DesignWare® AXI DMA Controller (DMAC)

Databook

DW_axi_dmac – Product Codes

Version 2.01a
December 2022

Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

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.

www.synopsys.com

Contents

Revision History 9

Preface 13
Product Codes 13
Databook Organization 13
Related Documentation 14
Web Resources 14
Reference Documentation 14
Synopsys Statement on Inclusivity and Diversity 14
Replacement of Noninclusive Terminology 15
Customer Support 15

Chapter 1
Product Overview 17
1.1 DesignWare System Overview 18
1.2 General Product Description 20
1.2.1 DW_axi_dmac Block Diagram 20
1.3 Features 21
1.3.1 General Features 21
1.3.2 Channel Buffering 22
1.3.3 Channel Control 22
1.3.4 Flow Control 22
1.3.5 Handshaking Interface 22
1.3.6 Interrupt Outputs 23
1.3.7 Bus Interface 23
1.3.8 Mission Critical Features 23
1.3.9 Unsupported Features 23
1.4 Terminology 24
1.5 Standards Compliance 27
1.6 Verification Environment Overview 27
1.7 Licenses 27
1.8 Deliverables 27
1.9 Where To Go From Here 28

Chapter 2
Functional Description 29

2.1 Data Flow 30
2.2 Register Bus Interface 32
2.2.1 Overview of Register Bus Interface 32
2.2.2 AHB Register Bus Interface 32
2.2.3 APB4 Register Bus Interface 32
2.2.4 Configuring Register Bus Interface 33
2.2.5 Signal Related to Register Bus Interface 33
2.3 Manager Interface 35
2.4 Arbitration Scheme 36
2.4.1 Fair-Among-Equals Arbitration 36
2.4.2 Round Robin Arbitration 41
2.4.3 Locks and Grants 43
2.4.4 Priority for Different Accesses 43
2.4.5 Same Channel Transfer, Fetch, and Access 43
2.5 Channel Locking 44
2.5.1 Enabling Channel Locking 44
2.5.2 Clearing Channel Locking 44
2.5.3 Possible Deadlock Conditions 44
2.6 Endian Scheme. 47
2.7 Single Transaction Region 52
2.8 Handshaking Interface 54
2.8.1 Hardware Handshaking 54
2.8.2 Software Handshaking 64
2.9 Interrupt Interface 65
2.10 Flow Control Configurations. 67
2.11 Early Terminated Burst Transaction. 69
2.12 Transfer Control 70
2.12.1 Single Block Transfer 70
2.12.2 Multi-block Transfer 70
2.13 External Memory Interface 78
2.13.1 Overview of External Memory Interface 78
2.13.2 Description of External Memory Interface 78
2.13.3 Enabling External Memory Interface 79
2.13.4 Signals Related to External Memory Interface 79
2.14 Unique ID Feature. 81
2.14.1 Overview of Unique ID Feature 81
2.14.2 Description of Unique ID Feature 81
2.14.3 Enabling the Unique ID Feature 85
2.14.4 Signals Related to the Unique ID Feature 85
2.15 Context Sensitive Low Power Option 87
2.15.1 DMA Channel Context Sensitive Low Power Technique 87
2.15.2 Register Bus Interface Context Sensitive Low Power Technique 88
2.15.3 AXI Manager Interface Context Sensitive Low Power Technique 89
2.15.4 Global Context Sensitive Low Power Technique 90
2.16 AXI Unaligned Transfer Support 91
2.16.1 Description of AXI Unaligned Transfers 91
2.16.2 AXI Unaligned Transfer Examples 93
2.17 Channel Suspend, Disable, and Terminate 109
2.17.1 Channel Suspend 109

2.17.2 Channel Suspend and Resume 109
2.17.3 Channel Suspend and Disable Prior to Transfer Completion 110
2.17.4 Channel Disable Prior to Transfer Completion without Suspend 110
2.17.5 Abnormal Channel Terminate 111
2.18 DMA Channel Status Register – CHx_STATUSREG 112
2.18.1 Completed Block Transfer Size 112
2.18.2 Data Left in the FIFO – CHx_STATUSREG.DATA_LEFT_IN_FIFO 112
2.19 Debug Interface 114
2.19.1 Register Bus Interface and Controller Status Indication 114
2.19.2 DMAC Hold Control 114
2.19.3 Error Handling 115

Chapter 3
Mission Critical Features 117
3.1 Register Bus Interface Parity Feature 118
3.1.1 Overview of the Register Bus Interface Parity Feature 118
3.1.2 AHB/APB4 Interface Address Parity 118
3.1.3 AHB/APB4 Interface Data Parity 118
3.1.4 Interface Parity Error Interrupt 119
3.1.5 Interface Parity Error Interrupt Signal 120
3.1.6 Enabling Register Bus Interface Parity Feature 121
3.1.7 Signals Related to Register Bus Interface Parity Feature 121
3.1.8 Registers Related to the Register Bus Interface Parity Feature 121
3.2 Error Correction Code (ECC) Protection Feature 122
3.2.1 Overview of ECC Protection Feature 122
3.2.2 Description of ECC Protection Feature 122
3.2.3 Enabling the ECC Protection Feature 124
3.2.4 Registers Related to ECC Protection Feature 124
3.3 ECC Protection Feature for FIFO Memory Interface 125
3.3.1 Overview of ECC Protection Feature for FIFO Memory Interface 125
3.3.2 Description of ECC Protection Feature for FIFO Memory Interface 125
3.3.3 Error Injection Capabilities for FIFO Memory Interface 126
3.3.4 Diagnostic Support for Error Management for FIFO Memory Interface 126
3.3.5 Enabling the ECC Protection Feature for FIFO Memory Interface 128
3.4 ECC Protection Feature for UID Memory Interface 129
3.4.1 Overview of ECC Protection Feature for UID Memory Interface 129
3.4.2 Description of ECC Protection Feature for UID Memory Interface 129
3.4.3 Error Injection Capabilities for UID Memory Interface 130
3.4.4 Diagnostic Support for Error Management for UID Memory Interface 130
3.4.5 Enabling the ECC Protection Feature for UID Memory Interface 133
3.5 ECC and Parity Protection Feature for AXI Manager Interface 134
3.5.1 Overview of ECC and Parity Protection Feature for AXI Manager Interface 134
3.5.2 ECC and Parity Generator for AXI Interface 134
3.5.3 ECC and Parity Monitor for AXI Interface 135
3.5.4 AXI Channel Organization – ECC and Parity 137
3.5.5 ECC Mode 139
3.5.6 ECC Granularity 141
3.5.7 Parity Mode 143

3.5.8 Diagnostic Support for Error Management for AXI Manager Interface (with ECC and Parity Protection Feature) . . . 144
3.5.9 AXI Interface VALID and READY Parity Generator and Checker . . . 147
3.5.10 Enabling ECC and Parity Protection Feature for AXI Manager Interface . . . 149
3.6 Lock Step Protection Feature . . . 150
3.6.1 Overview of Lock Step Protection Feature . . . 150
3.6.2 Description of the Lock Step Protection Feature . . . 150
3.6.3 Diagnostic Support for Error Management . . . 154
3.6.4 Enabling the Lock Step Protection Feature . . . 155
3.6.5 Signals Related to the Lock Step Protection Feature . . . 156

Chapter 4
Parameter Descriptions . . . 157
4.1 Top Level Parameters . . . 158
4.2 Manager Interface Configuration Parameters . . . 163
4.3 Register Bus Interface Configuration Parameters . . . 166
4.4 Clocking Parameters . . . 167
4.5 Safety Feature Configuration Parameters . . . 171
4.6 Low Power Configuration Parameters . . . 176
4.7 Channel x Configuration Parameters . . . 178

Chapter 5
Signal Descriptions . . . 187
5.1 Core Clock and Reset Signals . . . 189
5.2 Test Interface Signals . . . 191
5.3 AHB Signals . . . 192
5.4 APB Signals . . . 196
5.5 Manager Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals . . . 200
5.6 Hardware Handshaking Signals . . . 214
5.7 External FIFO Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS) . . . 217
5.8 External UID Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS) . . . 219
5.9 Debug Interface Signals . . . 221
5.10 Interrupt Interface Signals . . . 230
5.11 Lock Step Signals . . . 238
5.12 Low Power Interface Signals . . . 241

Chapter 6
Register Descriptions . . . 245
6.1 Common_Registers_Address_Block Registers . . . 251
6.1.1 DMAC_IDREG . . . 252
6.1.2 DMAC_COMPVERREG . . . 253
6.1.3 DMAC_CFGREG . . . 254
6.1.4 DMAC_CHENREG . . . 256
6.1.5 DMAC_CHENREG2 . . . 284
6.1.6 DMAC_CHSUSPREG . . . 309
6.1.7 DMAC_CHABORTREG . . . 348
6.1.8 DMAC_INTSTATUSREG2 . . . 391
6.1.9 DMAC_INTSTATUSREG . . . 399
6.1.10 DMAC_COMMONREG_INTCLEARREG . . . 402
6.1.11 DMAC_COMMONREG_INTSTATUS_ENABLEREG . . . 413

6.1.12 DMAC_COMMONREG_INTSIGNAL_ENABLEREG 428
6.1.13 DMAC_COMMONREG_INTSTATUSREG 444
6.1.14 DMAC_RESETREG 461
6.1.15 DMAC_LOWPOWER_CFGREG 462
6.1.16 DMAC_COMMON_PARCTLREG 466
6.1.17 DMAC_COMMON_ECCCTLSTATUSREG 468
6.1.18 DMAC_SFTY_ERR_INJ_CTRL 472
6.2 Channelx_Register_Address_Block Registers 475
6.2.1 CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS) 476
6.2.2 CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS) 477
6.2.3 CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS) 478
6.2.4 CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS) 480
6.2.5 CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS) 493
6.2.6 CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS) 505
6.2.7 CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS) 517
6.2.8 CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS) 519
6.2.9 CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS) 521
6.2.10 CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS) 525
6.2.11 CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS) 529
6.2.12 CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS) 531
6.2.13 CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS) 533
6.2.14 CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) 535
6.2.15 CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS) 537
6.2.16 CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS) 539
6.2.17 CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS) 540
6.2.18 CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS) 541
6.2.19 CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS) 555
6.2.20 CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS) 573
6.2.21 CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS) 587
6.2.22 CHx_CFG_EXTD (for x = 1; x <= DMAX_NUM_CHANNELS) 597

Appendix A
Area, Speed, Power, and Performance 599
A.1 Area, Frequency, and Power Consumption 600

Appendix B
Internal Parameter Descriptions 617

Appendix C
Glossary 621

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.

Note

- A change-bar version of this databook is available on the following IP web page under the "Documentation" section:
  https://www.synopsys.com/dw/ipdir.php?c=DW_axi_dmac
- Links and references to chapters, sections, tables, figures, and page numbers in the Revision History table are assured to be valid only for the current version.

Version	Date	Description
2.01a	December 2022	**Added:** ■ "Product Codes" on page 13 ■ A note in "AHB Register Bus Interface" on page 32 ■ Figure 2-36 on page 79 ■ Figure 2-37 on page 79 ■ A note in "Description of Unique ID Feature" on page 81 ■ "DMA Channel Status Register – CHx_STATUSREG" on page 112 ■ A note in "Description of ECC Protection Feature" on page 122 ■ "Error Injection Capabilities for FIFO Memory Interface" on page 126 ■ "Error Injection Capabilities for UID Memory Interface" on page 130 **Updated:** ■ Release version and date ■ Figure 1-2 on page 20 (Updated "8" as "32") ■ "Safety Features" section as "Mission Critical Features" on page 23 ■ Figure 2-1 on page 30 (Updated "7" as "N") ■ Figure 2-2 on page 31 (Updated "7" as "N") ■ "Manager Interface" on page 35

Version	Date	Description
		[Continued] ■ "Arbitration Scheme" on page 36 (Added introduction to "Fair-Among-Equals Arbitration" on page 36 section and "Round Robin Arbitration" on page 41) ■ "Locks and Grants" on page 43 ■ Figure 2-9 on page 45 (Fixed a typo – "LL1" as "LLI") ■ Figure 2-11 on page 47, Figure 2-13 on page 47, Figure 2-14 on page 48, Figure 2-16 on page 48, Figure 2-17 on page 49, Figure 2-18 on page 49, and Figure 2-20 on page 50 (Added dotted dividers for consistency and fixed missing arrow heads) ■ "Handshaking Interface" on page 54 (Updated maximum number of handshaking interfaces supported) ■ Figure 2-23 on page 56 (Updated dma_last for clarity) ■ The note in "Channel Suspend" on page 109 ■ "Register Bus Interface Parity Feature" on page 118 ■ "ECC Protection Feature for AXI Master Interface" section as "ECC and Parity Protection Feature for AXI Manager Interface" on page 134 ■ Figure 3-4 on page 151 and Figure 3-5 on page 153 ■ "Diagnostic Support for Error Management" on page 154 ■ "Signals Related to the Lock Step Protection Feature" on page 156 (Added intfc_type_ls_err_inj_pri and intfc_type_ls_err_inj_shdw signals to the interrupt signals list) ■ "Area, Frequency, and Power Consumption" on page 600 ■ Chapter 4, "Parameter Descriptions" ■ Chapter 5, "Signal Descriptions" ■ Chapter 6, "Register Descriptions" ■ Appendix B, "Internal Parameter Descriptions" **Replaced:** ■ Noninclusive and insensitive terms with globally accepted inclusive terminology. See "Replacement of Noninclusive Terminology" on page 15. **Moved:** The following content to the *DW_axi_dmac User Guide*: ■ "Clocks and Resets" section ■ "Programming the DW_axi_dmac", "Verification", and "Integration Considerations" chapters (except "Performance" section) ■ "Basic Core Module (BCM) Library" Appendix **Removed:** ■ "AXI 3 locked transfers (bus locking)" from "Unsupported Features" on page 23

Version	Date	Description
2.00a	March 2020	■ Revision version change for 2020.03a release **Added:** ■ Chapter 3, “Mission Critical Features” ■ “External Memory Interface” on page 78 ■ “Unique ID Feature” on page 81 **Updated:** ■ Release version and date ■ “Register Bus Interface” on page 32 ■ Synthesis Results in “Performance” ■ Chapter 4, “Parameter Descriptions”, Chapter 5, “Signal Descriptions”, and Chapter 6, “Register Descriptions” auto-extracted from the RTL
1.02a	February 2018	**Added:** ■ “AXI Unaligned Transfer Support” ■ “Context Sensitive Low Power Option” ■ “Asynchronous Hardware Handshake Support” ■ “Single Arbiter Scheme” ■ “Multi-Arbiter Scheme” **Updated:** ■ Release version and date ■ Synthesis Results in “Performance” ■ “Signal Descriptions”, “Parameter Descriptions”, and “Register Descriptions” 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
1.01a	March 2016	■ Revision version change for 2016.03a release ■ Added “Running SpyGlass® Lint and SpyGlass® CDC” section ■ Added “Running SpyGlass on Generated Code with coreAssembler” section ■ “Signal Descriptions”, “Parameter Descriptions”, and “Register Descriptions” auto-extracted from the RTL ■ Added “Internal Parameter Descriptions” ■ Removed “Narrow transfers” from “Unsupported Features” ■ 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, “Basic Core Module (BCM) Library” ■ Updated area and power numbers in “Performance” ■ Added “Programming Flow for Single Block Transfer”

Version	Date	Description
1.00a	October 2014	■ 2014.10a GA release ■ Included these sections in the “Verification”. ❑ “Overview of SV-UVM Tests” ❑ “Overview of DW_axi_dmac Testbench”
1.00a-ea04	August 2014	Added the Clock Parameters block in “Signal Descriptions”.
1.00a-ea03	July 2014	■ Modified the Integration Considerations chapter ■ Modified description for the DMAX_LLI_ENDIAN_SELECTION_PIN_EN parameter
1.00a-ea02	June 2014	■ Modified address range for the Undefined Register Space in “Parameter Descriptions” chapter. ■ 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
1.00a-ea01	April 2014	■ 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
1.00a-ea00	March 2014	Initial release

Preface

This databook provides information that you need to interface the DesignWare AXI 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-manager multi-channel DMA Controller with AXI as the bus interface for data transfer. This component conforms to the *AMBA Specification*, Revision 2.0, *AMBA APB4 Protocol Specification*, 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.

Product Codes

The following table lists the product codes for the DW_axi_dmac controller.

Base License or Add-On	Product Code
Base License	
DesignWare AXI DMA Controller Source	A415-0
DesignWare AXI DMA Controller Source Plus	H679-0
Add-On	
DesignWare AXI DMA Controller Mission Critical	E137-0

Databook 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, "Mission Critical Features" describes the various mission critical features supported by the DW_axi_dmac.

- Chapter 4, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_dmac.
- Chapter 5, "Signal Descriptions" provides a list and description of the DW_axi_dmac signals.
- Chapter 6, "Register Descriptions" describes the programmable registers of the DW_axi_dmac.
- Appendix A, "Area, Speed, Power, and Performance" provides the area, power, performance, and the hardware configuration parameters that affect the performance of the DW_axi_dmac.
- Appendix B, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals, Registers, and Parameters chapters.
- Appendix C, "Glossary" provides a glossary of general terms.

Related Documentation

Refer to the following documentation:

- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2, AMBA 3, and AMBA 4 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

Web Resources

For more information, check the following Synopsys online resources:

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

Reference Documentation

The following references contain useful information concerning the protocols addressed by this IP:

- *AMBA 2 Specification*, Revision 2.0, ARM Ltd. for AHB interface
- *AMBA 4 AXI and ACE Protocol Specification*, February 2013, ARM Ltd for AXI interface
- *AMBA 4 APB Protocol Specification*, v1.0, ARM Ltd for APB4 interface

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

Replacement of Noninclusive Terminology

The following table has the list and definition of the inclusive terminology used in this document:

Table 1 Replacement of Noninclusive Terminology

Noninclusive Terms	Replacement Terms	Definition
Master	Manager, Controller, Requester	Device or model that initiates and controls another device or peripheral.
Slave	Subordinate, Target, Completer	Device or model that is controlled by and responds to a manager.

Note To comply with the AMBA Specifications and to aid in seamless integration, Synopsys will not change parameter, signal, and register names.

Customer Support

To obtain support for your product, prepare the required files and contact the support center using one of the methods described:

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

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

  a. https://solvnetplus.synopsys.com

  Note SolvNetPlus does not support Internet Explorer.

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

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

  Make sure to include the following:

  - **Product L1:** DesignWare Library IP

- **Product L2:** AMBA

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

For more information about general usage information, refer to the following article in SolvNet-Plus:

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

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood.
  - Attach any debug files you created.
- Or, telephone your local support center:
  - North America:

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

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

1

Product Overview

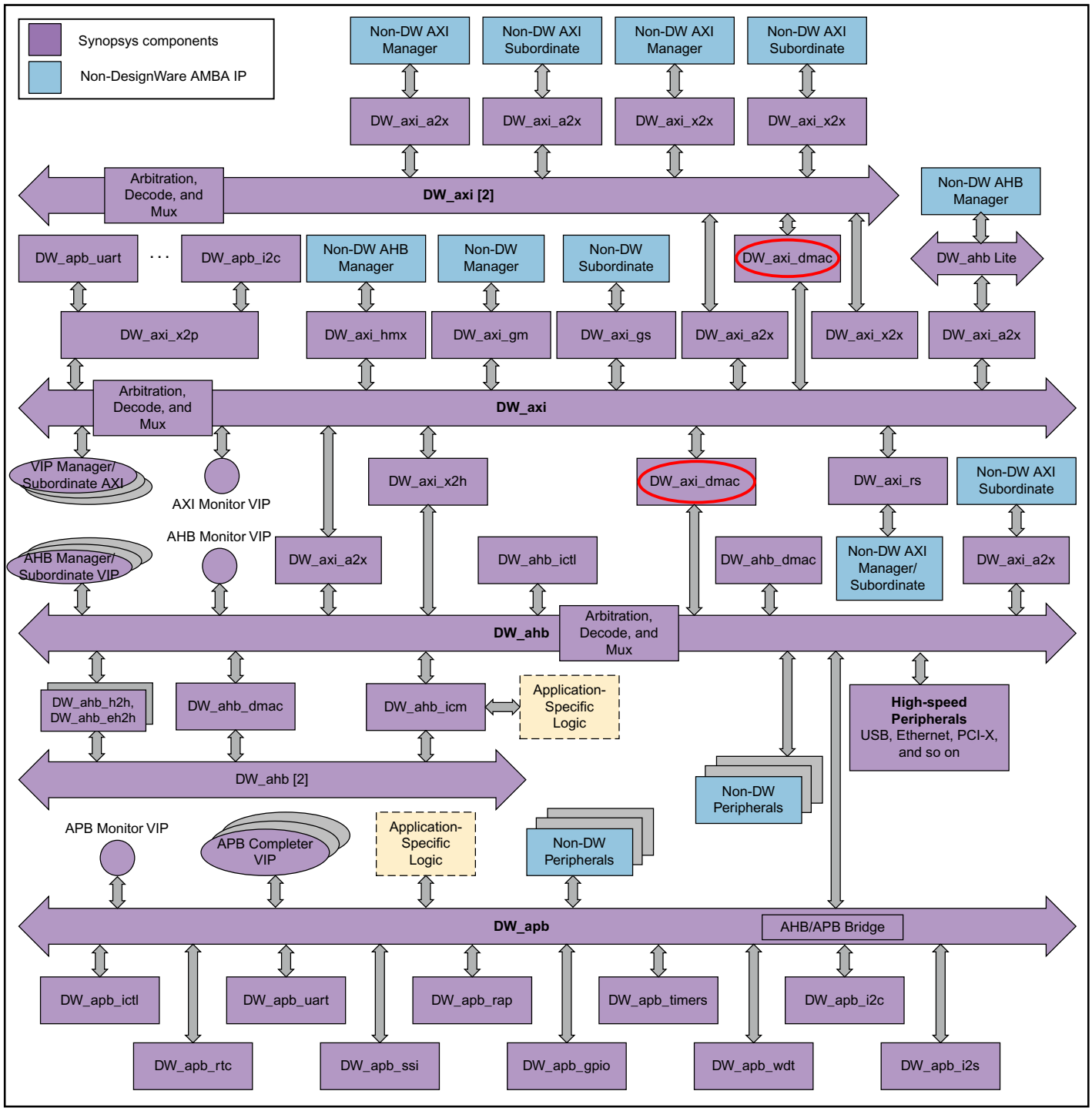
This chapter provides a basic overview of the DW_axi_dmac, which is a highly configurable, highly programmable, high performance, multi-manager multi-channel DMA Controller with AXI as the bus interface for data transfer.

