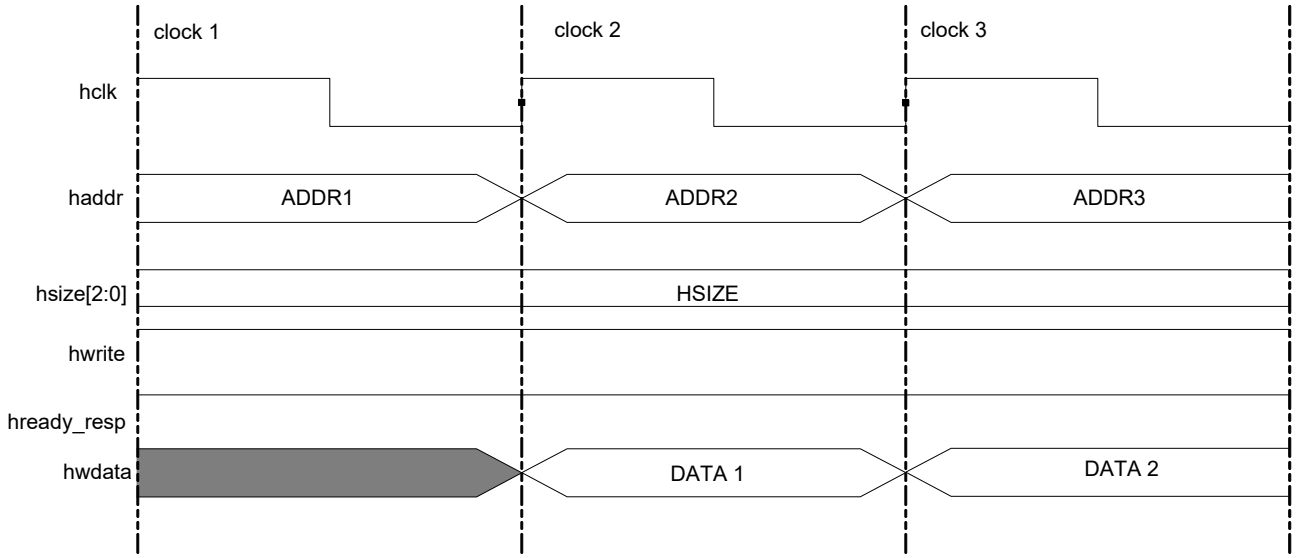
9.4 Write Accesses

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 read access, write access time is also dependent on the slave interface clock mode (parameter DMAX_SLVIF_CLOCK_MODE).

9.4.1 Slave Interface Clock is Synchronous to DW_axi_dmac Core Clock

Similar to read, a write access may be thought of as comprising a control and data cycle. As illustrated Figure 9-3, the address and control are 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.

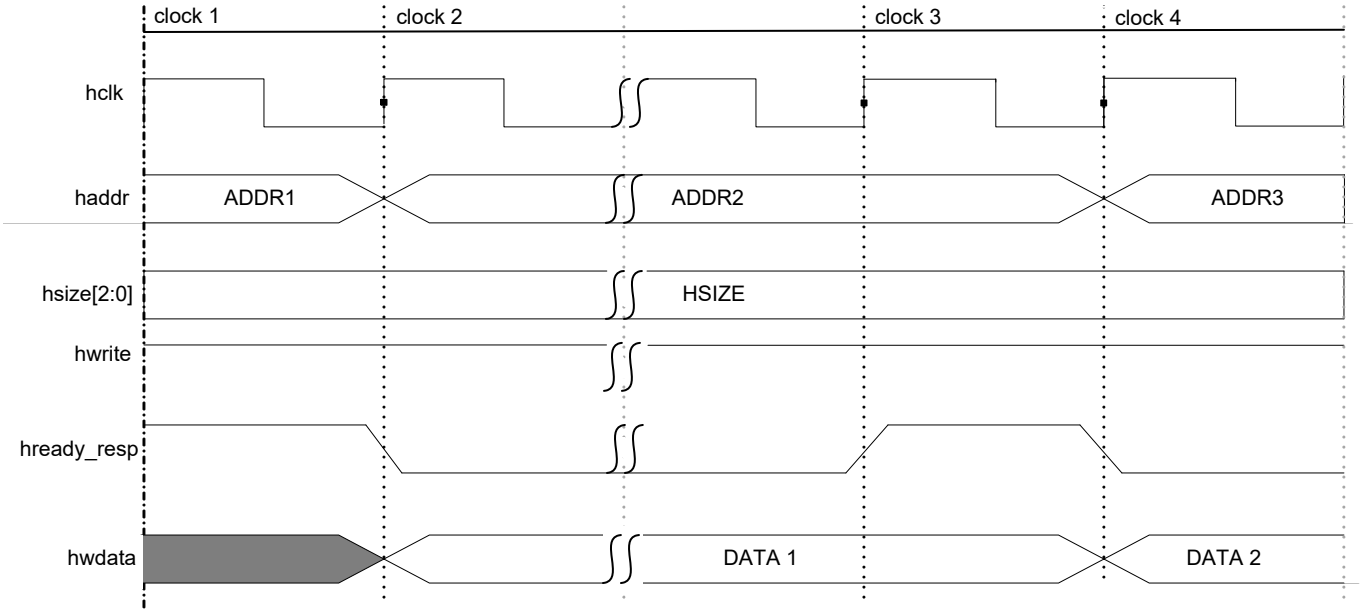
Figure 9-3 AHB Write When Slave Clock is Synchronous to DW_axi_dmac Core Clock



9.4.2 Slave Interface Clock is Asynchronous to DW_axi_dmac Core Clock

Similar to read, in this case also write requests are first synchronized in DW_axi_dmac core clock domain and then response is again synchronized to slave clock domain. Therefore, acceptance of next write and response only happens after synchronization delay from slave interface clock to DW_axi_dmac core clock and after synchronization delay from DW_axi_dmac core clock to slave interface clock. Until