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 manager/subordinate models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Figure 1-1 Example of DW_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*.

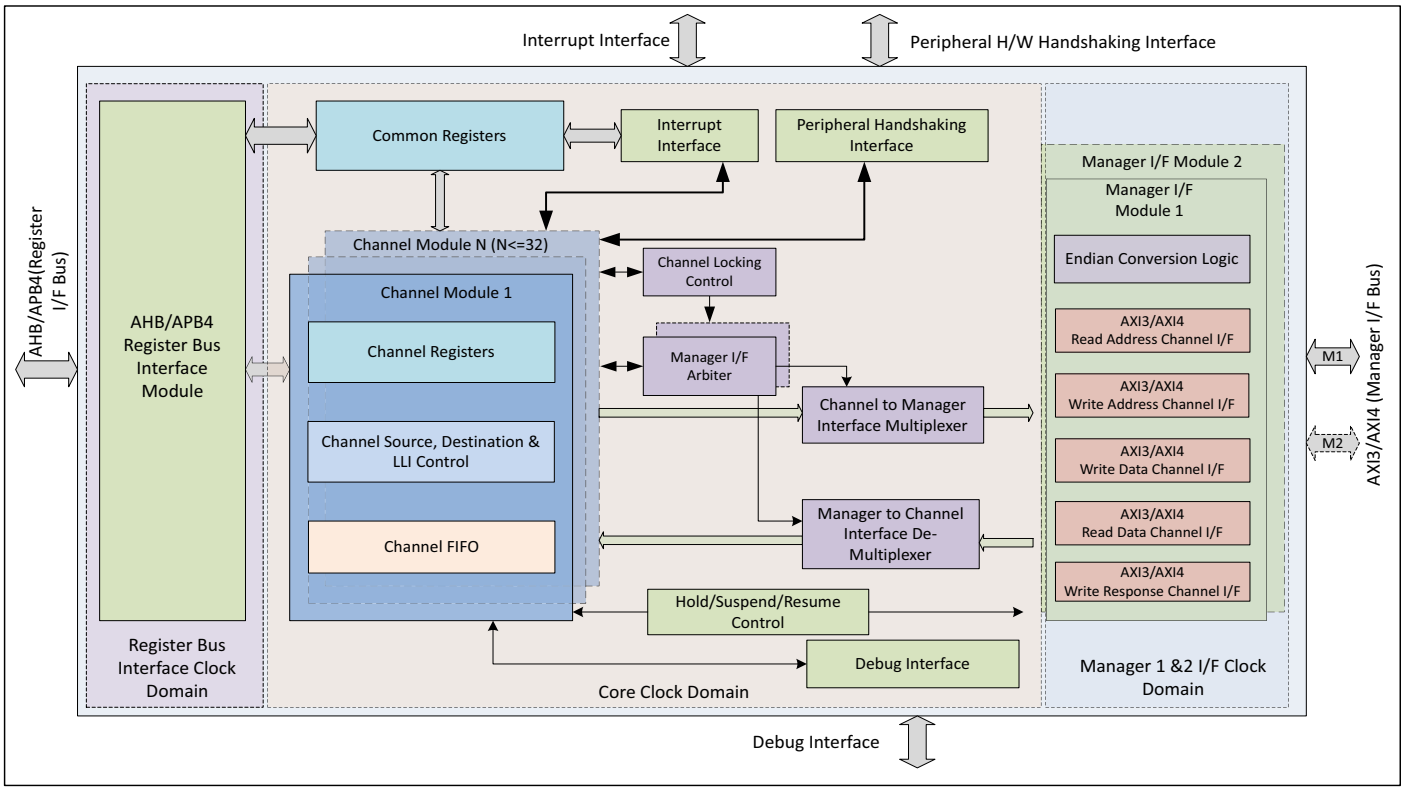
1.2 General Product Description

The Synopsys DW_axi_dmac controller 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

Figure 1-2 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, register bus interface and manager interface clocks
- Up to 32 channels, one per source and destination pair, configurable in coreConsultant
- Data transfers in one direction only (each channel is unidirectional)
- Up to two AXI manager interfaces, configurable in coreConsultant
  - Two manager interfaces for multilayer support
  - Multiple AXI managers 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
- Separate external memory interface (per channel) to connect SRAM or Register File based memories to Channel FIFO.
- AMBA 3 AXI/AMBA 4 AXI-compliant manager interface
- AHB/APB4 register bus interface for programming the DMA controller
  - AHB register bus interface supports only SINGLE transfers (hburst = 3'b000).
- AXI manager data bus width up to 1024 bits (for both AXI manager interfaces), configurable in core-Consultant
- Endian mode can be selected statically or dynamically for AXI manager interfaces, configurable in coreConsultant
- Input pin to dynamically select endianness
- Independent control for endianness of linked list access on manager interfaces, configurable in core-Consultant
- Optional identification register, configurable in coreConsultant
- Channel locking support – Supports locking of the internal channel arbitration for the manager bus interface at different transfer hierarchy
- DMAC status (idle/busy) indication outputs
- 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
- Support for Unique ID feature to generate AXI read/write transfers on the AXI Read/Write Channel with unique ID

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 manager bus interface at different transfer hierarchy
- Programmable multi-block transfer using linked list, contiguous address, auto reload, and shadow register methods
- Dynamic extension of linked list
- Independent configuration of SRC/DST multi-block 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, either source or destination peripheral 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 64 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 manager interface and AHB and APB 4 protocols for register bus interface
- Data bus width up to 1024 bits for manager interface, configurable in coreConsultant
- Outstanding transactions on manager interface
- Setting outstanding transaction limit per channel on the manager interface
- Configurable AXI transfer width
- Out-of-order transaction support for different channels connected on the same manager interface
- Incremental and fixed address transfers on manager interface
- Source and destination data transfer addresses; must be aligned to respective transfer widths
- Data bus width of 32/64 bits for register bus interface, configurable in coreConsultant
- Transfer size (width) used for register bus interface; must be same as data bus width

1.3.8 Mission Critical Features

- Parity Protection feature for all Common Registers and Channel-Specific Registers at the interface level to perform the data integrity checks using parity bits
- Error Correction Code (ECC) and Parity Protection feature corrects 1-bit errors and detects the 2-bit errors to enable protection against data corruption
  - ECC Protection Feature for FIFO Memory Interface
  - ECC Protection Feature for UID Memory Interface
  - ECC and Parity Protection Feature for AXI Manager Interface
- Lock Step feature for mission-critical applications to increase safety and reliability against unfavorable operating conditions such as Single Event Upset (SEU) and extreme EMI/EMC

1.3.9 Unsupported Features

The following features are not supported in this release:

- Bus locking on manager interface in AXI 3 mode

- Wrap address transfers

1.4 Terminology

The following are terms and 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 pairs 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.
- **Manager interface –** DW_axi_dmac is a manager 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 manager interfaces on the DW_axi_dmac, so that up to two independent source and destination channels can operate simultaneously. Each channel has to arbitrate for the manager interface. If the source and destination peripherals reside on two different AXI layers, there must be two manager interfaces.
- **Register Bus interface –** The AHB/APB4 interface over which the DW_axi_dmac is programmed. The register bus interface can be on the same layer as any of the manager 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 while the 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 –** The device (either the DW_axi_dmac, or a source or destination peripheral) that determines the length of a DMA block transfer and terminates the transfer. 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 on page 26 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

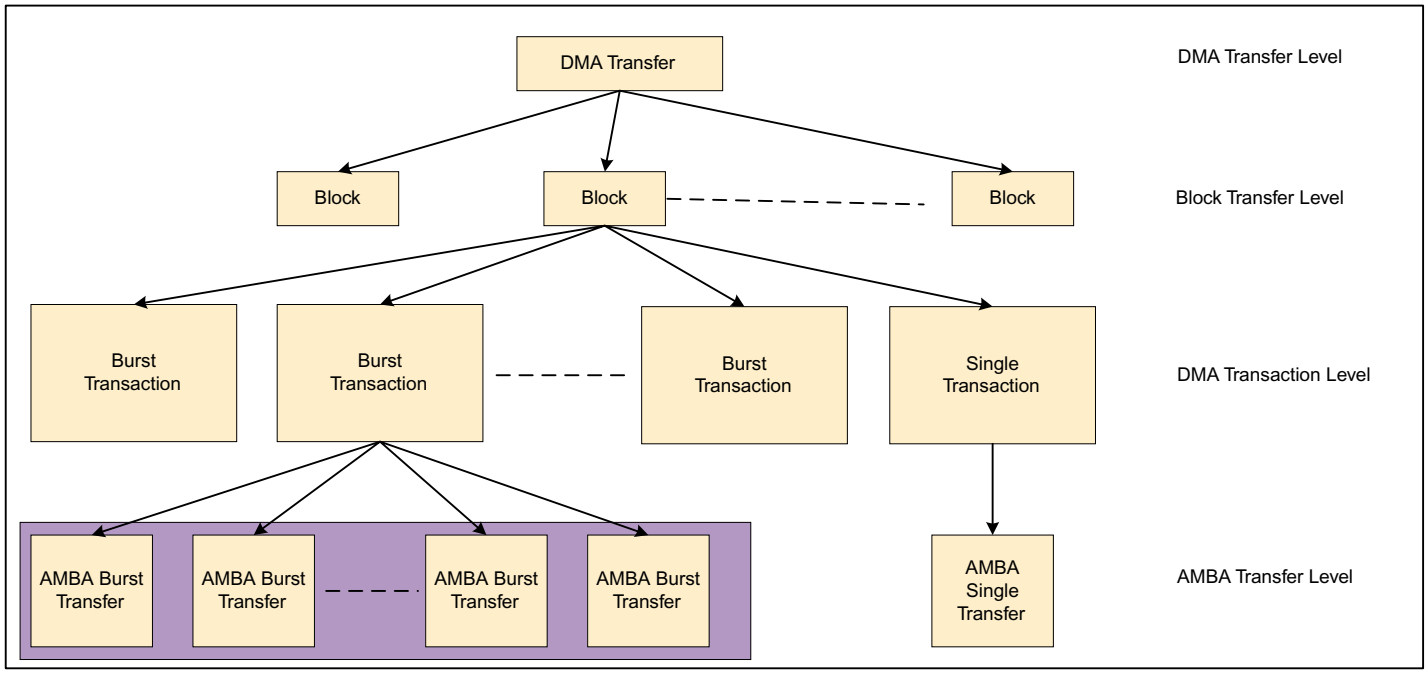
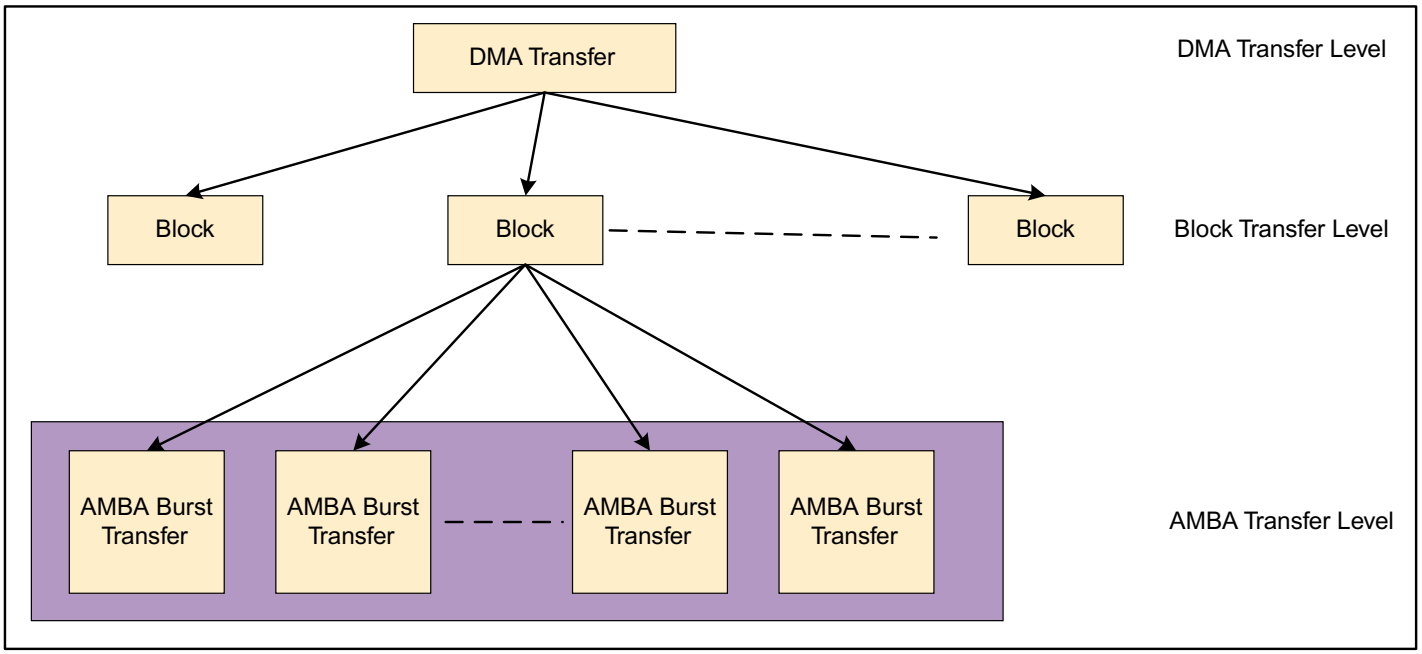


Figure 1-4 DMA Transfer Hierarchy for Memory Peripherals



- **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 manager interface by locking arbitration of the manager 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 includes an extensive verification environment, which sets up and invokes Synopsys VCS tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The “Verification” chapter 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 “Licenses” section in the *DesignWare AXI DMA Controller (DMAC) Installation Guide*.

1.8 Deliverables

DW_axi_dmac is packaged as a .run file. The license file required to use this Synopsys DesignWare IP is delivered separately. Use the Synopsys coreConsultant tool to configure, synthesize, and simulate the IP.

The DW_axi_dmac image contains the following:

- Verilog RTL source code for the DW_axi_dmac controller

  You can configure the DW_axi_dmac using the coreConsultant and coreAssembler tools to generate the RTL.
- UVM testbench and testcases to verify the configured RTL

  For more information on the UVM testbench for DW_axi_dmac, see *DesignWare AXI DMA Controller (DMAC) Databook*.
- Synthesis scripts for Synopsys Design Compiler and Synopsys Fusion Compiler; Synplify Pro Simulation regression scripts for Synopsys VCS tool

- Spyglass and VC SpyGlass Lint, CDC, and RDC checker rules used for linting, Clock Domain Crossing checks and Reset Domain Crossing checks, respectively

Note The VC VIP library license shipped with DW_axi_dmac is intended only to verify the interface-level testing of the specific IP configuration out-of-the-box in coreConsultant, to confirm that DW_axi_dmac meets the requirements. The VC VIP library license shipped with DW_axi_dmac is not intended to be used as a full featured verification model at the subsystem or SoC level.

1.9 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 the following activities, see *DesignWare AXI DMA Controller (DMAC) User Guide*:

- Configuring, synthesizing, and verifying just your DW_axi_dmac component using coreConsultant
- Implementing your DW_axi_dmac component within a DesignWare subsystem using coreAssembler

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-manager multi-channel DMA Controller with AXI as the bus interface for data transfer.

This chapter included the following topics:

- "Data Flow" on page 30
- "Register Bus Interface" on page 32
- "Manager Interface" on page 35
- "Arbitration Scheme" on page 36
- "Channel Locking" on page 44
- "Endian Scheme" on page 47
- "Single Transaction Region" on page 52
- "Handshaking Interface" on page 54
- "Interrupt Interface" on page 65
- "Flow Control Configurations" on page 67
- "Early Terminated Burst Transaction" on page 69
- "Transfer Control" on page 70
- "External Memory Interface" on page 78
- "Unique ID Feature" on page 81
- "Context Sensitive Low Power Option" on page 87
- "AXI Unaligned Transfer Support" on page 91
- "Channel Suspend, Disable, and Terminate" on page 109
- "Debug Interface" on page 114

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 on page 31 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

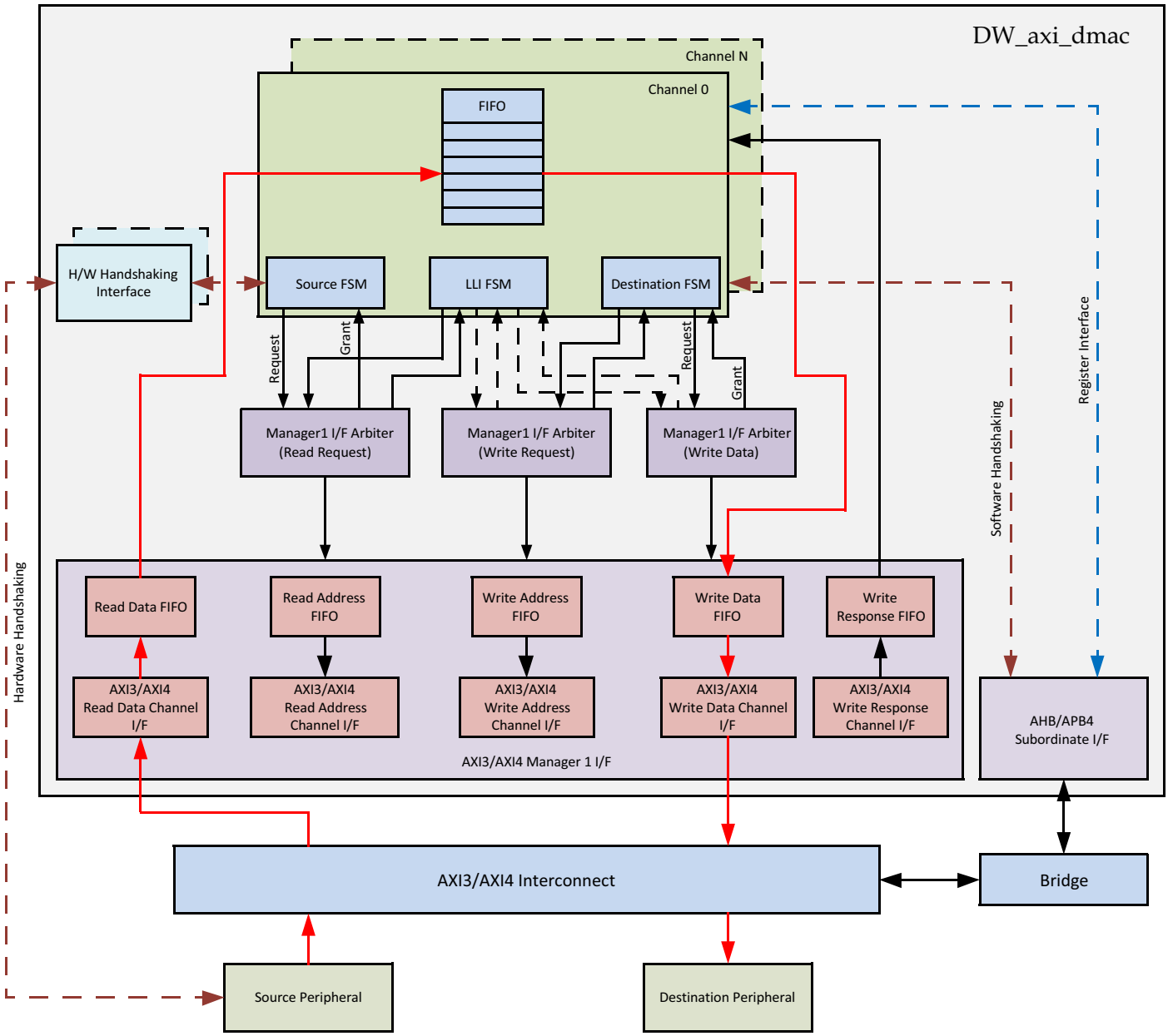
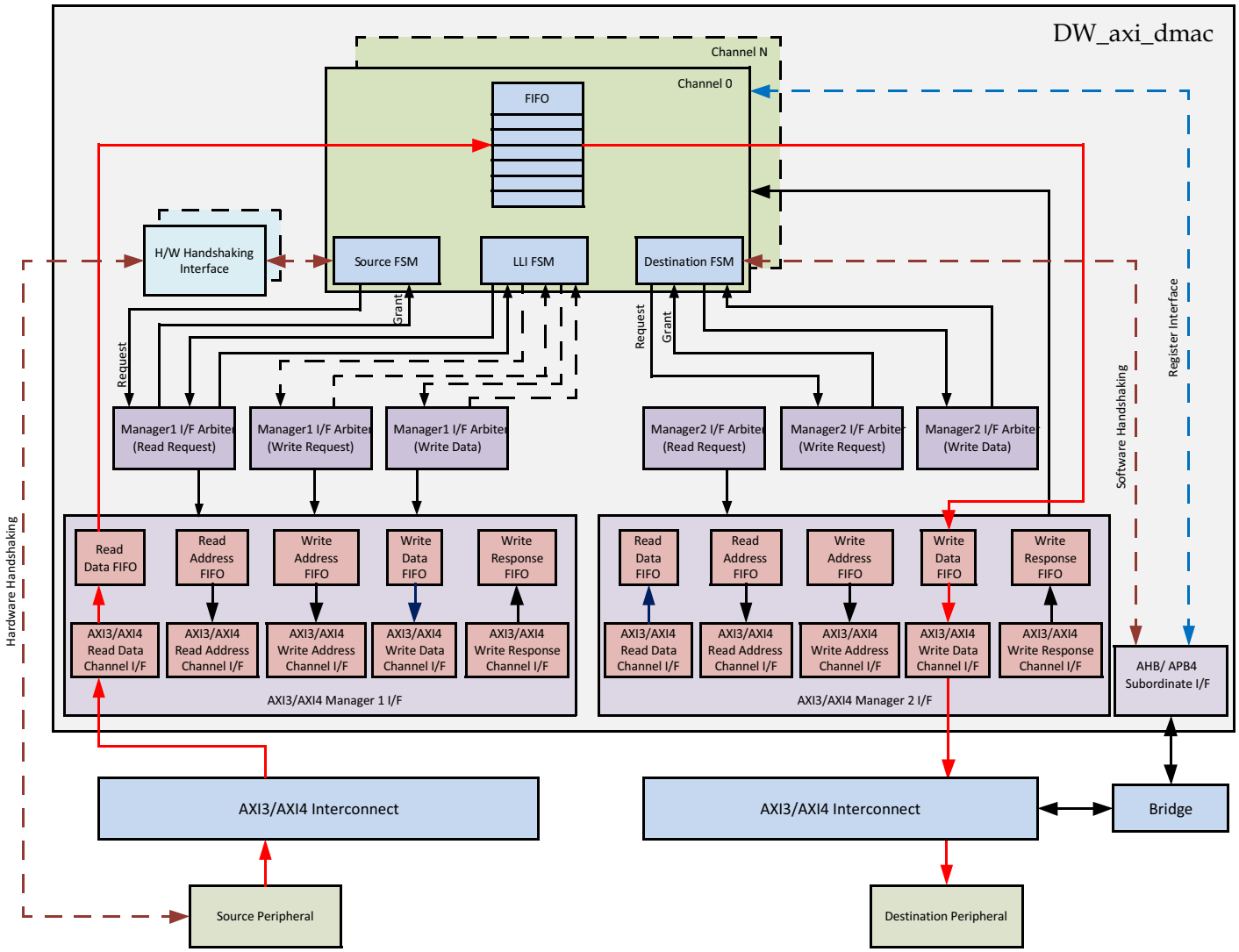


Figure 2-2 DW_axi_dmac Flow Diagram when Source and Destination Peripherals on Different AXI Layers



2.2 Register Bus Interface

2.2.1 Overview of Register Bus Interface

Register Bus Interface Module implements the logic to access the internal registers of DW_axi_dmac by an external AHB/APB4 Manager. The Register Bus interface can be configured through the `DMAX_SLVIF_MODE` parameter. Based on the subsystem requirement you can choose to have AHB/APB4 interface. These modules support only the little-endian scheme for data transfer.

The Register 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 using the `DMAX_SLVIF_CLOCK_MODE` parameter in coreConsultant. If `DMAX_SLVIF_CLOCK_MODE` parameter is set to 1, the Register Bus Interface module operates on `hclk/pclk` depending on the protocol used for this interface. The DW_axi_dmac design takes care of the clock domain crossing in this situation.

The AHB or APB4 Register Bus Interface can be used to perform the read or write operations to the Common Register and Channel Register space. While performing the read or write transfers, the Register Bus Interface might provide the Subordinate Interface access errors and read/write transactions might fail because of any of the following reasons:

- Decode Access Error
- Write to Read-Only Register Access Error
- Read from Write-Only Register Error
- Write-on-Hold Error
- Undefined Address Access Error

When Register Bus Interface is configured as AHB, the error response is provided on the `HRESP` signal by driving `ERROR`(2'h01). When Register Bus Interface is configured as APB4, the error response is provided by asserting the `PSLVERR` signal.

2.2.2 AHB Register Bus Interface

The Register bus interface can be configured to have AHB Interface to program the internal Common and Channel-specific registers of DW_axi_dmac. The AHB Register Bus Interface supports a 32-bit and 64-bit data bus width. This Register Bus Interface supports only single AHB transfer.

AHB Register Bus Interface can be configured to have a clock different than the `dmac_core_clock` using the `DMAX_SLVIF_CLOCK_MODE` coreConsultant parameter. This AHB Register Bus Interface can be configured by setting the `DMAX_SLVIF_MODE` parameter to 0.

Note The APB register bus interface can also be configured to have AHB-LITE interface to program the internal Common and Channel-specific registers of the DW_axi_dmac controller.

2.2.3 APB4 Register Bus Interface

The Register Bus Interface can be configured to have APB4 Interface to access the internal Common and Channel-specific registers of DW_axi_dmac. The APB4 Register Bus Interface supports a 32-bit data bus width (`DMAX_S_DATA_WIDTH` is always fixed to 32).

APB4 Register Bus Interface can be configured to have a clock different than the `dmac_core_clock` by using the coreConsultant parameter `DMAX_SLVIF_CLOCK_MODE`. This APB4 Register Bus Interface can be configured by setting the `DMAX_SLVIF_MODE` parameter to 2.

APB4 interface support enhances the configurability of the IP, and ease of integration in the subsystem.

2.2.4 Configuring Register Bus Interface

To configure the Register Bus Interface, choose the required option in "DW_axi_dmac Register Bus Interface Protocol" field using the Register Bus Interface Configuration tab during the **Specify Configuration** Activity in coreConsultant.

This field provides the following options:

- AHB
- APB4

2.2.5 Signal Related to Register Bus Interface

The following signals are related to the Register Bus Interface feature:

APB4 Signals

- `pclk`
- `presetn`
- `psel`
- `penable`
- `paddr[31:0]`
- `pwrite`
- `pwdata[DMAX_S_DATA_WIDTH-1:0]`
- `pstrb[3:0]`
- `prdata[DMAX_S_DATA_WIDTH-1:0]`
- `pready`
- `pslverr`

AHB Signals

- `hclk`
- `hresetn`
- `hsel`
- `haddr[31:0]`
- `hsize[2:0]`
- `htrans[1:0]`
- `hready`
- `hwrite`
- `hwdata[(DMAX_S_DATA_WIDTH-1):0]`

- `hwuser[((DMAX_S_DATA_WIDTH/8)-1):0]`
- `hrdata[(DMAX_S_DATA_WIDTH-1):0]`
- `hruser[((DMAX_S_DATA_WIDTH/8)-1):0]`
- `hresp[1:0]`
- `hready_resp`

2.3 Manager Interface

The Manager 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 manager interfaces, which operate independently to transfer data between peripherals and memories. Both Manager Interfaces must use the same AXI protocol. Address and Data bus widths of Manager Interfaces are configurable in coreConsultant, but both Manager Interfaces use the same configuration.

The Manager Interface implements all five channels specified in the AXI protocol:

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

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

The Manager 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 Manager Interface module operates on `aclk_mN`. The DW_axi_dmac design takes care of the clock domain crossing in this case.

The Manager 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 Manager Interface.

For more information about configuring or programming the Manager Interface, see the “Parameter Descriptions”, “Signal Descriptions”, and “Register Descriptions” chapters.

DW_axi_dmac supports a maximum of 32 channels, which can be assigned to either of the Manager Interfaces, based on a well-defined, programmable arbitration scheme for accessing the Manager Interface. Apart from channel source and destination peripherals, LLI fetch and LLI write-back operations also need access to the Manager Interface. DW_axi_dmac implements a programmable arbitration scheme which is explained in the following section.

2.4 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

The DW_axi_dmac controller implements an arbitration scheme, with options for channel locking at different DMA transfer hierarchy levels. The arbitration scheme implemented based on the Dynamic Priority with options to have different second tier arbitration.

The Dynamic Priority arbitration uses the priority inputs of the requests and decides which one of the requesting clients is issued the grant signal. The channel request with the highest priority is granted access to the AXI channel on the manager interface. 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 (`DMAX_NUM_CHANNELS-1`) is the highest priority, and 0 is the lowest.

If two or more requests to the arbiter have the same programmed priority value and there are no other client requesting at a higher priority level, the second-tier arbitration kicks in. The DW_axi_dmac supports two types of the second-tier arbitration schemes, specifically:

- Fair-among-equals Arbitration
- Round Robin Arbitration

2.4.1 Fair-Among-Equals Arbitration

After second-tier arbitration kicks in, the fair-among-equal arbitration scheme is used when `DMAX_ARB_TYPE` is configured with Dynamic Priority Arbitration with Fair-Among-Equals (0). The fair-among-equal arbitration scheme is one in which the grant is issued fairly, on a cycle-by-cycle basis for one cycle each, among the channels with same priority level as long as these clients are actively requesting.

For the fair-among-equal arbitration, every input request of arbiter maintains an internal register containing an internal priority (values from `0` to $2^{\texttt{ceil(log2(DMAX_NUM_CHANNELS))}}$) that is updated each cycle. For each channel request, the internal priority is appended to the external input priority and the result is used by the arbiter to decide which of the requesting channel gets the grant.

The internal priorities are updated based on the below criteria:

- The non-requesting channels have their internal priorities set to lowest value.
- The internal priorities of actively requesting channels is increased by one each cycle until granted.
- The internal priority is rolled over to lowest priority value when highest priority is reached but the request is not granted. This happens when a channel whose external priority is lower than currently granted and it has not been granted the resource in last $2^{\texttt{ceil(log2(DMAX_NUM_CHANNELS))}}$ cycles.
- The internal priority of the currently granted channel is set to the lowest value in the next cycle.

A potential deadlock condition could occur if two or more channels of the arbiter have the same programmed priority value and assert the request during the same cycle. In such cases, the arbiter uses the index of requests as a tie-breaker among the requesting clients to grant a specific requesting channel.

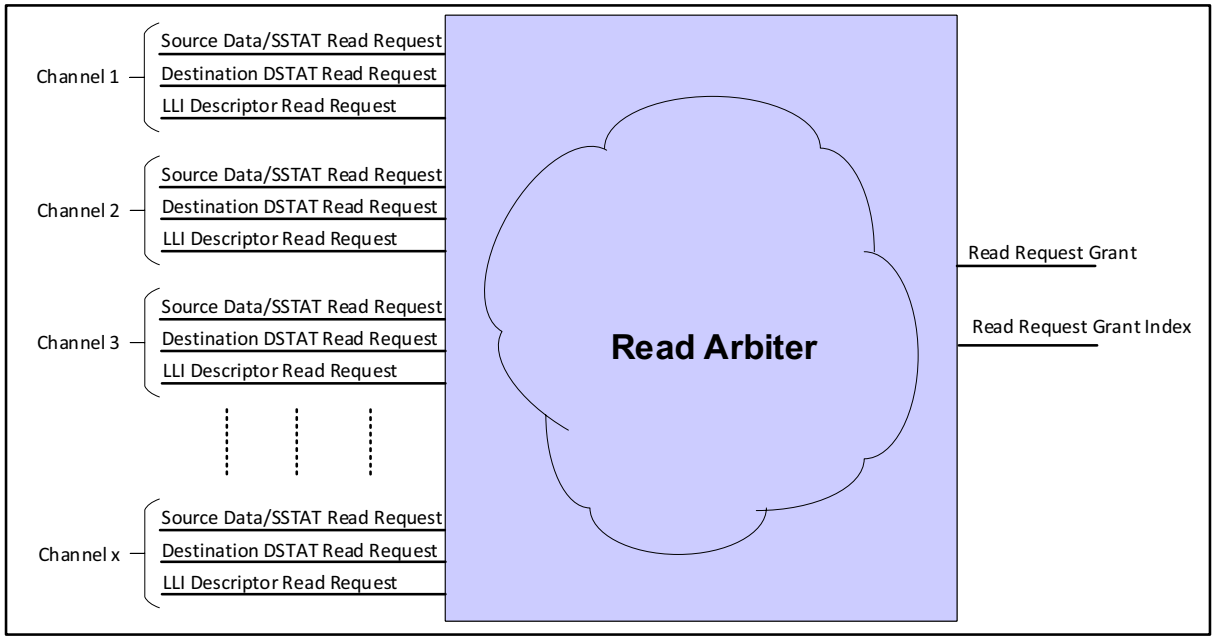
2.4.1.1 Single Arbiter Scheme

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 36. The granted read request gets the access to the AXI Manager 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



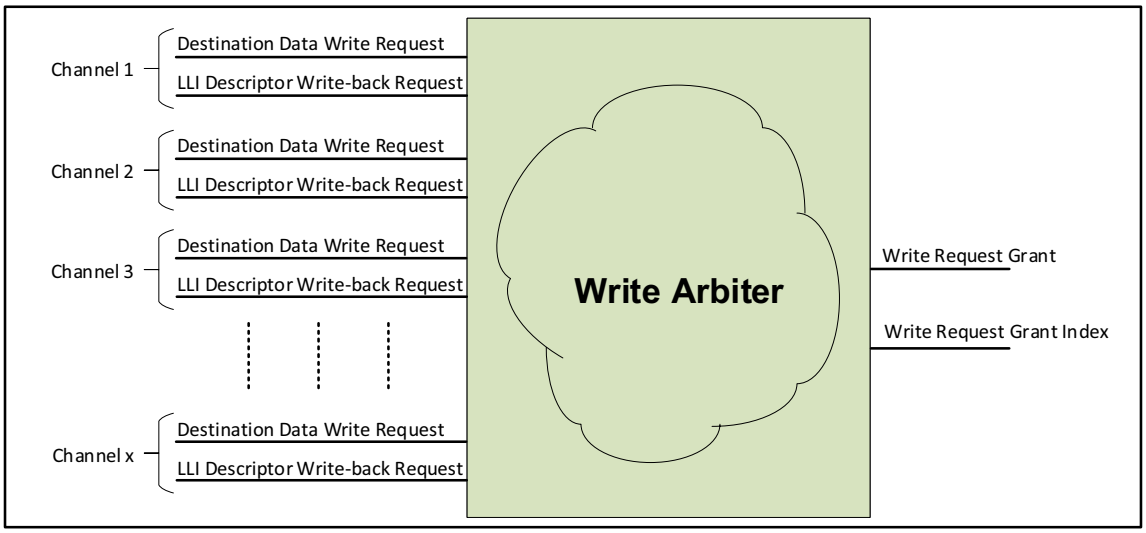
Where, `x = DMAX_NUM_CHANNELS` that is configured number of Channels.

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 Manager Interface. The block diagram of the Single Arbiter Scheme - Write Arbiter is as shown in Figure 2-4 on page 38.

Figure 2-4 Single Arbiter Scheme – Write Arbiter



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

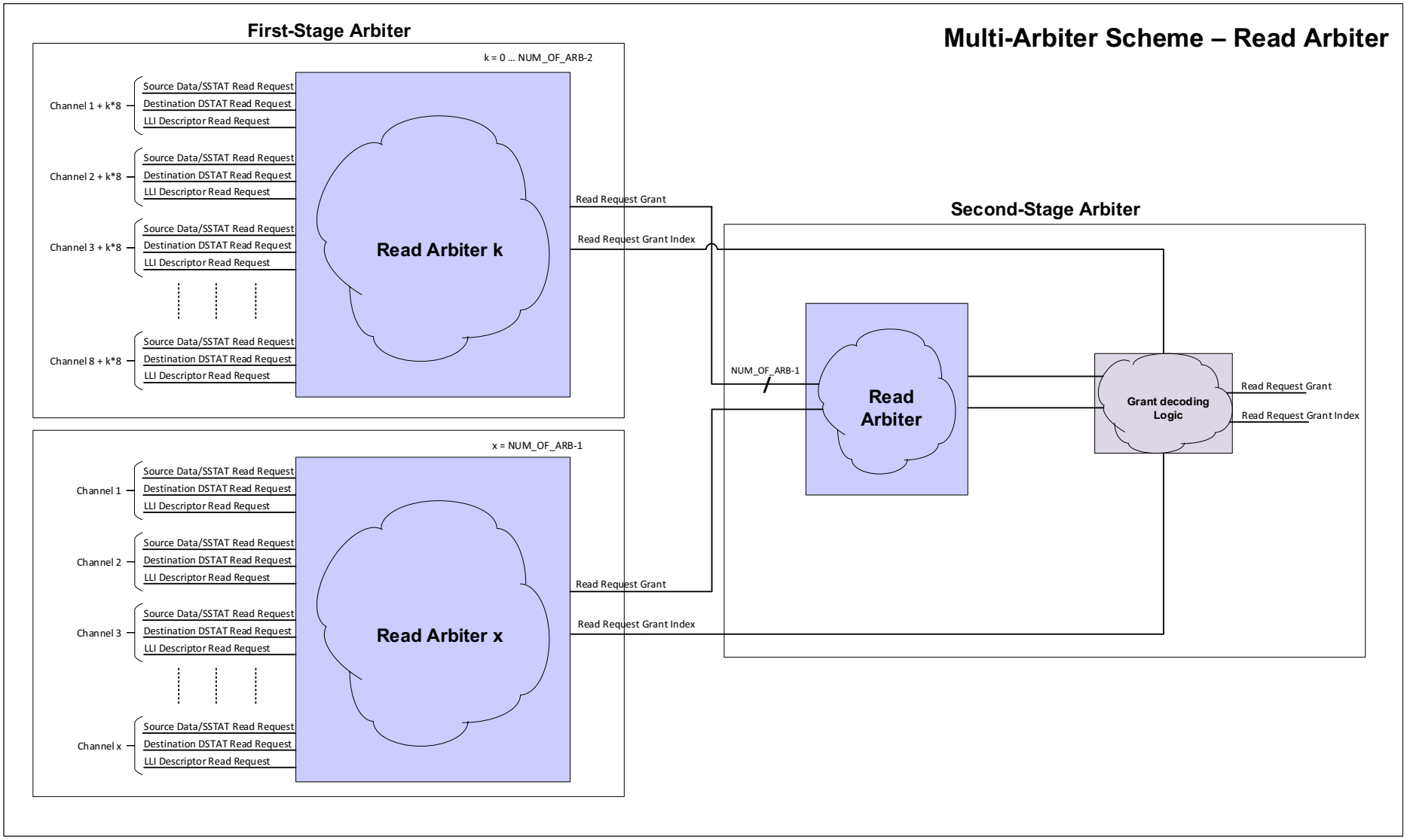
If the number of channels that is `DMAX_NUM_CHANNELS` are less than or equal to eight, then DW_axi_dmac is configured with Single Arbiter Scheme that is `DMAX_MULT_ARB_EN` is always hardcoded to 0. If the number channels are greater than eight, then DW_axi_dmac can be configured with Single Arbiter Scheme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

2.4.1.2 Multi-Arbiter Scheme

When the number of channels are greater than eight, 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 eight 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 37. 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 on page 39 and Figure 2-6 on page 40. 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 on page 39. 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.

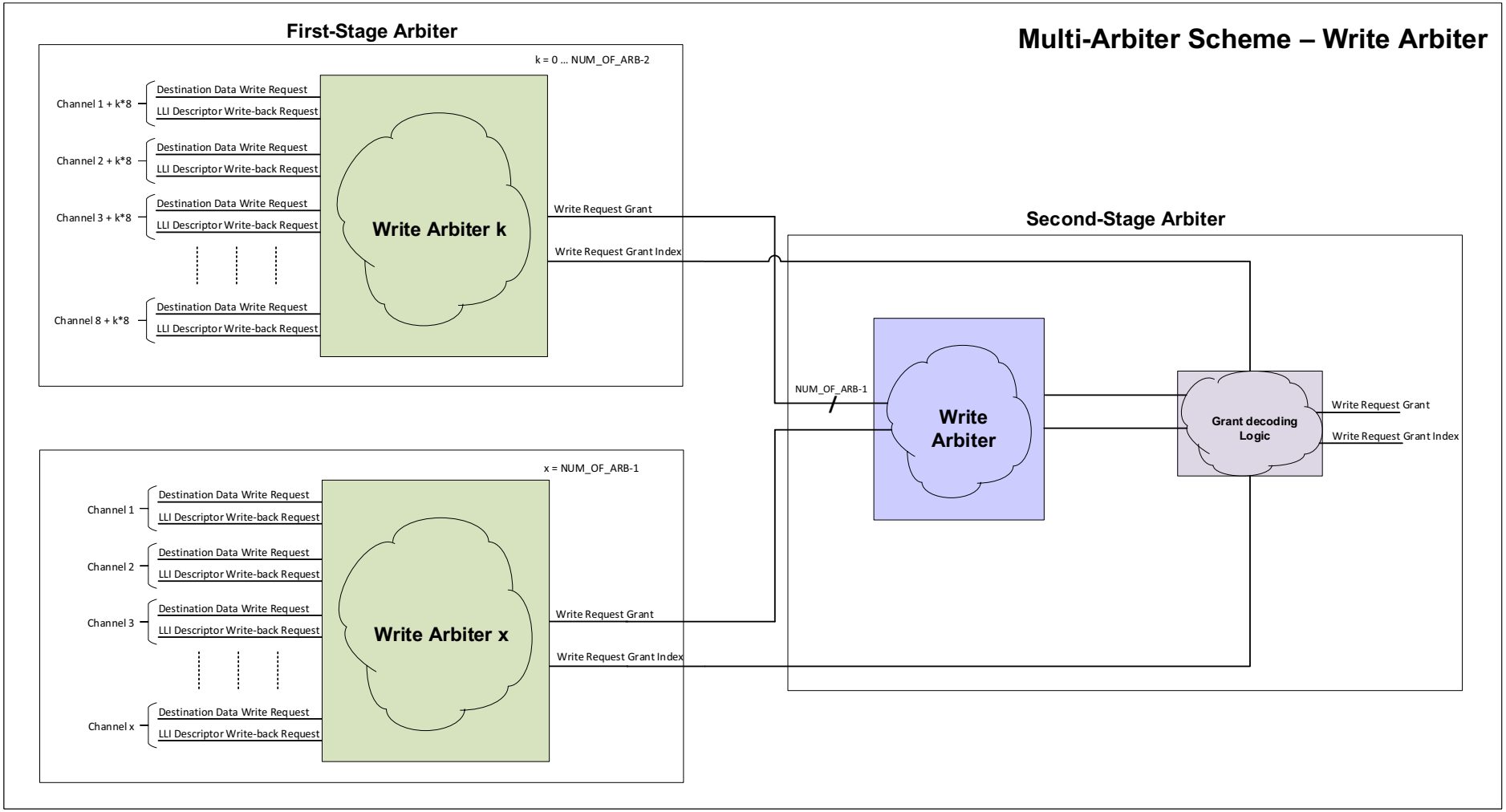
Figure 2-5 Multi-Arbiter Scheme – Read Arbiter



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`

Table 2-1 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 eight Channels	4
27	4	First three Arbiters receives request from eight Channels Last Arbiter receives request from three Channels	4
21	3	First two Arbiters receives request from eight Channels Last Arbiter receives request from five Channels	3
12	2	First Arbiter receives request from eight Channels Second or last Arbiter receives request from four Channels	2

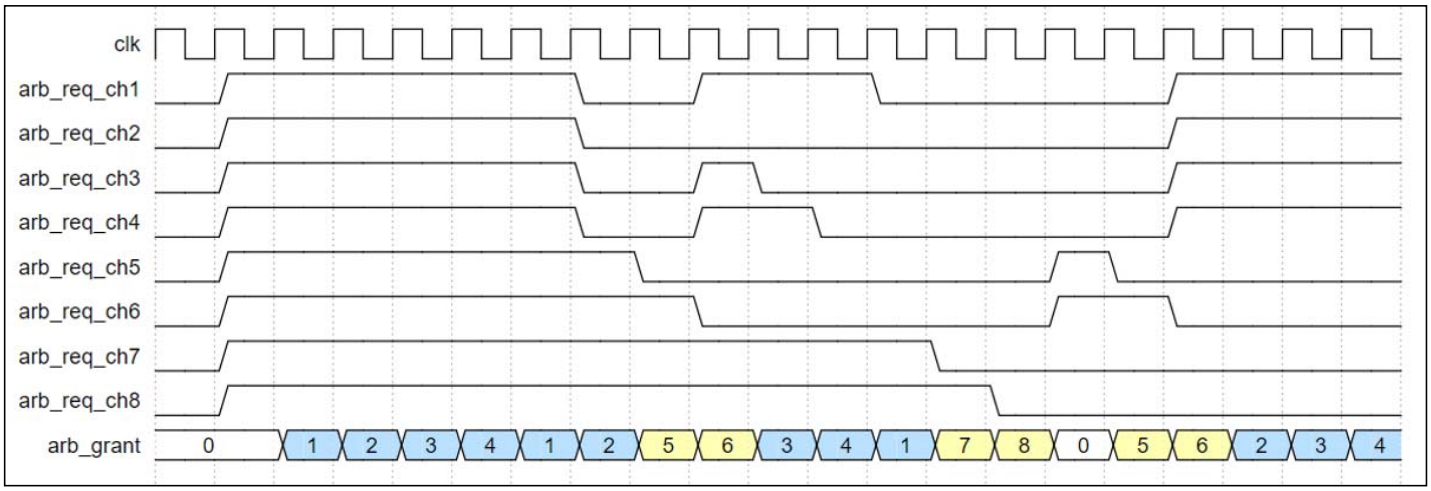
2.4.2 Round Robin Arbitration

When second-tier arbitration starts, the Round Robin arbitration scheme is used if DMAX_ARB_TYPE parameter is configured with Dynamic Priority Arbitration with Round Robin (1). In Round Robin arbitration scheme, the grant is issued to the next sequentially numbered requesting channel among the channels with same priority level as long as these clients are actively requesting.