then, hready_resp signal is driven low by DW_axi_dmac. As shown in Figure 9-4, control is driven by AHB master on clock 1 and data along with control information for second write is driven on clock 2; at clock 2 DW_axi_dmac pulls down hready_resp to 0 indicating that current data and control has not yet been accepted. After synchronization delay on clock 3, hready_resp is asserted indicating that current data has been written. At clock 4, AHB master samples hread_resp and drives data from previous address and next control information.

Figure 9-4 AHB Write When Slave Clock is Asynchronous to DW_axi_dmac Core Clock

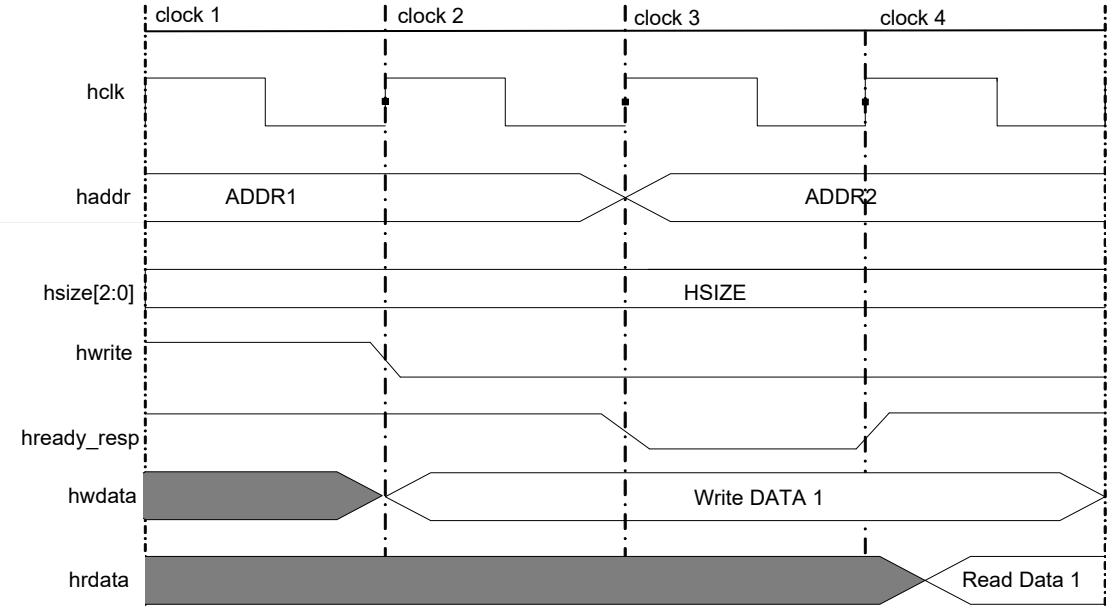


9.5 Consecutive Write-Read

This is a specific case for the AHB slave interface (only applicable when slave interface clock is synchronous to DW_axi_dmac core clock). The AMBA specification mentions that for a read after a write to the same address, the newly written data must be read back, and not the old data. To comply with this, the slave interface in the DW_axi_dmac inserts a "wait state" when it detects a read immediately after a write to the