In the Round Robin Arbitration, initially the arbiter grants access to the lowest numbered channel for which arbiter request is present. Subsequently, the next sequentially numbered channel with the request gets the grant, regardless of whether the last-granted channel releases its request or not. Thus, priority and grant are passed in "Round Robin" fashion to each of the requesting channel in turn.

Consider an example with 4 channels. Say, Channel 2 and 3 have equal and highest priority. Channel 1 and 4 have the lowest priority.

Figure 2-7 Round Robin Arbitration – Example 1



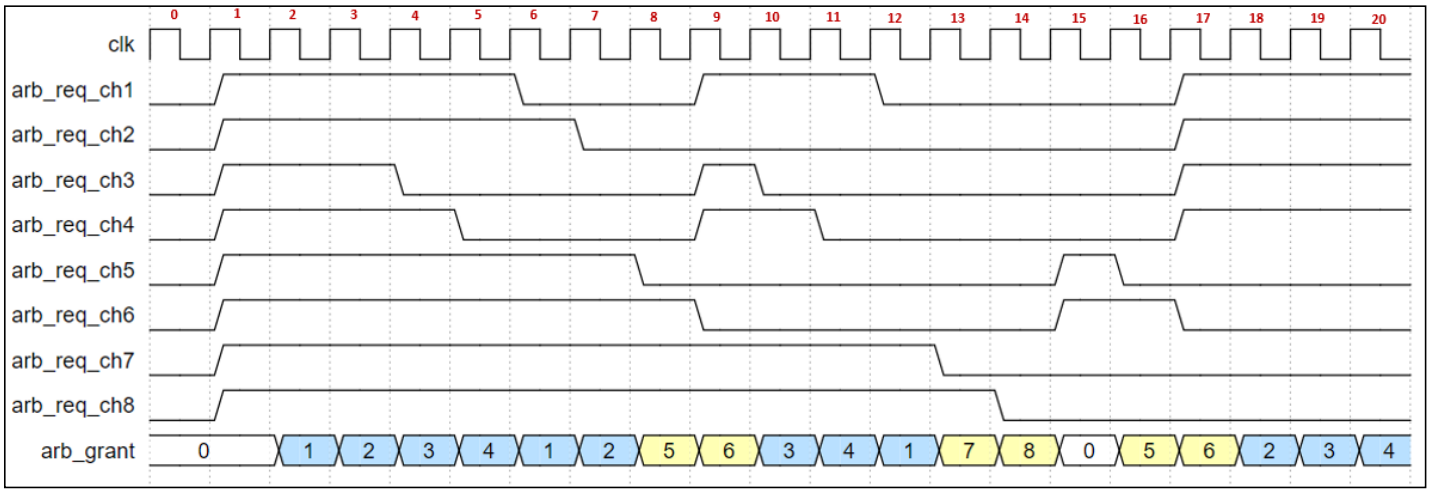
The `arb_req_ch(x)` signal is the arbitration request from Channel `x`, which is an internal signal. The grant corresponding to the current cycle is provided in the next cycle after registering the grant, through an internal signal registered `arb_grant`. These signals are used to explain the operation of the Round Robin algorithm.

- In **cycle 1**, channel 1, 2, and 3 requests are asserted at the same time. In the current arbitration cycle, based on Dynamic Arbitration, channel 2 and 3 get selected for the second-tier Round Robin arbitration. The Round Robin arbitration provides grant for channel 2 as it is the lowest-numbered channel with the request. The registered grant is available in cycle 2.
- In **cycle 2**, channel 1, 3, and 4 requests are asserted. In the current cycle, based on Dynamic Priority arbitration, channel 3 which has the highest priority gets the grant.
- In **cycle 3**, channel 1 and 4 requests are asserted. Based on the Dynamic Priority arbitration, channel 1 and 4 get selected for the second-tier arbitration. The Round Robin arbitration provides grant for channel 1 as it is lowest-numbered channel in the current group of channels that entered the second-tier arbitration.
- In **cycle 4 and 5**, only channel 4 request is asserted. Therefore, based on the Dynamic Priority arbitration, channel 4 gets the grant in the cycle 5 and 6.
- In **cycle 6**, the channel 1 and 4 requests are asserted. The channel 1 gets the grant based on the Round Robin arbitration because it is the next sequentially-numbered channel with request.
- In **cycle 7**, channel 1, 2, 3, and 4 requests are asserted. The channel 2 and 3 gets selected for the second-tier arbitration. The Round Robin arbitration checks which channel was granted in the last arbitration cycle in this group of channels (that is, channel 2 and 3) with the same priority. In this case, it was channel 3. Therefore, in the current Round Robin arbitration cycle, channel 2 gets the grant as it is next sequentially-numbered channel with the request.
- In **cycle 8**, channel 1, 2, 3, and 4 requests are asserted. Similar to cycle 7, the channel 2 and 3 get selected for the second-tier arbitration. In the previous arbitration cycle between channel 2 and 3, because channel 2 was granted, in this cycle channel 3 gets the grant as it is next sequentially-numbered channel.
- In **cycle 9**, the channel 1, 2, and 4 requests are asserted. The highest priority channel that is, channel 2 gets the grant based on the Dynamic priority arbitration.
- In **cycle 10**, the channel 1 and 4 requests are asserted. Channel 4 gets the grant based on the Round Robin arbitration because in the last arbitration cycle between channel 1 and 4, channel 1 was granted.
- In **cycle 11**, the channel 1 and 4 requests are asserted. Channel 1 gets the grant based on the Round Robin arbitration.
- In **cycle 12**, only channel 4 request is asserted. As there are no requests from other channels, channel 4 gets the grant.

In this example, the second-tier Round Robin arbitration happens within two group of channels with same priority (that is, one group with channel 2 and 3, and second group with channel 1 and 4). However, DW_axi_dmac supports Dynamic Priority arbitration with second-tier Round Robin arbitration with all possible channel groups with legal priority levels.

Consider another example with 8 channels. Say, channel 1, 2, 3, and 4 have the equal and highest priority. Channel 5, 6, 7, and 8 have the lowest priority. In this scenario also, arbitration happens similar to the previous example.

Figure 2-8 Round Robin Arbitration – Example 2



2.4.3 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 (not supported 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 Manager Interface until the end of locking period, and requests from other channels during this time are ignored.

2.4.4 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 Manager 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)

2.4.5 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.

2.5 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 Manager Interface until the end of the locking period, and requests from other channels during this time are ignored.

2.5.1 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, a lock of the corresponding channel to the AXI Manager Interface is established.
- Transfer is a memory-to-memory transfer.

Note 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.

2.5.2 Clearing Channel Locking

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

2.5.3 Possible Deadlock Conditions

A deadlock condition can occur when multiple channels are enabled concurrently on a Manager Interface and one of these channels has obtained a read lock on the Manager Interface and the remaining channels are locked out from proceeding with a DMA transfer. While programming, you must ensure that deadlock does not occur. A deadlock can occur only for configurations where the number of managers are equal to 2 (`DMAX_NUM_MASTER_IF = 2`) and the number of channels are more than 1 (`DMAX_NUM_CHANNELS > 1`).

Figure 2-9 on page 45 illustrates a scenario in which the read channel is locked out for subsequent read requests.

Figure 2-9 Scenario 1: Deadlock Condition

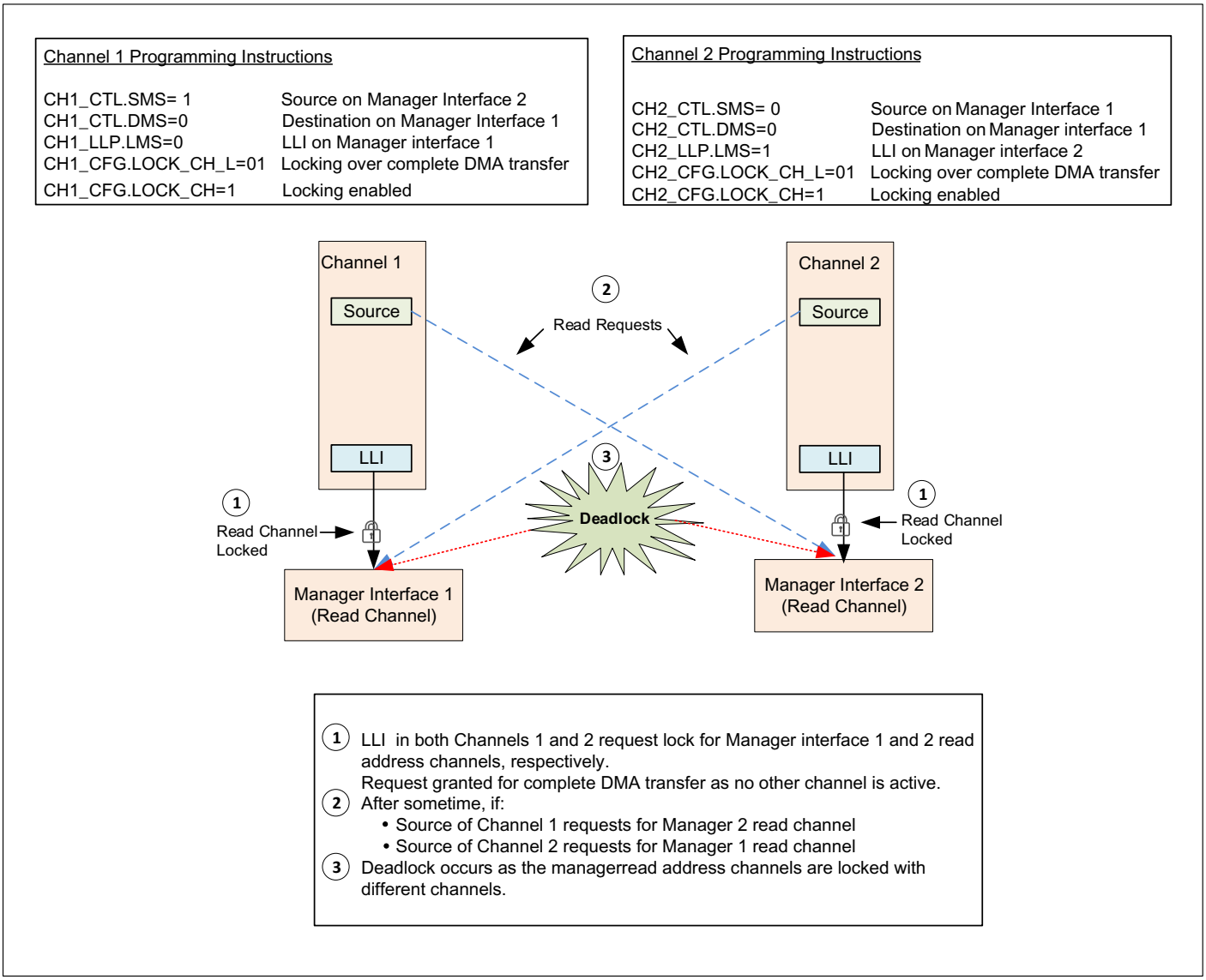
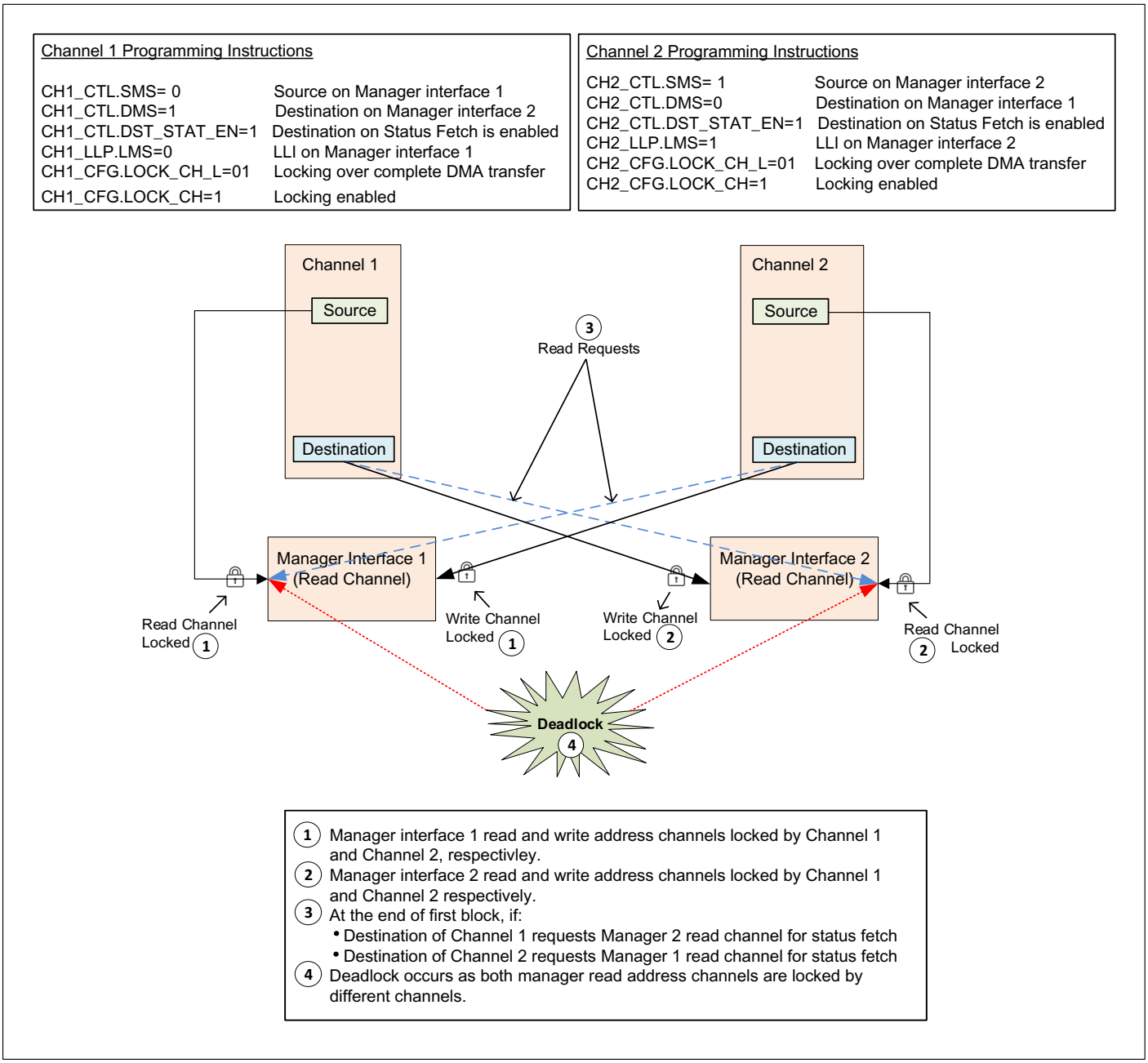


Figure 2-10 on page 46 illustrates another scenario in which the read channel is locked out for subsequent read requests.

Figure 2-10 Scenario 2: Deadlock Condition



2.6 Endian Scheme

DW_axi_dmac supports little-endian and big-endian format for data access on the AXI Manager Interface. The endianness 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 Manager 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.

From Figure 2-11 to Figure 2-22 on page 50 show the conversion process between big-endian BE-8 and little-endian data appearing on the AXI Manager Bus for different data bus width and transfer size combinations.

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

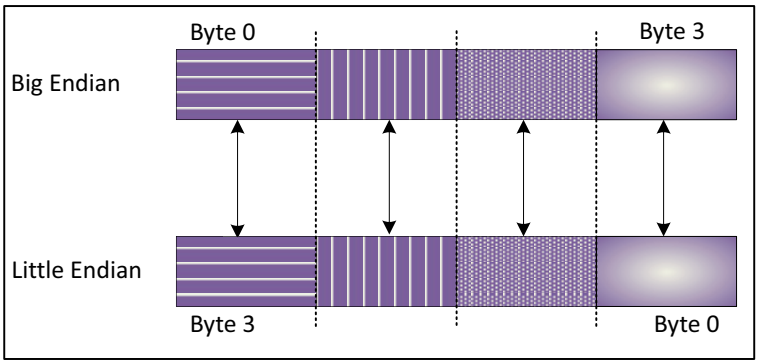


Figure 2-12 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 32-Bit Data Bus (Half Word Access)

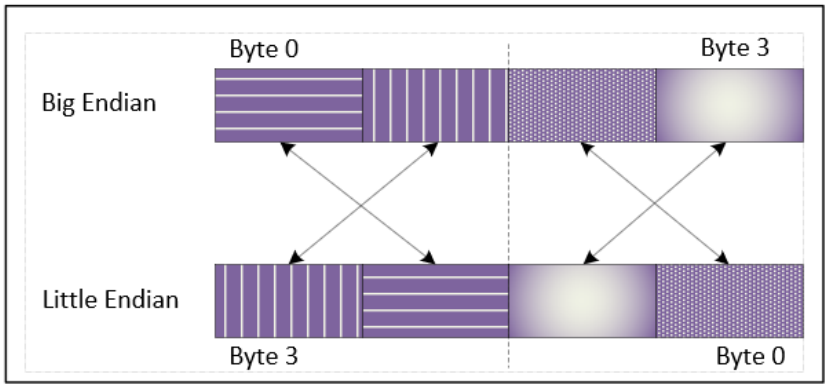


Figure 2-13 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 32-Bit Data Bus (Word Access)

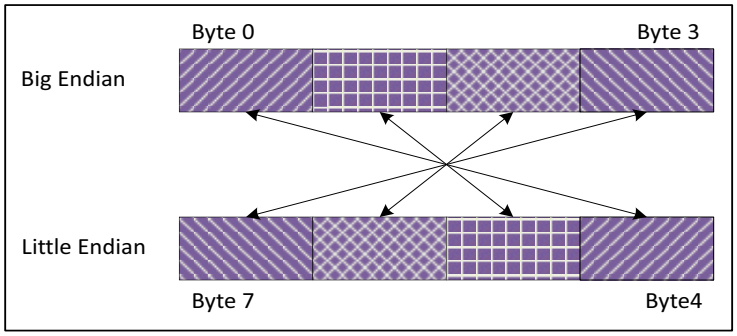


Figure 2-14 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Byte Access)

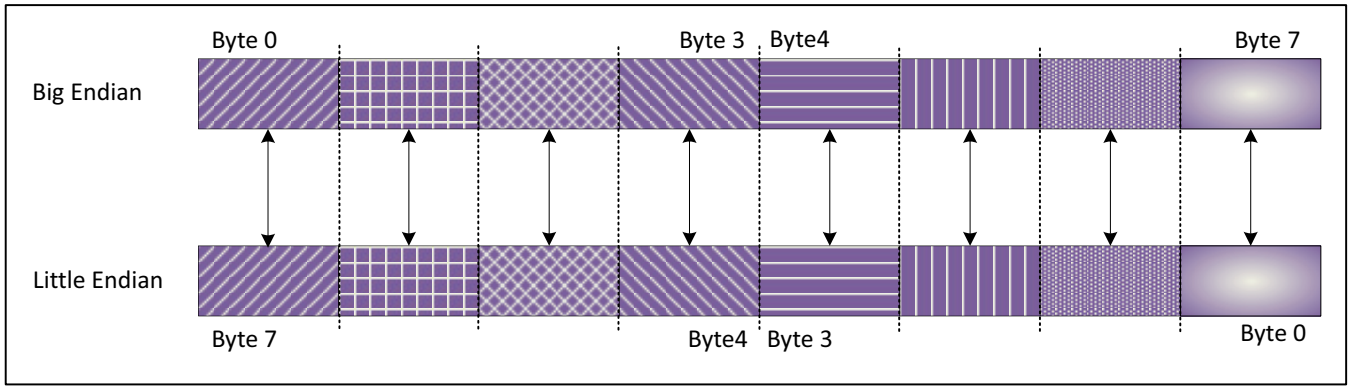


Figure 2-15 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Half Word Access)

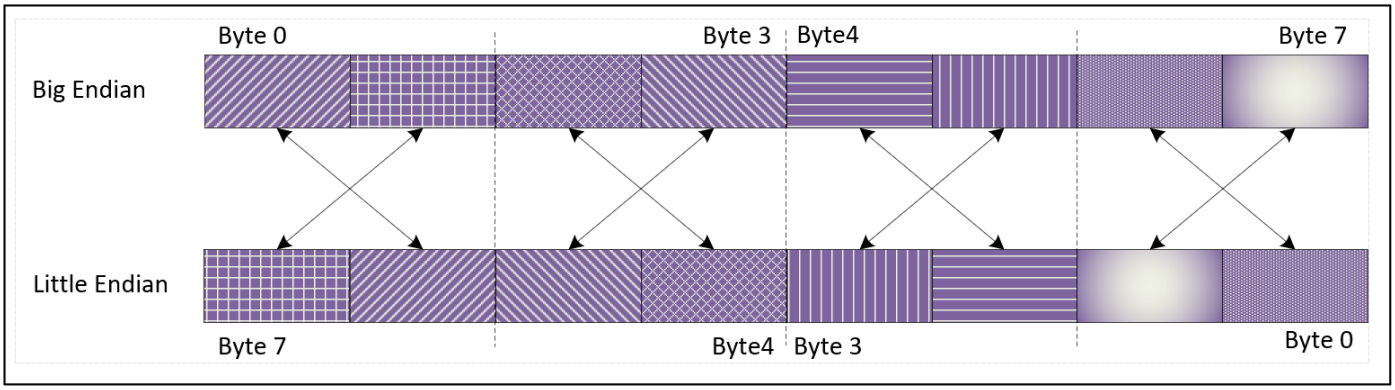


Figure 2-16 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Word Access)

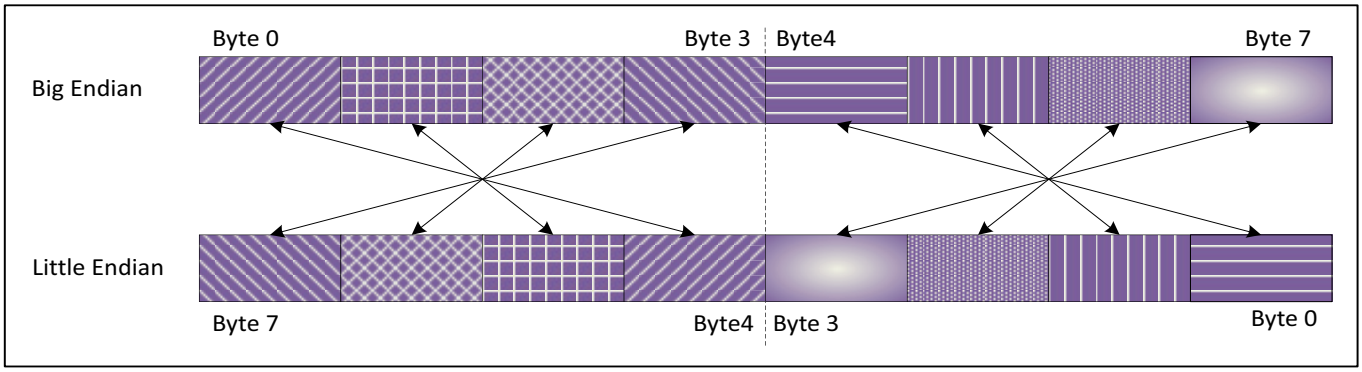


Figure 2-17 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 64-Bit Data Bus (Double Word Access)

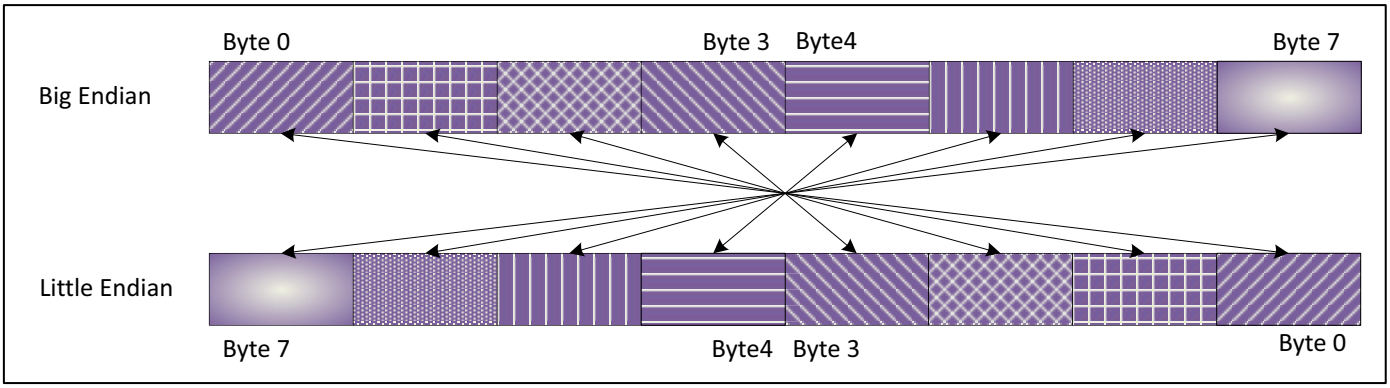


Figure 2-18 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Byte Access)

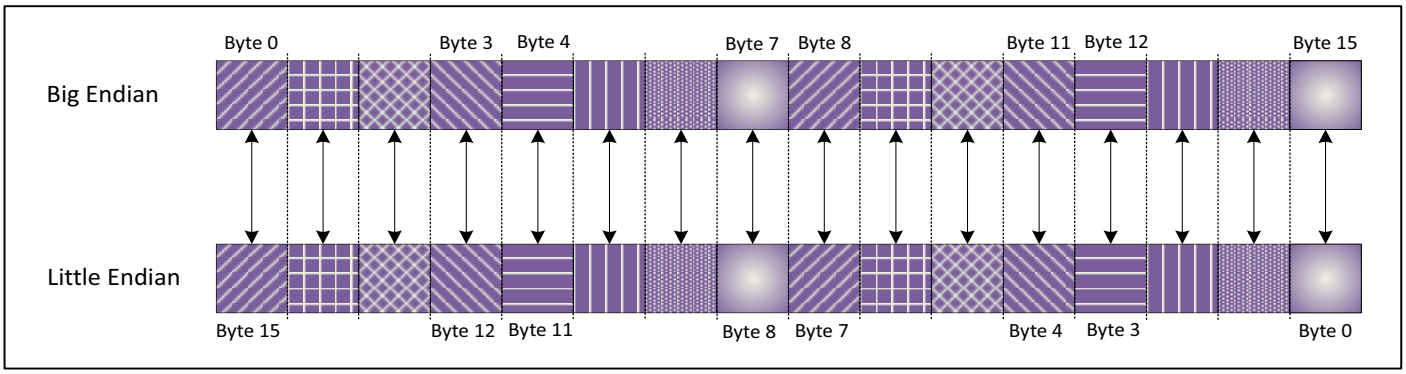


Figure 2-19 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Half Word Access)

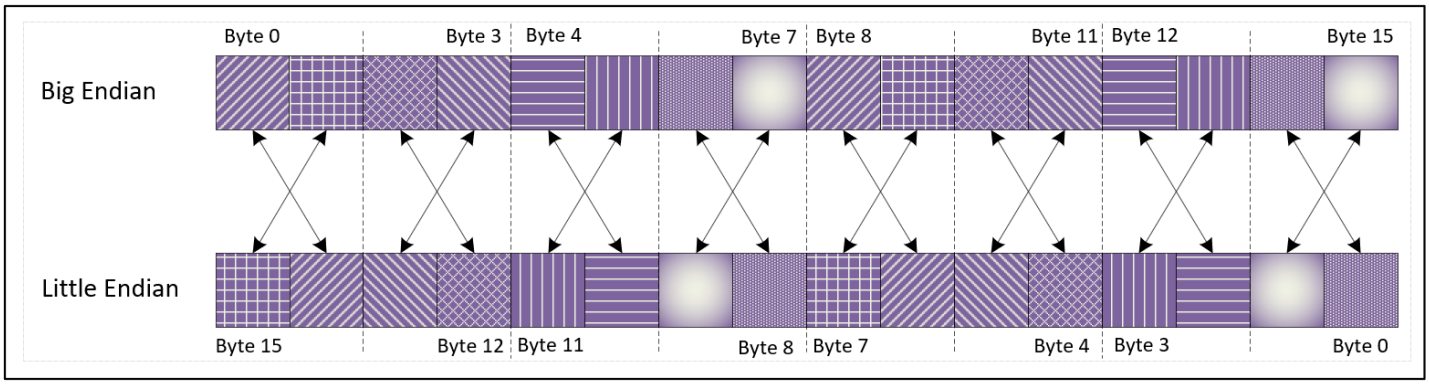


Figure 2-20 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Word Access)

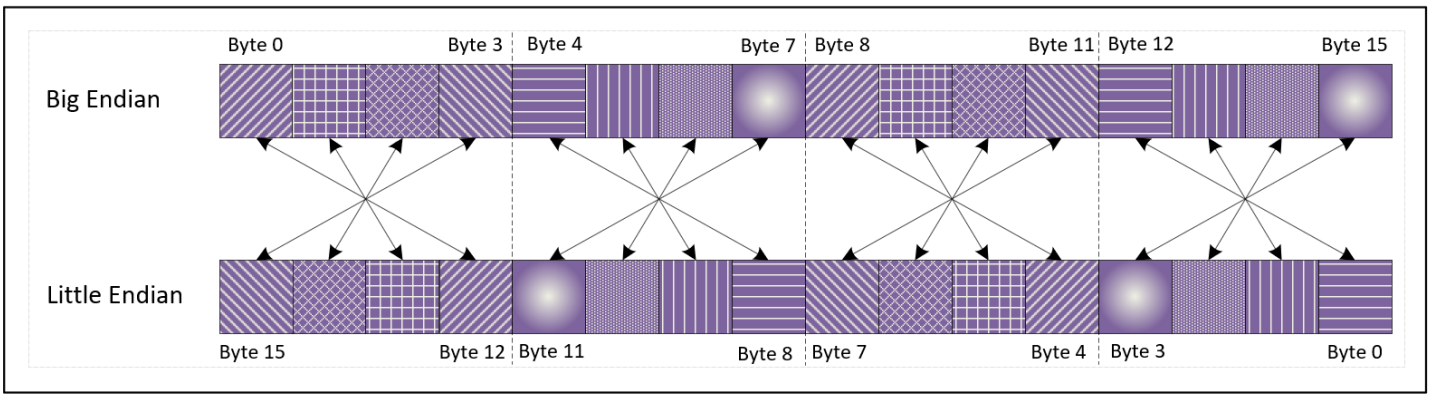


Figure 2-21 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (Double Word Access)

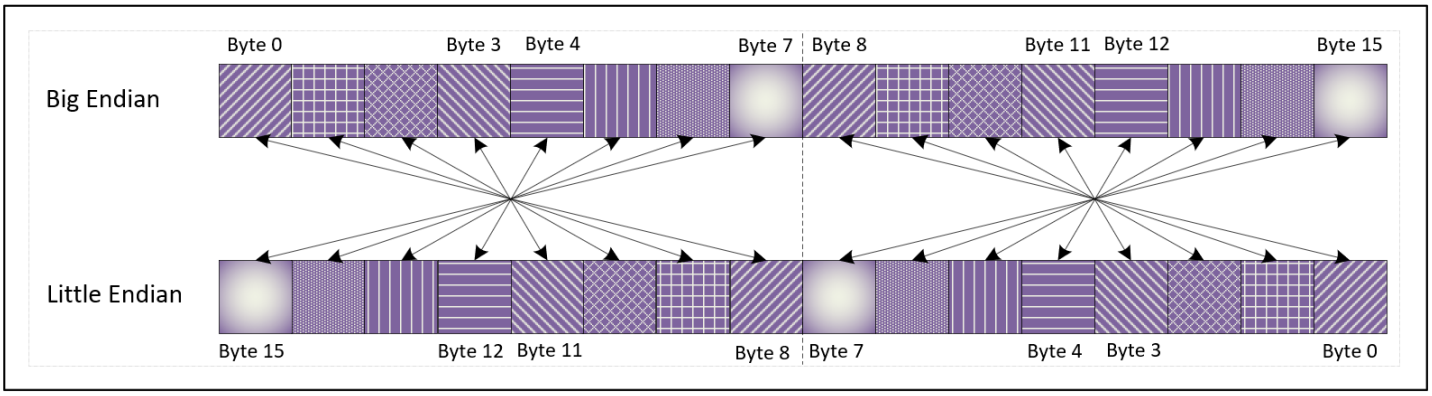
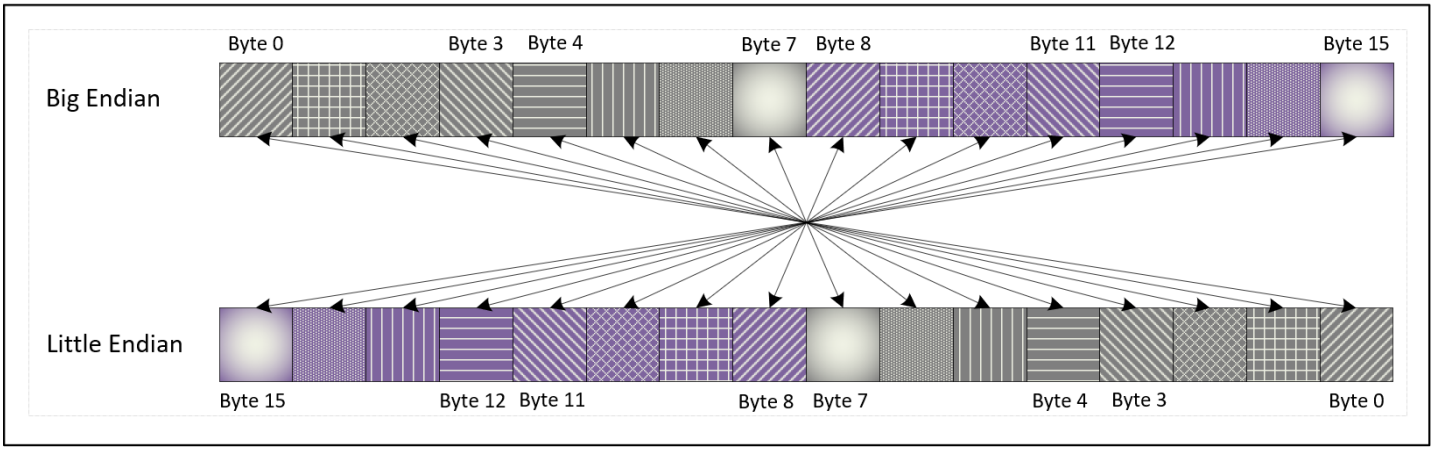


Figure 2-22 BE-LE Conversion Using BE-8 (Byte Invariant) Method for 128-Bit Data Bus (128-Bit Access)



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 schemes used for the source and destination data transfer, LLI fetch, and status and control write-back access on the Manager Interfaces are 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 Manager Interface.

Table 2-2 Endian Formats Supported on Manager Interface

DMAX_STATIC_ENDIAN_SELECT_MSTIF	DMAX_ENDIAN_FORMAT_MSTIF	dmac_endian_format_mstif_m1/m2	Endian Scheme Used for SRC/DST Data Access on M1/M2 Manager Interface	DMAX_LLI_ENDIAN_SELECTION_PIN_EN	dmac_le_select_lli_mstif_m1/m2	Endian Scheme Used for LLI Fetch/ Status and Control Write Back Access on M1/M2 Manager 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

2.7 Single Transaction Region

In certain cases, a DMA block transfer cannot complete using only burst transactions. 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 remaining 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 outside this region.

The following terms are used for explaining the 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 the DW_axi_dmac is 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 the 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 the 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`.

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

2.8 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 64 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 dedicated handshaking interfaces.

2.8.1 Hardware Handshaking

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

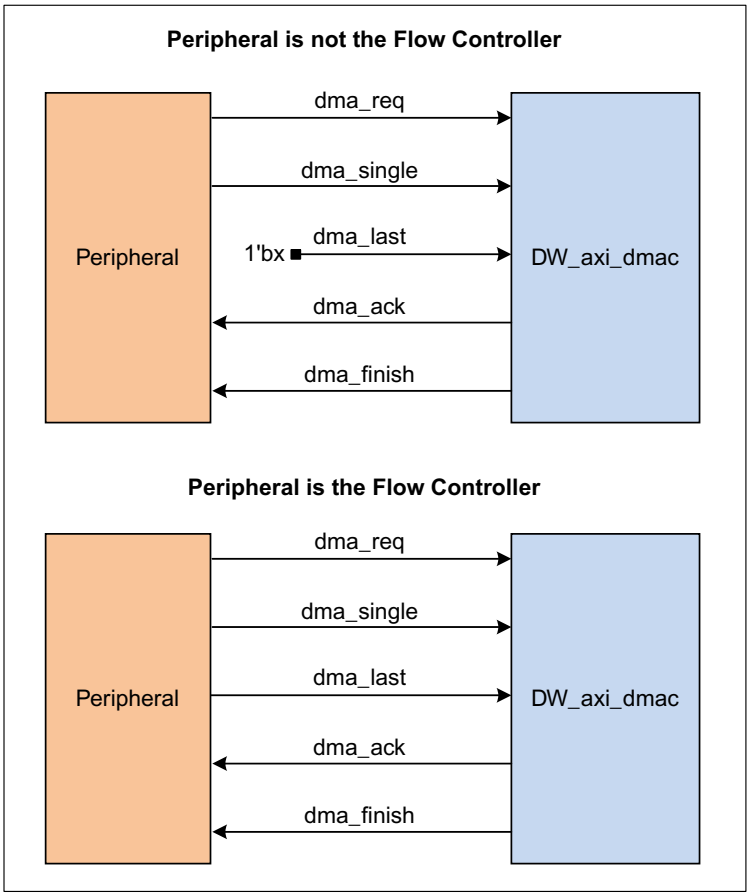
- **dma_req** – Burst transaction request input from peripheral. The functionality of this signal depends on whether the peripheral is the flow controller or not.
  - **If peripheral is not flow controller** – The DW_axi_dmac always interprets the `dma_req` signal as a burst transaction request, regardless 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.
  - I**f 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 2-23 on page 56 shows the hardware handshaking interface between a destination or source peripheral and DW_axi_dmac.

Figure 2-23 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 the 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 on page 57 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	X	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.
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 initiate a transaction. DW_axi_dmac waits until `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.

2.8.1.1 Hardware Handshaking Transaction Examples

This section provides various examples of hardware handshaking scenarios:

Burst Transaction – DMA Flow Controller

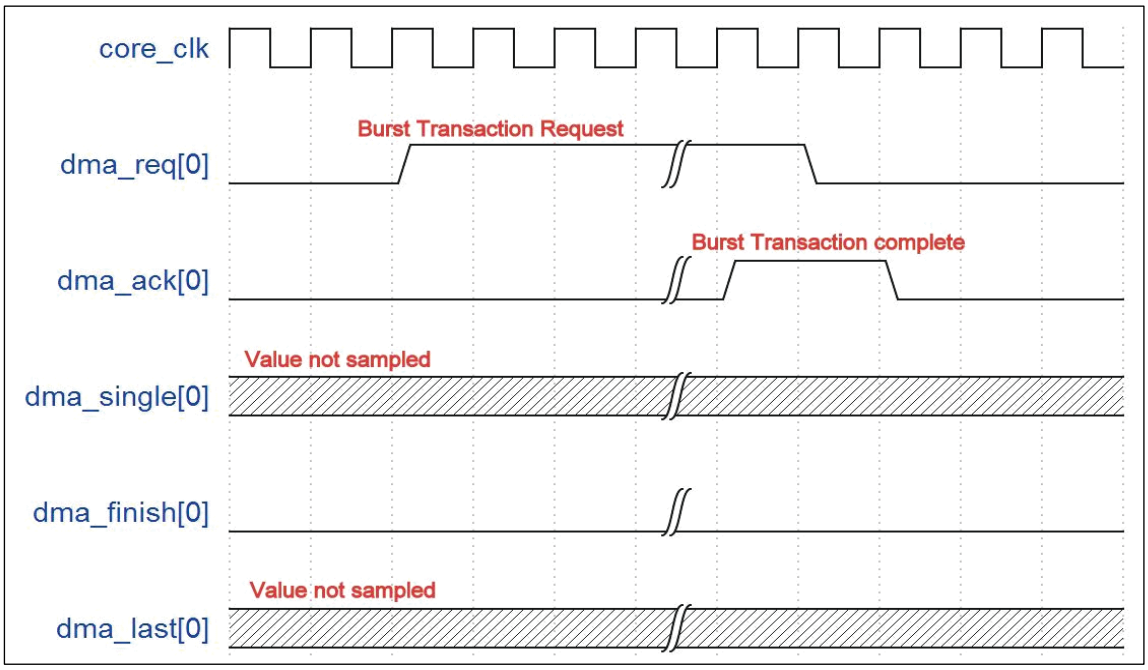
Figure 2-24 shows the timing diagram of a burst transaction. Because the peripheral is outside the Single Transaction Region, DW_axi_dmac does not sample dma_single[0].

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

Note After the DW_axi_dmac controller asserts the `dma_ack` signal, the peripheral may de-assert the `dma_req` signal after one or more clock cycles.