same address. As shown in Figure 9-5, 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 9-5 AHB Wait State Read/Write



9.6 Accessing Top-Level Constraints

To get SDC constraints from coreConsultant (cC), 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>
   ```

A
Synchronizer Methods

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

This appendix contains the following sections:

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

A.1 Synchronizers used in DW_axi_dmac

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

Table A-1 Synchronizer used in DW_axi_dmac

Synchronizer module file	Sub module file	Synchronizer Type and Number
DW_axi_dmac_bcm21.v		Synchronizer 1: Simple Multiple register synchronizer
DW_axi_dmac_bcm22.v	DW_axi_dmac_bcm21.v	Synchronizer 2: Dual Clock Pulse Synchronizer
DW_axi_dmac_bcm23.v	DW_axi_dmac_bcm21.v	Synchronizer 3: Pulse Synchronizer with Acknowledge
DW_axi_dmac_bcm37.v	DW_axi_dmac_bcm22.v DW_axi_dmac_bcm21.v	Synchronizer 3: Reset Sequence Synchronizer
DW_axi_dmac_bcm07.v	DW_axi_dmac_bcm05.v DW_axi_dmac_bcm21.v	Synchronizer 4: Synchronous dual clock FIFO controller with Static Flags

Caution

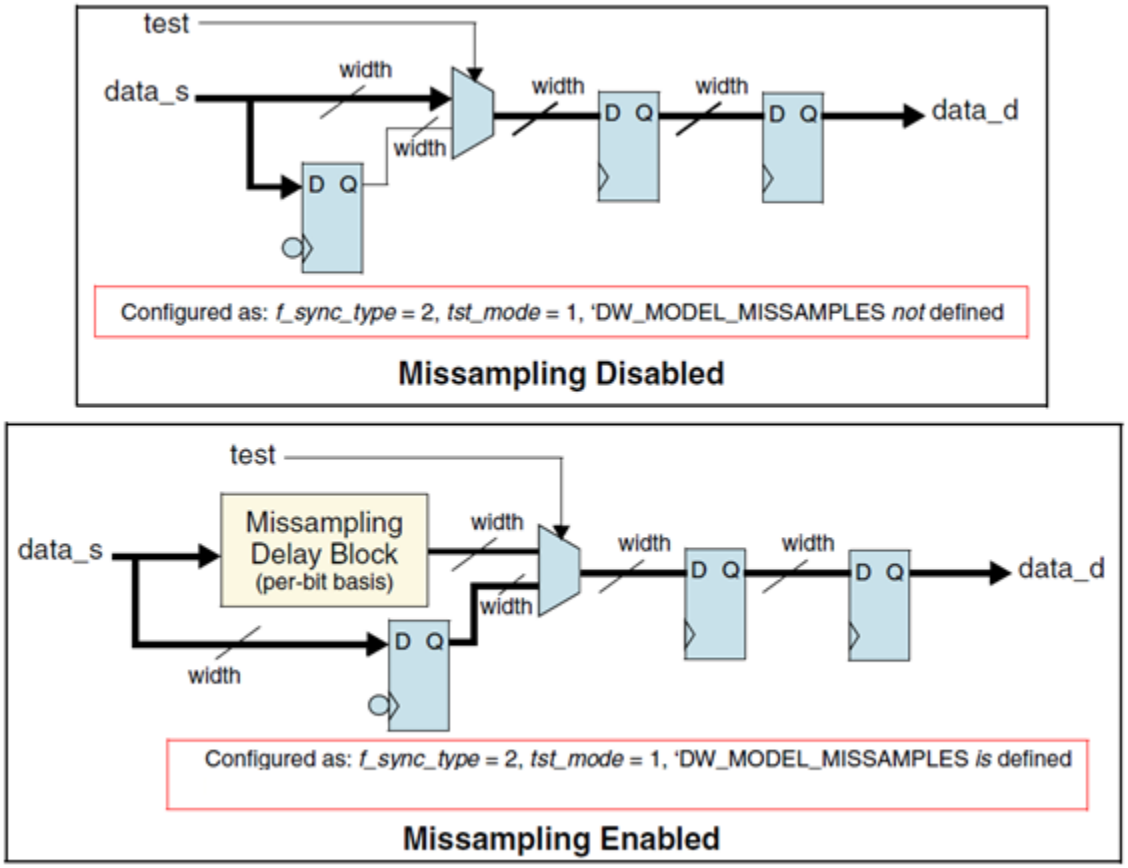
Be cautious while choosing the depth (DMAX_*_SYNC_DEPTH) for different synchronization mechanisms as 1 (where the first stage - negative edge flip-flop is used and for the second stage - positive edge flip-flop is used). At higher frequencies, as the maximum time available for meta-stability resolution is halved with respect to the available clock period, this can lead to meta-stability - when synchronizer depth is selected as 1.

It is recommended to use the synchronizer depths that are greater than or equal to 2. The depth of 1 will be deprecated in the future releases.

A.2 Synchronizer 1: Simple Double Register Synchronizer

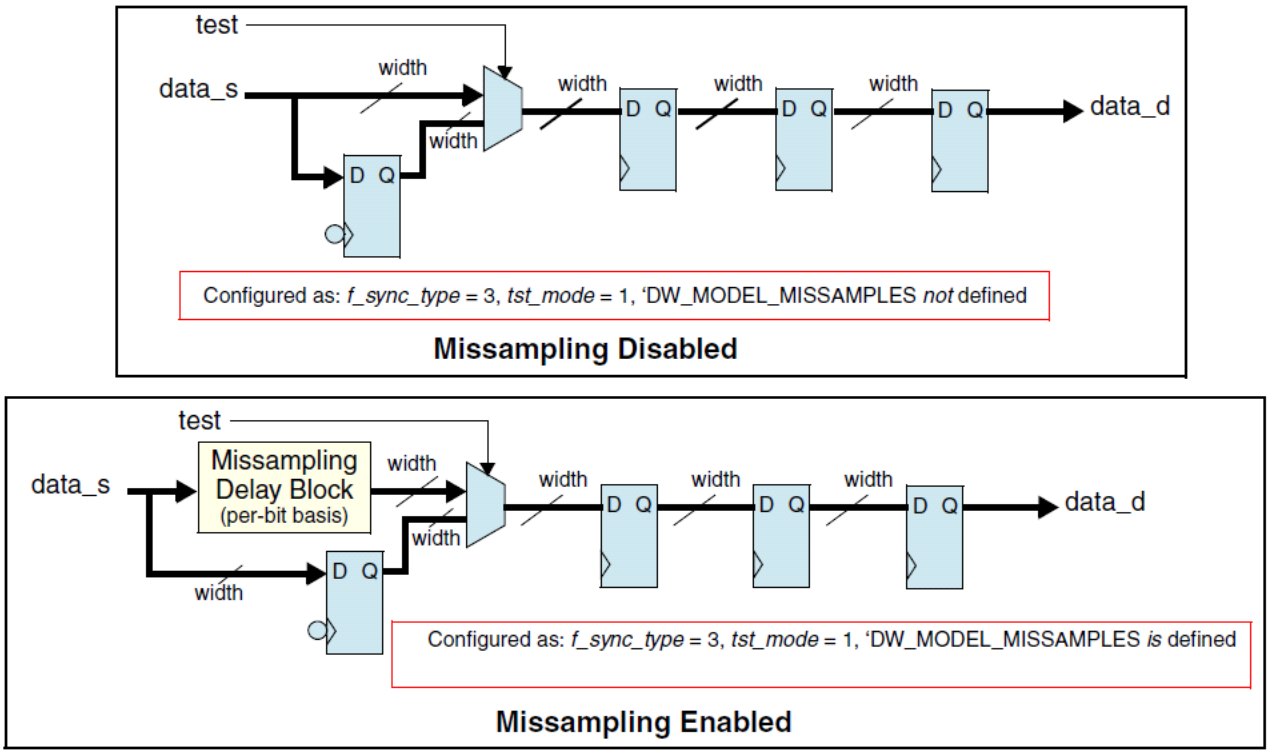
This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If aclk_m1 or aclk_m2 and dmac_core_clock are asynchronous (DMAX_MSTIF1(/2)_CLOCK_MODE = 1) then DW_axi_dmac_bcm21 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameters DMAX_M1(/2)_2_C_SYNC_DEPTH and DMAX_C_2_M1(/2)_SYNC_DEPTH. The following example shows the two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block Diagram of Synchronizer 1 with Two-Stage Synchronization (Both Positive Edges)



The following example shows the three stage synchronization process (Figure A-2) both using positive edge of clock.

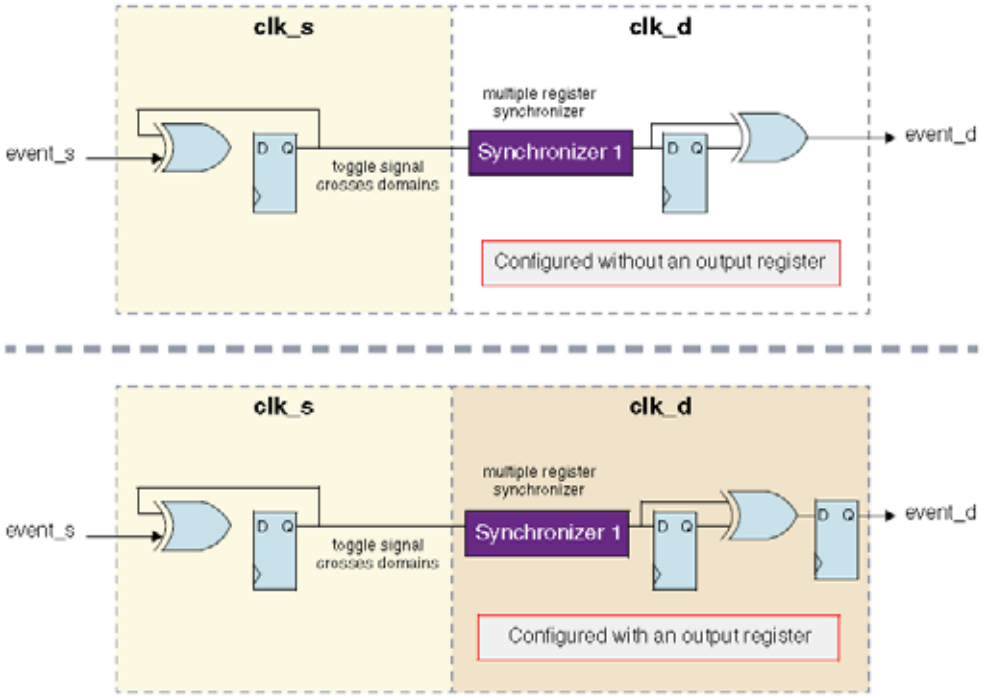
Figure A-2 Block Diagram of Synchronizer 1 with Three Stage Synchronization (Both Positive Edge)



A.3 Synchronizer 2: Dual Clock Pulse Synchronizer

This is a dual clock pulse synchronizer which provides a low-risk method for transmitting single clock cycle pulses between two different clock domains. This synchronizer is used to synchronize the pulse between slave clock (hclk) domain to dmac_core_clock domain. The synchronizer will be instantiated when hclk is asynchronous with dmac_core_clock (DMAX_SLVIF_CLOCK_MODE = 1). The number of stages of synchronization is configurable through the parameters DMAX_S_2_C_SYNC_DEPTH and DMAX_C_2_S_SYNC_DEPTH. Figure A-3 shows the block diagram of Dual clock pulse synchronizer which uses DW_axi_bcm21 sub-module in design.