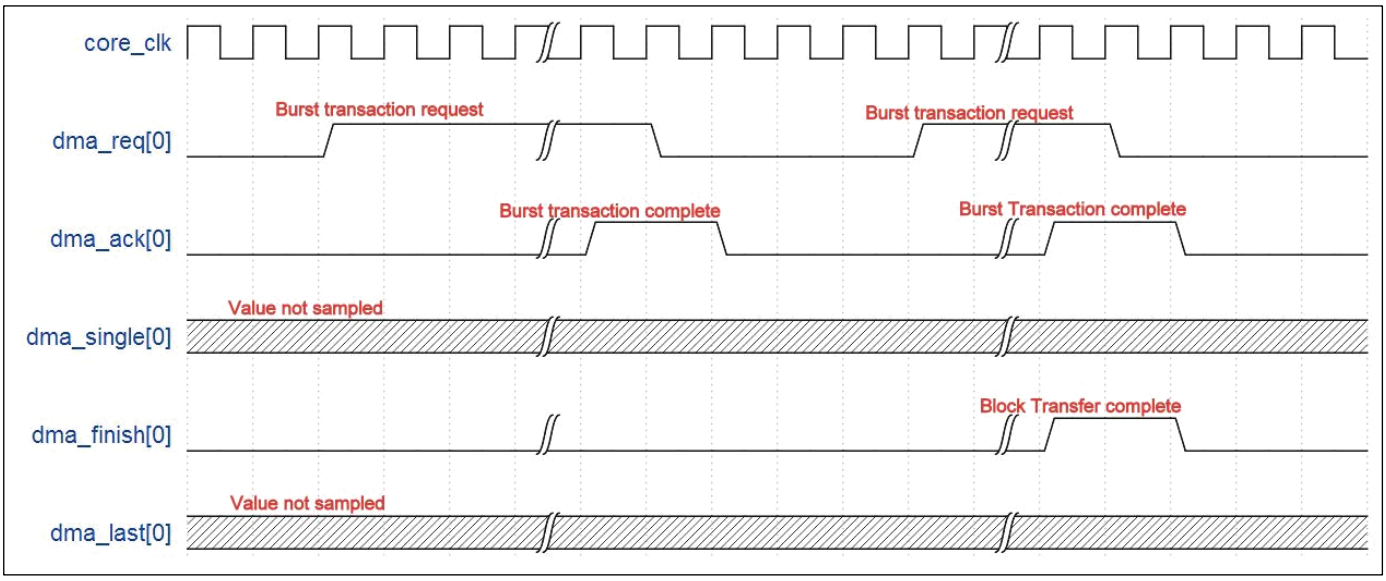
Figure 2-24 Burst Transaction



Back-to-Back Burst Transaction – DMA Flow Controller

Figure 2-25 on page 59 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 2-25 Back-to-Back Burst Transaction



Note

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

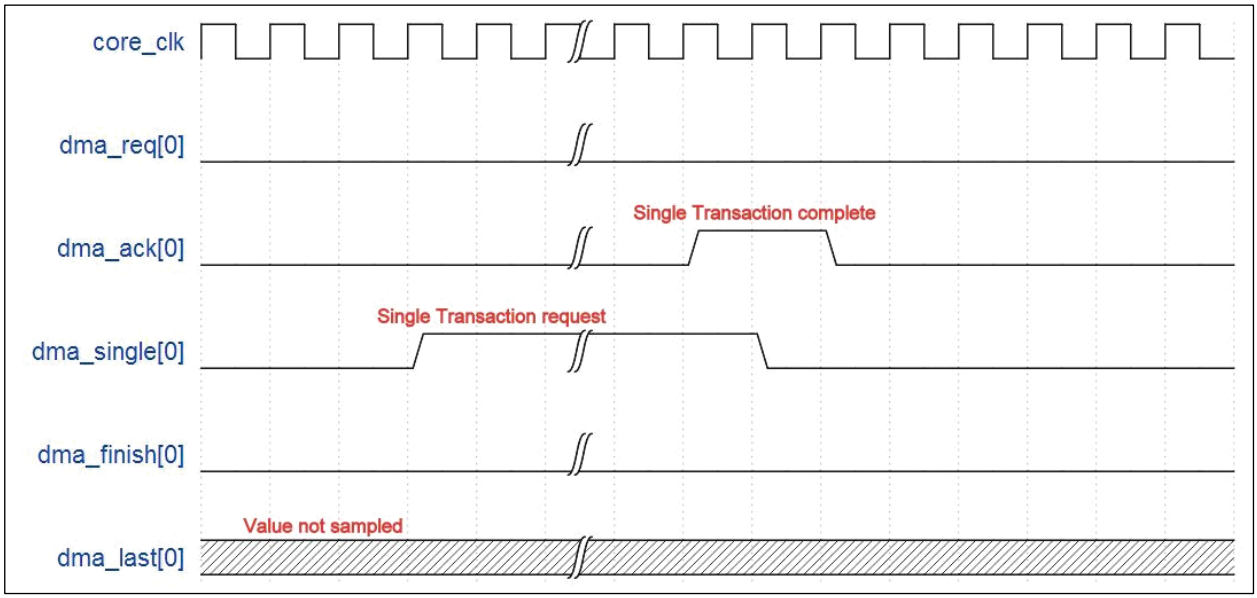
- Once asserted, the `dma_req` burst request signal must remain asserted until the corresponding `dma_ack` signal is received, even if the condition that generates `dma_req` in the peripheral is False.
- The `dma_req` signal must be de-asserted when `dma_ack` is asserted, even if the condition that generates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

Single Transaction – DMA Flow Controller

Figure 2-26 on page 60 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

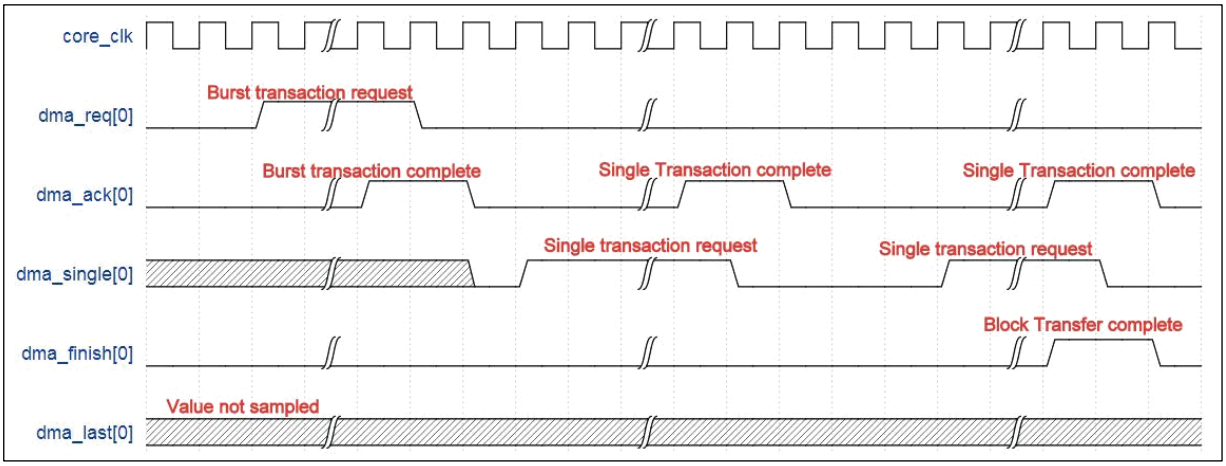
Figure 2-26 Single Transaction



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 2-27 illustrates this scenario.

Figure 2-27 Burst Followed by Back-to-Back Single Transaction

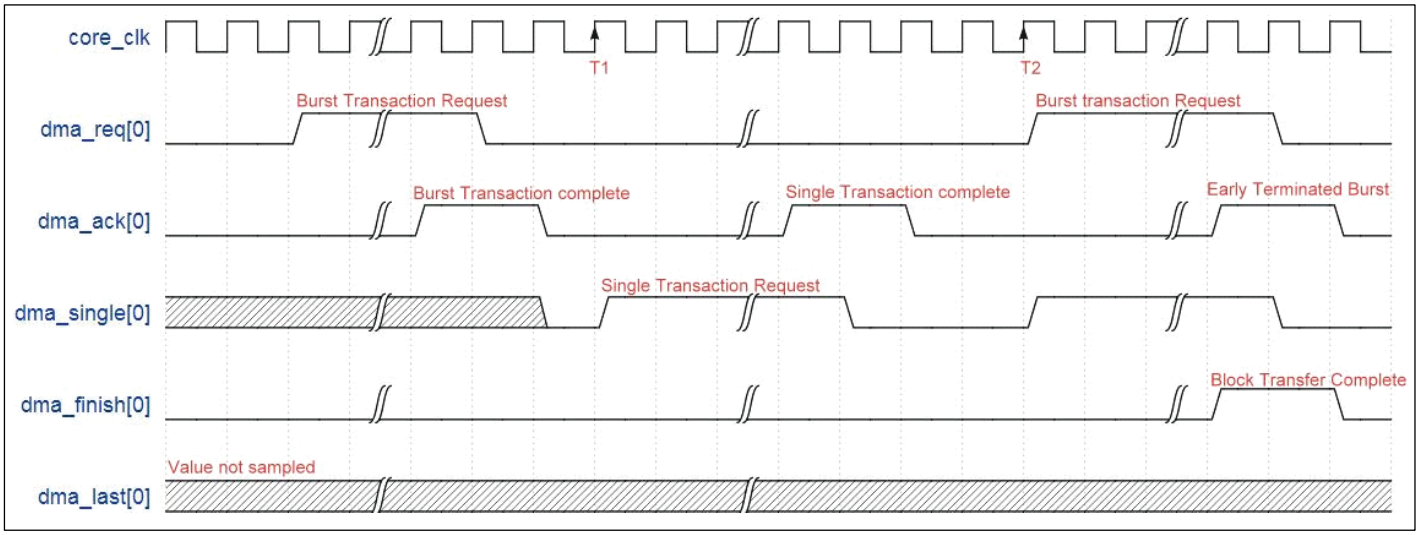


Early Terminated Burst Transaction – DMA Flow Controller

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 2-28, 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]`.

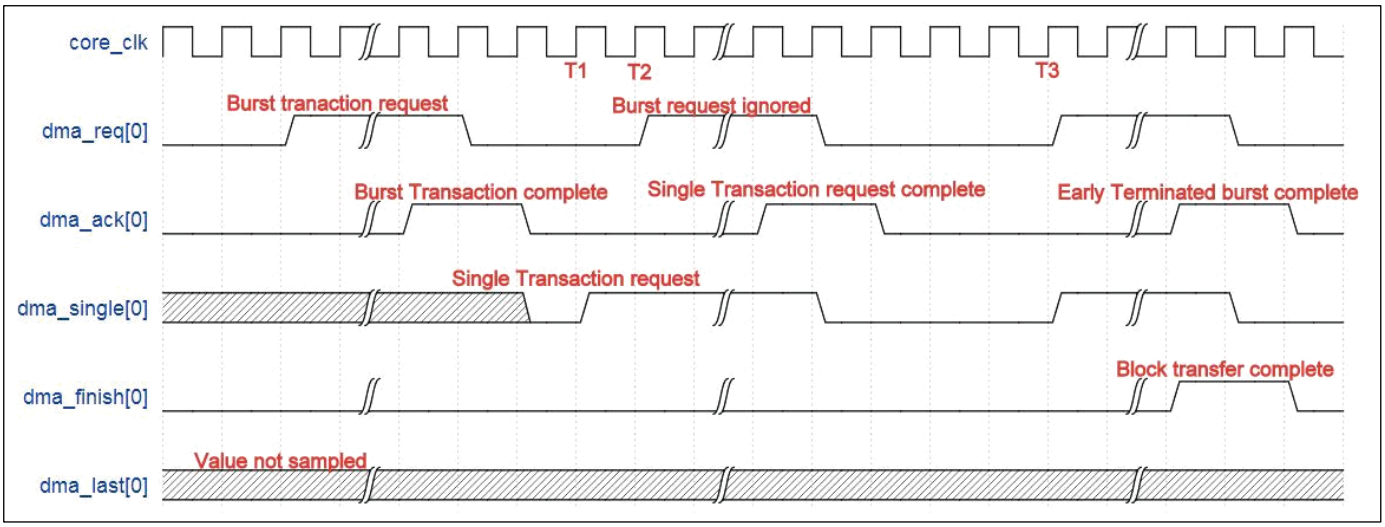
Figure 2-28 Early Terminated Burst Transfer



Burst Transaction Ignored During Active Single Transaction – DMA Flow Controller

As illustrated in Figure 2-29, 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 2-29 Burst Transaction Ignored During Active Single Transaction



Burst Transaction Followed by Single Transaction – Peripheral Flow Controller

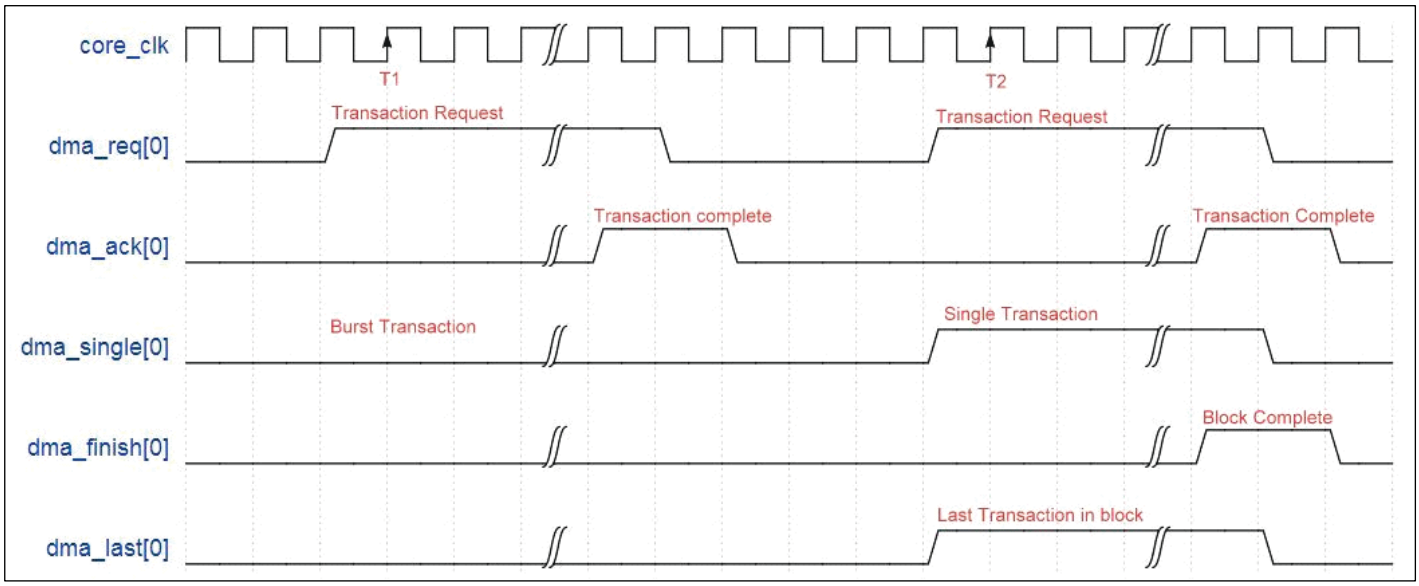
Figure 2-30 shows a burst transaction followed by a single transaction, where the single transaction is the last in the block. On clock edge T1, DW_axi_dmac samples that `dma_req[0]` is asserted, and `dma_single[0]` and `dma_last[0]` are de-asserted. This is a request for a burst transaction, which is 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

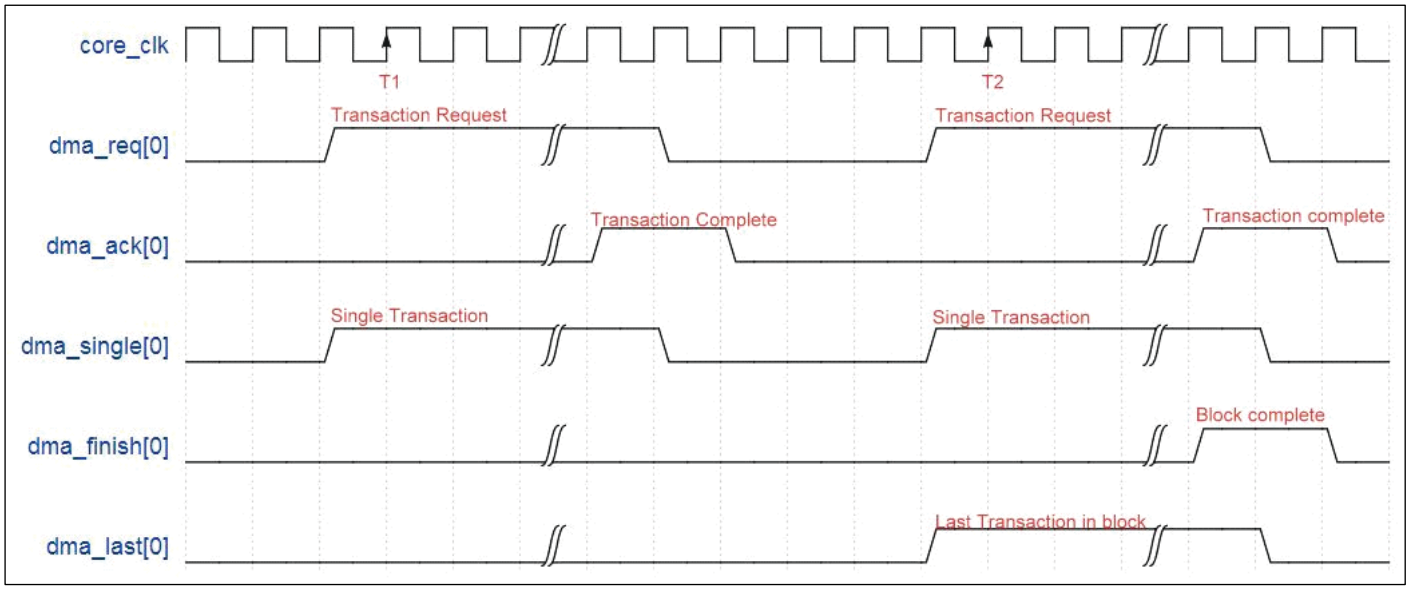
Figure 2-30 Burst Followed by Single Transaction (last in block)



Back-to-Back Single Transaction – Peripheral Flow Controller

Figure 2-31 on page 63 shows a back-to-back single transaction, where the last single transaction is the last transaction in the block.

Figure 2-31 Back-to-Back Single Transaction – Peripheral Flow Controller



2.8.1.2 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 subordinate 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 controller.

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.

2.8.1.3 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 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` - 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 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 54.

2.8.2 Software Handshaking

When the subordinate 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` and `HS_SEL_DST` bits 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.

Note 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.

2.9 Interrupt Interface

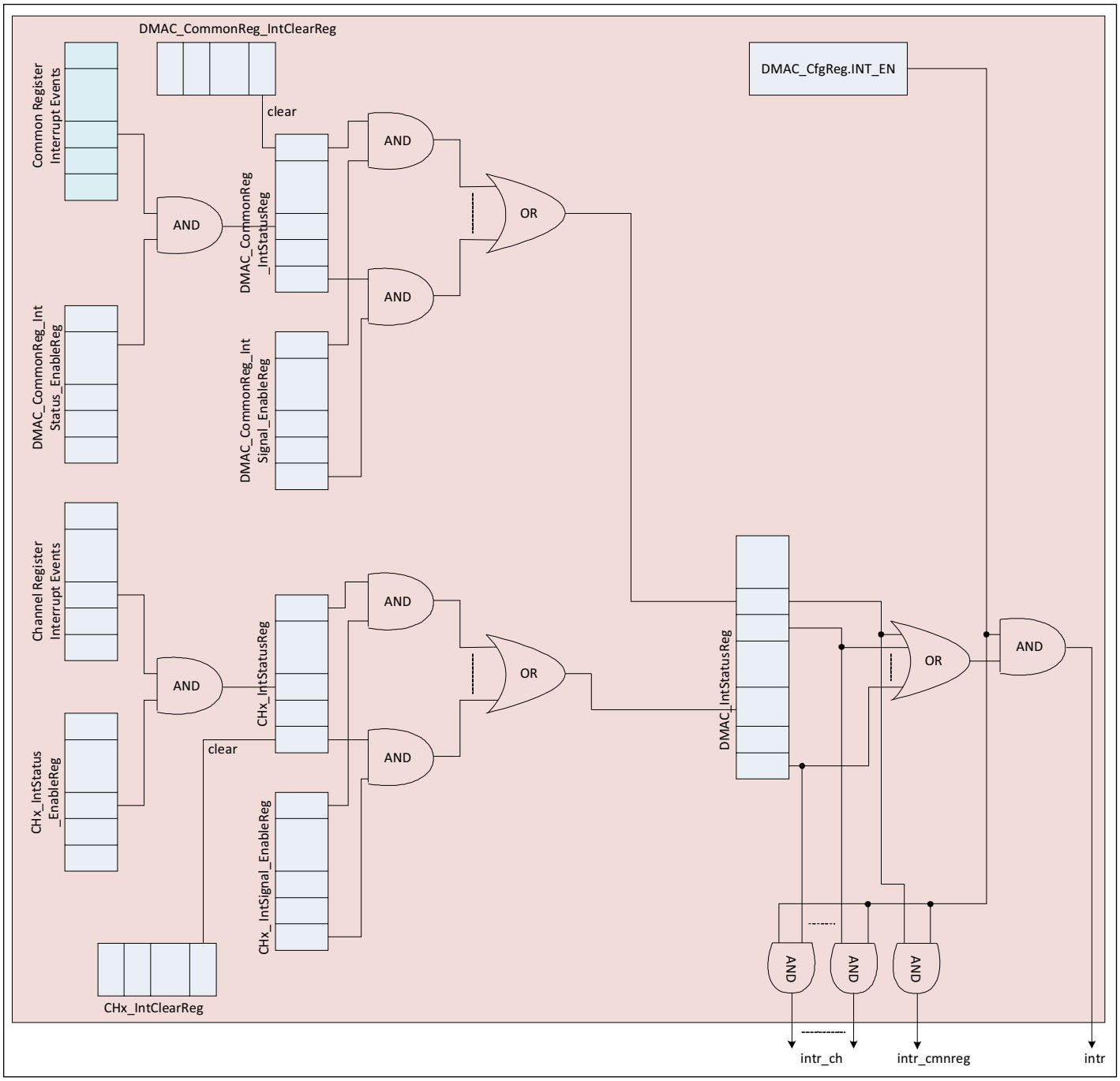
DW_axi_dmac supports combined and individual interrupt outputs configurable in coreConsultant. The polarity of the interrupt (Active High/Active Low) can be configured in coreConsultant. Interrupt outputs, by default, are synchronous to the core clock (`dmac_core_clock`), but they can be synchronized to the Register Bus 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 Register Bus 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 Register Bus Interface clock domain. The synchronization mechanism is not captured in Figure 2-32 on page 66.

Figure 2-32 on page 66 shows the interrupt generation mechanism in DW_axi_dmac.

Figure 2-32 DW_axi_dmac Interrupt Generation

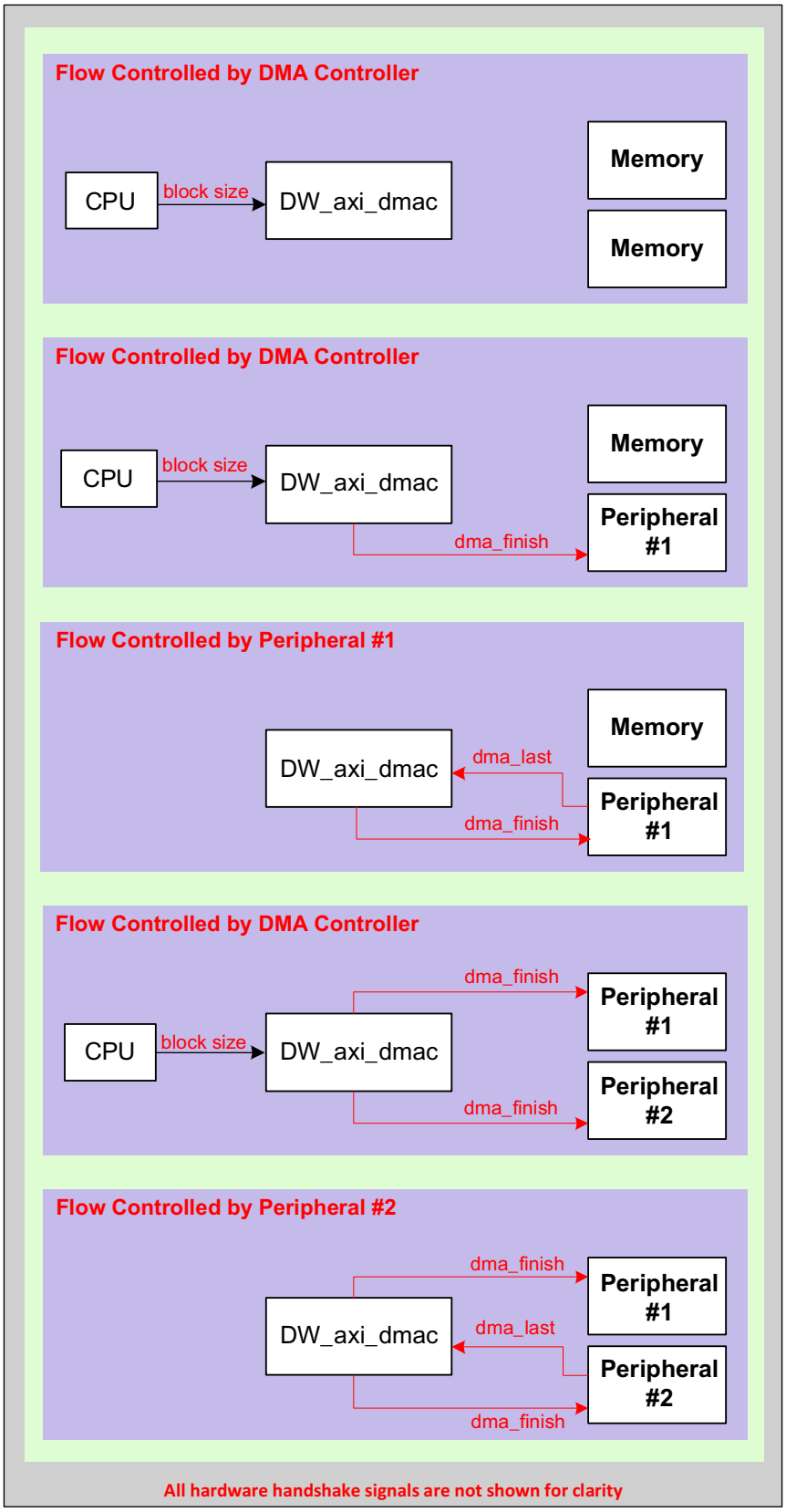


2.10 Flow Control Configurations

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 2-33 on page 68 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 2-33 DW_axi_dmac Flow Control Configuration



2.11 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`

2.12 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.

2.12.1 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.

2.12.2 Multi-block Transfer

If DW_axi_dmac is programmed for multi-block transfers, the `CHx_SAR` and `CHx_DAR` registers are reprogrammed using either of the following methods on successive blocks of a multi-block 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 multi-block 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 multi-block transfer is disabled and only single block transfers are allowed.
- If a multi-block transfer is enabled, there must be at least two blocks in the complete DMA transfer for Contiguous Address and Auto-Reloading-based multi-block transfers. Otherwise, it will result in unpredictable behavior.

The register update method for multi-block transfers depends on the value of the multi-block type bits in the `CHx_CFG` register. All possible combinations of the multi-block register update methods are captured in Table 2-4 on page 72.

Table 2-4 Register Update Methods for Multi-block Transfer

DMAX_CHx_MULTI_BLK_EN	Multi-block Type (Bits in CHx_CFG Register)		Register Update Method				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 multi-block
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	-

DMAX_CHx_MULTI_BLK_EN	Multi-block Type (Bits in CHx_CFG Register)		Register Update Method				MLTBLK_TFR_TYPE	Remarks
	SRC_MLTBLK_TYPE	DST_MLTBLK_TYPE	CHx_SAR	CHx_DAR	CHx_CTL	CHx_BLOCK_TS		
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	-
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

2.12.2.1 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 multi-block transfer is enabled, there must be at least two blocks in the complete DMA transfer. Otherwise, it will result in unpredictable behavior.

2.12.2.2 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 two blocks in the complete DMA transfer. Otherwise, it will result in unpredictable behavior.

2.12.2.3 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 multi-block 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 multi-block transfer used. If a shadow-register-based multi-block 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 multi-block 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" in the user guide.

2.12.2.4 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.

Note 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 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.
- The `LLI.CHx_CTL.ShadowReg_Or_LLI_Valid` bit will be 0 after the LLI write-back operation.

Figure 2-34 DW_axi_dmac Linked List Item (Descriptor)

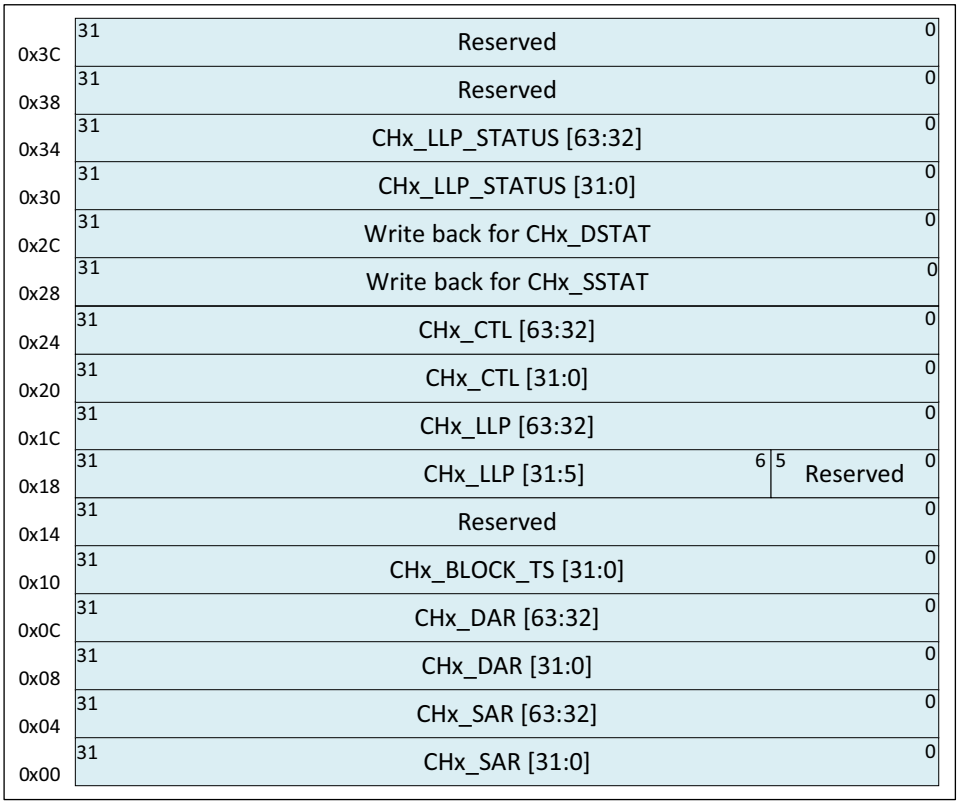


Figure 2-35 CHx_LLP_STATUS Write-Back Field of LLI

63 – 62	61 – 47	46 – 32	31 – 22	21 – 0
Status Indication CHx_IntStatus[1:0]	Reserved	Data left in Channel FIFO (CHx_Status[46:32])	Reserved	Completed Block Transfer Size (CHx_Status[21:0])

For more information about programming linked-list-based multi-block transfers, see “Programming Flow for Linked-List-Based Multi-Bock Transfer” in the user guide.

2.12.2.5 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 multi-block transfers (neither source nor destination peripheral uses Shadow Register or Linked-List-based multi-block 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.

2.12.2.6 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 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 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.

2.13 External Memory Interface

2.13.1 Overview of External Memory Interface

Each channel has dedicated FIFO for data storage. The dimensions of the FIFO memory are configurable with width supported up to 512 bits, and depth supported up to 256. The DW_axi_dmac channel FIFO memory uses flip-flop-based memory. If flip-flop-based memory is used when memory dimension is large, it is not area and power efficient.

External Memory interface optimizes the DMAC channel FIFO memory with external memories such as SRAM, Register File and so on for area and power optimization.

2.13.2 Description of External Memory Interface

The QoR of the DW_axi_dmac can be improved significantly by implementing channel FIFO memory with SRAM, or Register File based memories for larger memory dimensions. This feature provides separate external memory interface per channel to connect SRAM or Register File based memories to Channel FIFO. External Memory Interface also allows you to connect flip-flop-based memory through this external memory interface. This is beneficial when Channel FIFO memory dimensions are small, and flip-flop-based memories are more QoR efficient than SRAM or register file based memories.

The DMA channel reads the data from source and stores the data in channel FIFO temporarily. Then, the DMA channel reads data out of the FIFO memory and writes to destination. This read and write operation can work in parallel. Two-port Register file, SRAM, or flip-flop memory can be used based on the QoR requirement.

Generally, Register File or SRAM uses synchronous read timing, and read data is valid in the next cycle after read enable is active. Different FIFO memory types can be used as mentioned earlier. For example, flip-flop-based memory can be used for smaller memory dimensions; and Register File or SRAM can be used for larger memory dimensions. However, no matter which type is adopted, synchronous read timing must be used.

The `DMAX_CH_MEM_EXT` parameter is added to select internal channel FIFO memory interface scheme or external channel FIFO memory interface scheme.

This parameter also supports the top level value that is the channel FIFO memory is instantiated on the top level of the DW_axi_dmac.

For timing consideration, the `DMAX_CH_MEM_REGOUT` parameter is added which is used by all channels. When the `DMAX_CH_MEM_REGOUT` parameter is set to 1, a pipeline register is added internally for memory read data before sampled by internal FIFO control logic. In this case, read data has two clock cycles latency.

Figure 2-36 External Memory Interface – Read Timing Diagram

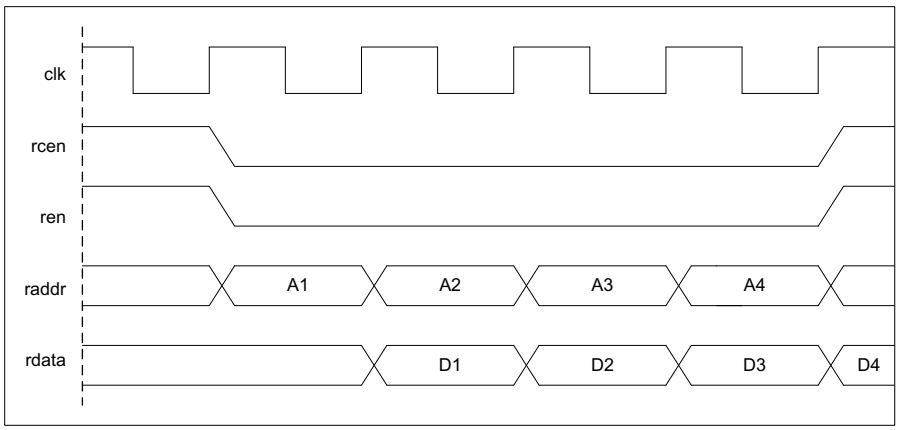
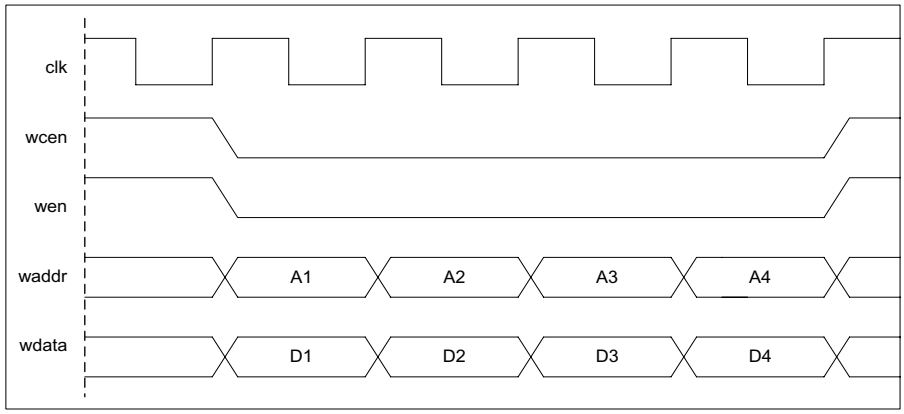


Figure 2-37 External Memory Interface – Write Timing Diagram



Note If the Read Unique ID feature is enabled, then this feature is also used for the UID Memory or Reordering Buffer.

2.13.3 Enabling External Memory Interface

To enable the External Memory Interface for DW_axi_dmac, choose the "External" option for the "Use external memory?" drop-down using the **Top Level Parameters** tab during the **Specify Configuration** Activity in coreConsultant.

2.13.4 Signals Related to External Memory Interface

The following are the signals related to External Memory Interface:

- `dmac_chx_fifomem_clk`
- `dmac_chx_fifomem_wcen`
- `dmac_chx_fifomem_wen`
- `dmac_chx_fifomem_waddr`

- `dmac_chx_fifomem_wdata`
- `dmac_chx_fifomem_rcen`
- `dmac_chx_fifomem_ren`
- `dmac_chx_fifomem_raddr`
- `dmac_chx_fifomem_rdata`

For more information about these signals, see the “Signal Descriptions” chapter.

2.14 Unique ID Feature

2.14.1 Overview of Unique ID Feature

Without AXI Unique ID support, the DW_axi_dmac core generates AXI read/write transfers with the same ID for a channel. The same ID generation ensures that the data is seen in order with respect to their address requests.

While this implementation satisfies multiple applications, there are certain scenarios where it becomes necessary to generate AXI Unique ID for each AXI read/write transfer. This feature is required because different banks/part of the source/destination memory responds with different latency. The overall performance can be improved, only if the DW_axi_dmac supports accepting out of order response, which can be achieved through the AXI Unique ID feature.

AXI Unique ID support requires programming only the `CHx_CFG` register to choose the number of AXI Unique IDs for AXI Read and Write Channel. Thus, Unique ID generation requires only a minimal software overhead, and thereby supports out-of-order read data or write response handling.

2.14.2 Description of Unique ID Feature

The AXI Unique ID feature allows DW_axi_dmac AXI Manager Interface to generate AXI read transfer on the AXI Read Channel with unique ID. The `DMAX_CH(x)_RD_UID` parameter and the RD_UID field of the `CHx_CFG` register settings control the number of unique IDs generated. This feature allows DW_axi_dmac to accept out-of-order read data from the source memory/peripheral. Then, DW_axi_dmac performs the re-ordering of the data to match with the issued address before the data is moved to the destination memory.

The AXI Unique ID feature allows DW_axi_dmac AXI Manager Interface to generate AXI write transfer with unique ID on the AXI Write Channel. The `DMAX_CH(x)_WR_UID` parameter and the `CHx_CFG.WR_UID` register field settings control the number of unique IDs generated. This feature allows DW_axi_dmac to accept out-of-order write response from the destination memory/peripheral.

AXI Unique ID feature can be enabled through the configuration parameter `DMAX_CH(x)_RD_UID_EN` for AXI Read Channel; and `DMAX_CH(x)_WR_UID_EN` for AXI Write Channel. You can configure the number of supported AXI unique IDs through the following configuration parameters:

- `DMAX_CH(x)_RD_UID` - Number of Unique IDs supported for AXI Read Channel
- `DMAX_CH(x)_WR_UID` - Number of Unique IDs supported for AXI Write Channel

Note Unique ID feature cannot be enabled when Unaligned transfer is enabled.

The following sections describe the AXI Read and Write Channel ID generation in different scenarios.

2.14.2.1 AXI Read Channel ID Format

The AXI Read channel IDs include the following signals. where

- arid_mi signal is used as the identification tag for the Read Address channel signals
- rid_mi signal is used as the identification tag for the Read Data channel signals

The format of the Read channel ID depends on the following:

- Source Data Transfer or LLI Fetch access
- AXI Unique ID supported or not

The upper M bits of the AXI Read Channel ID is derived based on:

- Channel number of the channel that is currently accessing the Manager Interface
- Current access is Source data transfer or LLI fetch

Example-1 When DMAX_NUM_CHANNELS = 32

- `Source Data Transfer – arid_mi/rid_mi:`
  `Channel 1 = 6’b00_0000`
  `Channel 2 = 6’b00_0001`
  … ……
  `Channel 32 = 6’b01_1111`
- `LLI Fetch Access – arid_mi/rid_mi:`
  `Channel 1 = 6’b10_0000`
  `Channel 2 = 6’b10_0001`
  … ……
  `Channel 32 = 6’b11_1111`

Example-2 When DMAX_NUM_CHANNELS = 8

- `Source Data Transfer – arid_mi/rid_mi:`
  `Channel 1 = 4’b0000`
  `Channel 2 = 4’b0001`
  … ……
  `Channel 8 = 4’b0111`
- `LLI Fetch Access – arid_mi/rid_mi:`
  `Channel 1 = 4’b1000`
  `Channel 2 = 4’b1001`
  … ……
  `Channel 8 = 4’b1111`

The lower N bits of the AXI Read Channel ID are derived based on whether the Unique ID feature is enabled or not.

- When Unique ID feature is not enabled,
  - The lower N bits is equal to the programmed value in the `AXI_Read_ID_Suffix` field of the `CHx_AXI_IDReg` register.
- When Unique ID feature is enabled,

- The lower N bits are used for the generation of the AXI Unique ID in the incrementing fashion. The number of unique ID generated is limited by the `DMAX_CH(x)_RD_UID` parameter, and the `RD_UID` field of the `CHx_CFG` register. The remaining bits out of N bits are driven to 0s.

Example-3 When DMAX_NUM_CHANNELS=32, DMAX_AXI_ID_SUFFIX_WIDTH=6, DMAX_CH(x)_RD_UID=4, and CHx_CFG.RD_UID=4.

The lower N bits of the `arid_mi/rid_mi` signals are generated in the following order:

1. 6'b00_0000
2. 6'b00_0001
3. 6'b00_0010
4. 6'b00_0011

Where,

```
M = DMAX_M_ID_WIDTH - DMAX_AXI_ID_SUFFIX_WIDTH*
N = DMAX_AXI_ID_SUFFIX_WIDTH*
```

2.14.2.2 AXI Write Channel ID Format

The AXI Write channel IDs include `awid_mi`, `wid_mi`, and `bid_mi` signals, where

- `awid_mi` signal is used as the identification tag for the Write Address channel signals
- `wid_mi` signal is used as the identification tag for the Write Data channel signals
- `bid_mi` signal is used as the identification tag for the Write Response channel signals

The format of the Write channel ID varies depending on the following:

- Destination Data Transfer or LLI Write Back access
- AXI Unique ID is supported or not

The upper M bits of the AXI Write Channel ID is derived based on:

- The channel number of the channel that is currently accessing the Manager Interface and
- The current access is Destination data transfer or LLI Write back

Example:1 When DMAX_NUM_CHANNELS = 32.

- `Destination Data Transfer – awid_mi/wid_mi/bid_mi:`

```
Channel 1 = 6'b00_0000
Channel 2 = 6'b00_0001
…     ……
Channel 32 = 6'b01_1111
```

- `LLI Write back Access – awid_mi/wid_mi/bid_mi:`
  `Channel 1 = 6’b10_0000`
  `Channel 2 = 6’b10_0001`
  … ……
  `Channel 32 = 6’b11_1111`

Example:2 When DMAX_NUM_CHANNELS = 8

- `Destination Data Transfer – awid_mi/wid_mi/bid_mi:`
  `Channel 1 = 4’b0000`
  `Channel 2 = 4’b0001`
  … ……
  `Channel 8 = 4’b0111`
- `LLI Write back Access – awid_mi/wid_mi/bid_mi:`
  `Channel 1 = 4’b1000`
  `Channel 2 = 4’b1001`
  … ……
  `Channel 8 = 4’b1111`

The lower N bits are derived based on whether the Unique ID feature is enabled or not.

- When Unique ID feature is not enabled:
  - The lower N bits are equal to the programmed value in the `CHx_AXI_IDReg` field of the `AXI_Write_ID_Suffix` register.
- When Unique ID feature is enabled:
  - The lower N bits are used for the generation of the AXI Unique ID in the incremental fashion. The `DMAX_CH(x)_WR_UID` parameter and the WR_UID field of the `CHx_CFG` register limits the number of Unique IDs generated. The remaining bits out of N bits are driven to 0s.

Example:3 When DMAX_NUM_CHANNELS=32, DMAX_AXI_ID_SUFFIX_WIDTH=6, DMAX_CH(x)_WR_UID=4, and CHx_CFG.WR_UID=4.

The lower N bits of the `awid_mi/wid_mi/bid_mi` signals are generated in the following order:

1. 6’b00_0000
2. 6’b00_0001
3. 6’b00_0010
4. 6’b00_0011

Where,

`M = DMAX_M_ID_WIDTH - DMAX_AXI_ID_SUFFIX_WIDTH*`

`N = DMAX_AXI_ID_SUFFIX_WIDTH*`

2.14.2.3 Additional Considerations

Following is the equation to limit the maximum number of Read unique IDs (`CHx_CFG.RD_UID`) that can be programmed:

Number of Beats per Reordering Buffer Location

```
= N(beats_per_loc)
= Reordering buffer width/Source transfer width
= DMAX_CHx_FIFO_WIDTH/CHx_CTL.SRC_TR_WIDTH
```

Number of Locations per ARLEN

```
= N(locs_per_arlen)
= AXI read transfer length/Number of beats per reordering buffer Location
= Ceil[CHx_CTL.ARLEN/N(beats_per_loc)]
```

Maximum number of Read UID's or Maximum limit on `CHx_CFG.RD_UID`

```
= Re-ordering Buffer depth/No of locations per ARLEN
= Floor[DMAX_CHx_REORDER_BUFF_DEPTH/N(locs_per_arlen)]
```

Thus reordering buffer depth, and the burst length limits the generation of maximum number of Read Unique IDs.