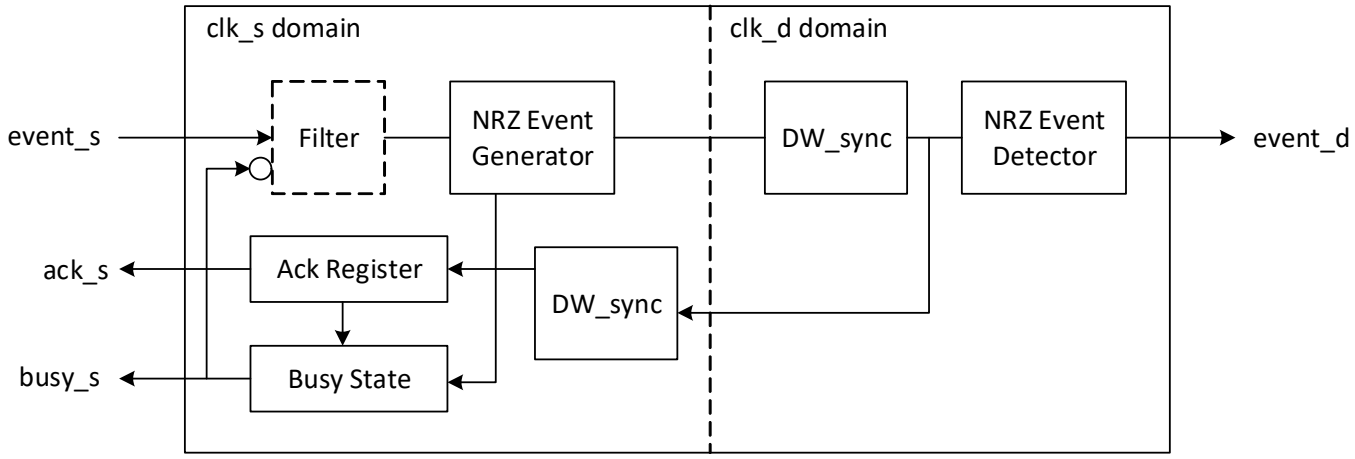
Figure A-3 Dual Clock Pulse Synchronizer Block Diagram



A.4 Synchronizer 3: Pulse Synchronizer with Acknowledge

Dual-clock pulse synchronizer with acknowledge, provides a low-risk method for transmitting single-clock cycle pulses between two different clock domains using acknowledge. The synchronization scheme depends on the core configuration. If hs_clk[y-1] and dmac_core_clock are asynchronous (DMAX_HS(y)_ASYNC_CLK = 1), then DW_axi_dmac_bcm23 is instantiated inside the core for synchronization of the DMA Handshake signals. The number of stages of synchronization is configurable through the parameters DMAX_HS_2_C_SYNC_DEPTH and DMAX_C_2_HS_SYNC_DEPTH. Figure A-4 shows the block diagram of Pulse Synchronizer with Acknowledge, which uses DW_axi_dmac_bcm21 (DW_sync) sub-module in design.

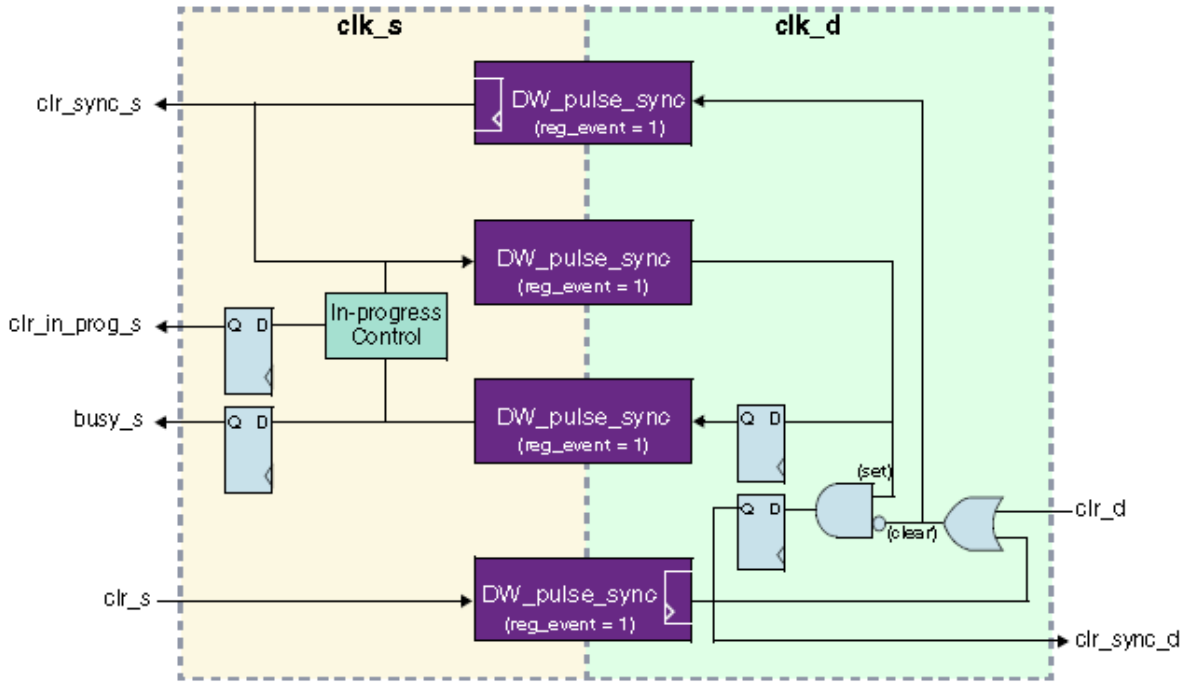
Figure A-4 Block Diagram of Pulse Synchronizer with Acknowledge



A.5 Synchronizer 4: Reset Sequence Synchronizer

This module provides a coordinated reset sequence to the source and destination domain logic when a "clear" is initiated by either domain. The synchronization scheme depends on core configuration. If aclk_m1/ aclk_m2 and dmac_core_clock are asynchronous (DMAX_MSTIF1(/2)_CLOCK_MODE = 1) then DW_axi_dmac_bcm37 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameters DMAX_M1(/2)_2_C_SYNC_DEPTH and DMAX_C_2_M1(/2)_SYNC_DEPTH. Figure A-5 shows the block diagram of Reset sequence synchronizer which uses DW_axi_bcm21 and DW_axi_bcm22 sub-modules in design.

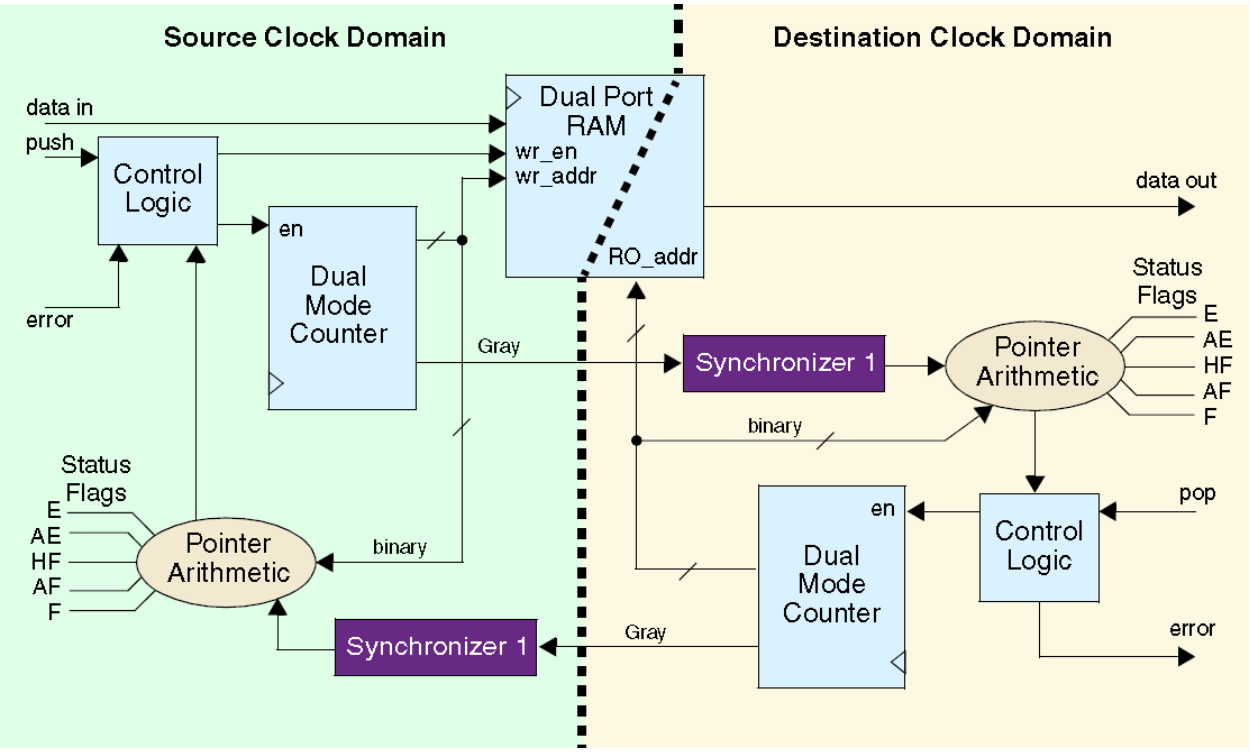
Figure A-5 Reset Sequence Synchronizer Block Diagram



A.6 Synchronizer 5: Dual Clock FIFO Controller with Static Flags

This module implements the functions required to implement a FIFO once connected to a RAM. The push and pop interfaces are in different clock domains. Gray coded pointers are used to pass information between domains. The synchronization scheme depends on core configuration. If aclk_m1/ aclk_m2 and dmac_core_clock are asynchronous (DMAX_MSTIF1(/2)_CLOCK_MODE = 1) then DW_axi_dmac_bcm07 is instantiated inside the core for synchronization for data from dmac_core_clock domain to aclk_m1/2 clock domain. The number of stages of synchronization is configurable through the parameters DMAX_M1(/2)_2_C_SYNC_DEPTH and DMAX_C_2_M1(/2)_SYNC_DEPTH. Figure A-6 shows the block diagram of dual clock FIFO controller which used DW_axi_bcm21 module to synchronize signals inside the module.

Figure A-6 Dual Clock FIFO Controller Block Diagram



B
Glossary

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

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

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

Index

N

P

R

S

T

V

W

Z