2.14.3 Enabling the Unique ID Feature

- To enable Unique ID feature for AXI read address, select true in the Include AXI Unique ID Feature in AXI Read Channel? drop-down in the Channel Configuration tab during the Specify Configuration Activity in coreConsultant.
- To enable Unique ID feature for AXI write address, select true in the Include AXI Unique ID Feature in AXI Write Channel? drop-down in the Channel Configuration tab during the Specify Configuration Activity in coreConsultant.

2.14.4 Signals Related to the Unique ID Feature

The following signals are related to the Unique ID feature:

UID Memory Interface

- `dmac_chx_uidmem_clk`
- `dmac_chx_uidmem_wcen`
- `dmac_chx_uidmem_wen`
- `dmac_chx_uidmem_waddr`
- `dmac_chx_uidmem_wdata`
- `dmac_chx_uidmem_rcen`
- `dmac_chx_uidmem_ren`
- `dmac_chx_uidmem_raddr`
- `dmac_chx_uidmem_rdata`

UID Memory Interface – AXI Manager Interface Signals

- `arid_mi[(DMAX_M_ID_WIDTH-1):0]`

- `rid_mi[(DMAX_M_ID_WIDTH-1):0]`
- `awid_mi[(DMAX_M_ID_WIDTH-1):0]`
- `wid_mi[(DMAX_M_ID_WIDTH-1):0]`
- `bid_mi[(DMAX_M_ID_WIDTH-1):0]`

For more information about these signals, see the “Signal Descriptions” chapter.

2.15 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, Register Bus Interface, and AXI Manager 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, Register Bus interface, and AXI Manager interfaces through the following parameters:

- DMA Channels Context Sensitive Low Power Technique - `DMAX_CHNL_CSLP_EN`
- Register Bus Interface Context Sensitive Low Power Technique - `DMAX_SBIU_CSLP_EN`
- AXI Manager 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, Register Bus Interface, and AXI Manager Interface modules.

Note

- The Context Sensitive Low-Power feature adds a latch based clock gating cell (DW_axi_dmac_clkgate_cell), which is behavioral model of the clock gating cell.
- This clock gating cell behavioral model must be replaced with the technology specific latch based Integrated Clock Gating (ICG) Cell during the synthesis process.

2.15.1 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

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 Manager Interface, but it is waiting for grant from the AXI Manager Interface arbiter (because Other DMA Channels Source, Destination, and LLI state machines also competing for the AXI Manager 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).
- The DMA Channel is suspended based on the Channel Suspend procedures described in the section
- The DMA Channel is terminated based on the Abnormal Channel termination procedure described in the section)
- 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 Manager 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_CF-GREG.DMAC_EN`.
- On the rising or falling edge of the Channel suspend or Channel terminate.
- 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 `DMAX_GLCH_LPDLY` parameter is used to configure the default load value of the `DMAC_LOWPOWER_CFGREG.CHNL_LPDLY` field.

2.15.2 Register Bus Interface Context Sensitive Low Power Technique

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

The Context Sensitive Low Power option monitors the activity on the Register Bus Interface and based on based on the SBIU state machine. Whenever the Register Bus Interface low power condition is detected, (that is when no AHB/APB 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 Register 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 Register 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 Register 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 Register Bus Interface module enters the Low Power state:

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

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

- Any AHB/APB 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 `DMAX_SBIU_LPDLY` parameter is used to configure the default load value of the `DMAC_LOWPOWER_CFGREG.SBIU_LPDLY` field.

2.15.3 AXI Manager Interface Context Sensitive Low Power Technique

The AXI Manager 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 Manager interface to optimize the power consumption in the AXI Manager 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 `DMAX_MXIF_LPDLY` parameter is used to configure the default load value of the `DMAC_LOWPOWER_CFGREG.MXIF_LPDLY` field.

2.15.4 Global Context Sensitive Low Power Technique

The Global Context Sensitive Low Power technique implements the low power option based on the DMA Channel, Register Bus Interface, and AXI Manager 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 `DMAX_GLCH_LPDLY` parameter is used to configure the default load value of the `DMAC_LOWPOWER_CFGREG.GLBL_LPDLY` field.

2.16 AXI Unaligned Transfer Support

The DW_axi_dmac supports the generation of unaligned DMA transfers on the AXI Manager 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.

2.16.1 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 Manager 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, you 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.

2.16.1.1 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 following (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`

2.16.1.2 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 following (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`

2.16.1.3 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_d-st_u + Source Unaligned bytes
  - Destination Contiguous: Next CHx_DAR = Current CHx_DAR + blk_size_bytes_dst_u

2.16.2 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 Manager 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 91.

Conventions

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

	Invalid byte of the AXI DMA Transfer
N	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.

2.16.2.1 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 2-5 on page 95 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		
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 2-38 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 2-39 on page 96 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 2-38.

Figure 2-39 Example 1 - Unaligned Source Transfer Waveform

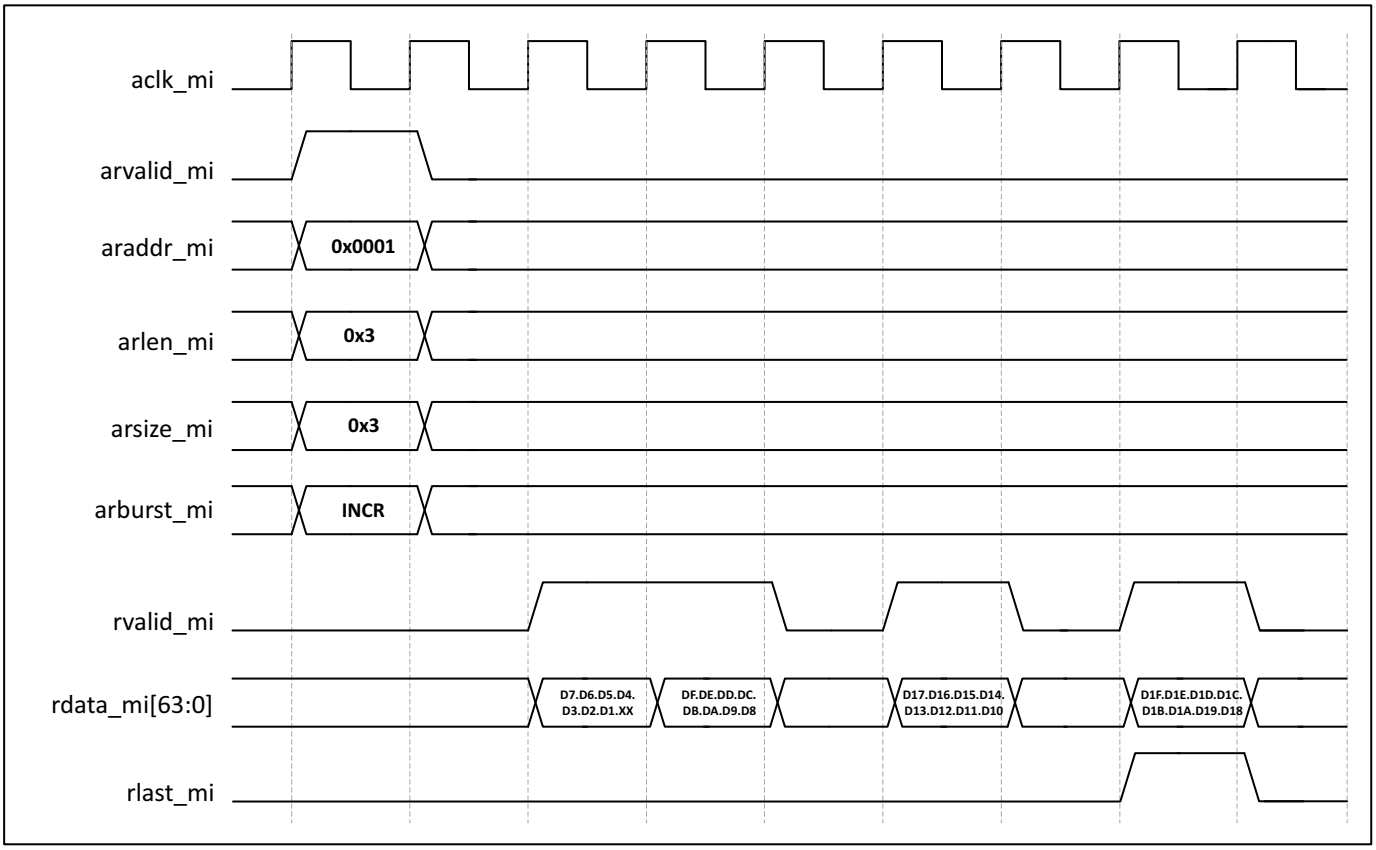
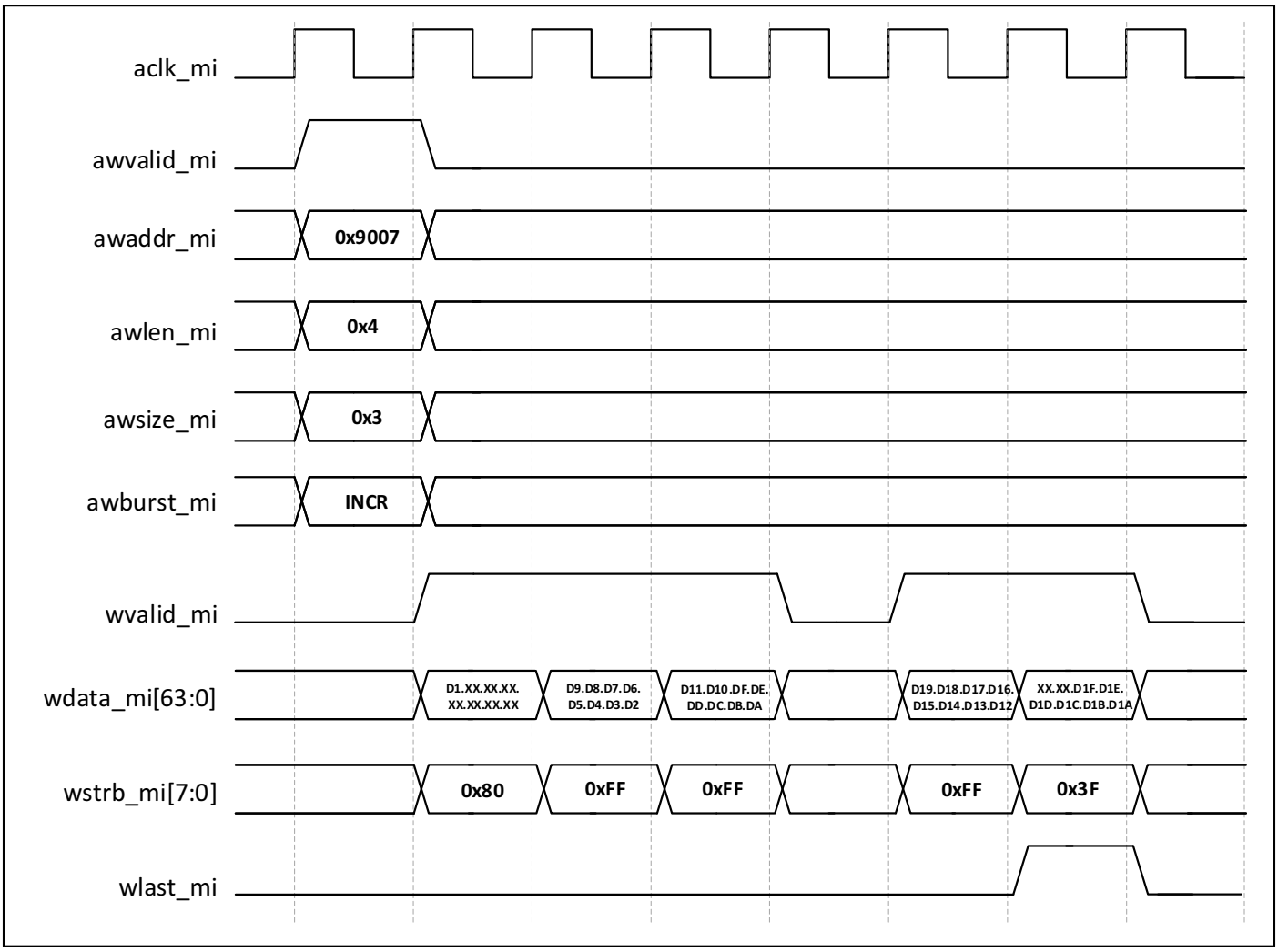


Figure 2-40 on page 97 shows the destination transfer where the destination address is unaligned - 0x9007. This unaligned destination transfer has five beats, with the AWSIZE of three that is eight bytes. In the first write data beat generated, only the most significant data byte is valid as the address is unaligned with seven 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 2-40 Example 1 - Unaligned Destination Transfer Waveform



2.16.2.2 Example 2

Assume that

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

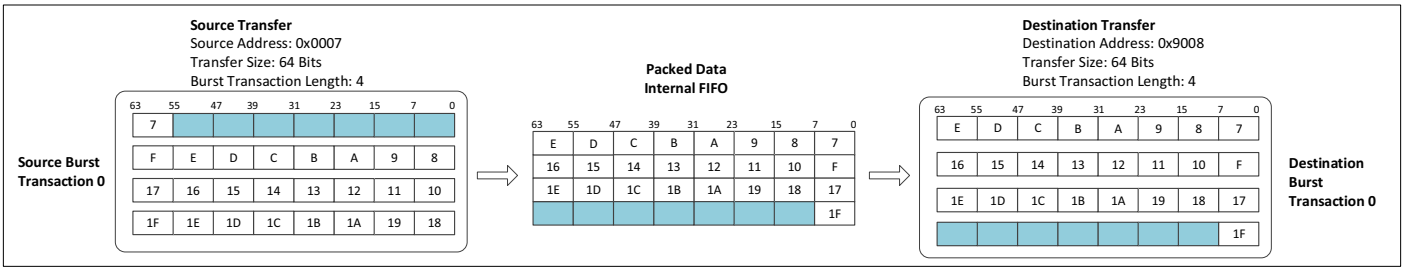
Table 2-6 describes all major DMA parameter settings considered for this example.

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

Configuration/Programming	Value	Description
Programming Register		
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 2-41 Example 2 - Unaligned DMA Transfer Handling



2.16.2.3 Example 3

Assume that,

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

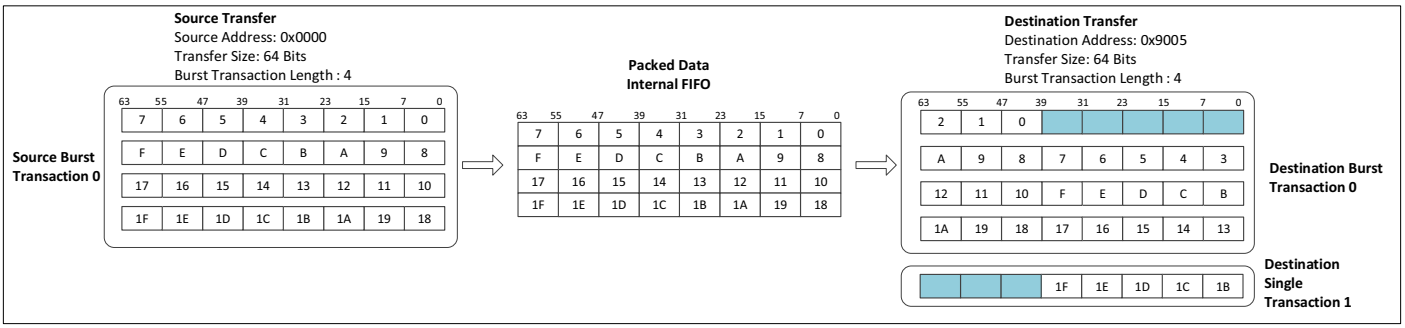
Table 2-7 describes all major DMA parameter settings considered for this example.

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

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 2-42 Example 3 - Unaligned DMA Transfer Handling



2.16.2.4 Example 4

Assume that,

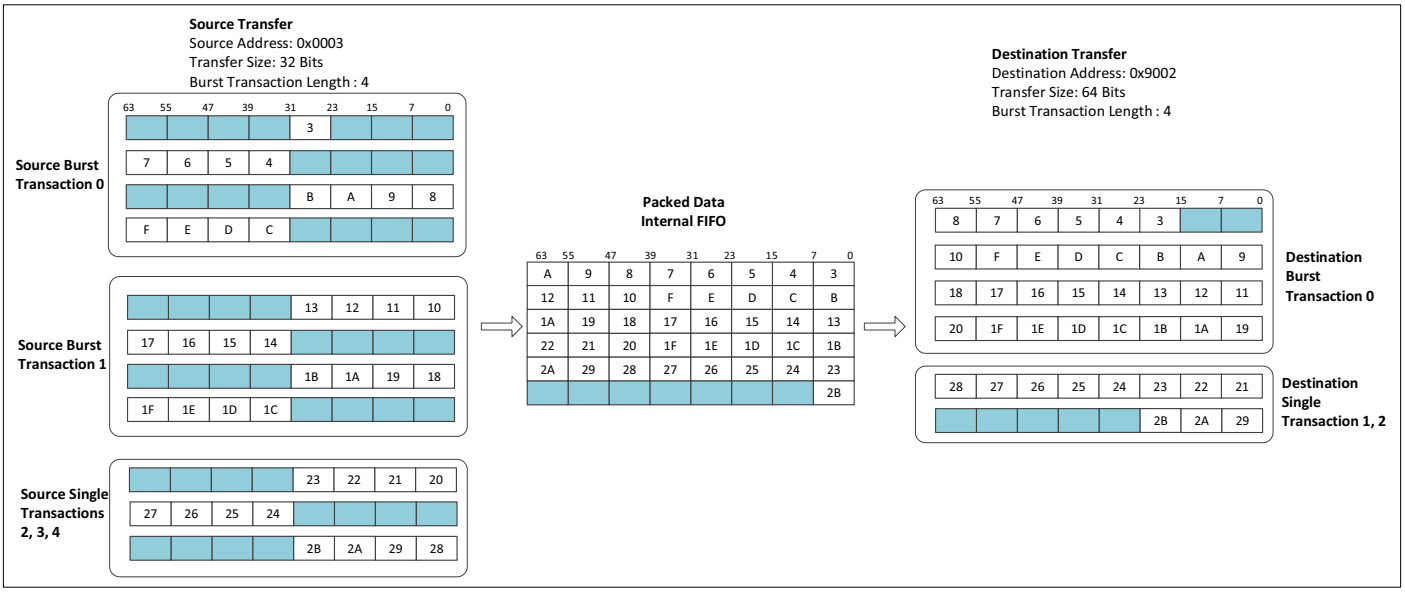
- the source address is unaligned to the source transfer width,
- the destination address is also unaligned to the destination transfer width
- the source transfer width is not the same as the Manager Interface data bus width with multiple source burst/single DMA transactions

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
CHx_CTL.DST_TR_WIDTH	0x3	Destination Transfer Width is 64 bits

Figure 2-43 Example 4 - Unaligned DMA Transfer Handling



2.16.2.5 Example 5

Assume that,

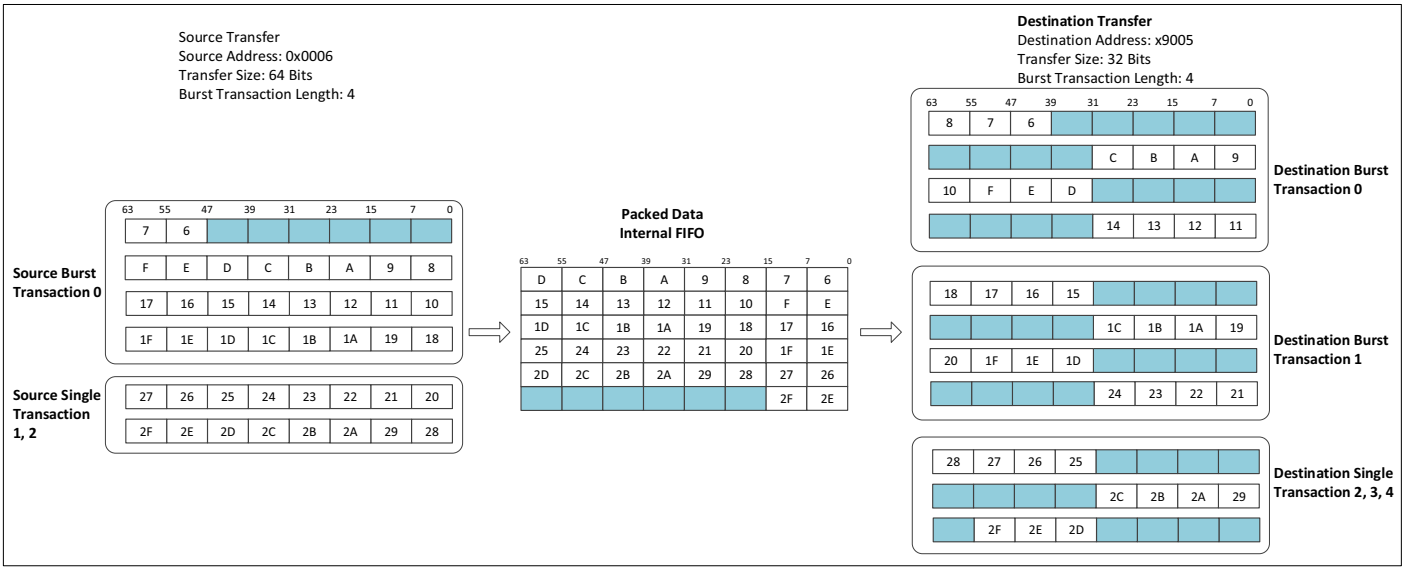
- the source address is unaligned to the source transfer width
- the destination address is also unaligned to the destination transfer width
- the destination transfer width is not the same as the Manager Interface data bus width with multiple destination burst/single DMA transactions

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
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 2-44 Example 5 - Unaligned DMA Transfer Handling



2.16.2.6 Example 6

Assume that,

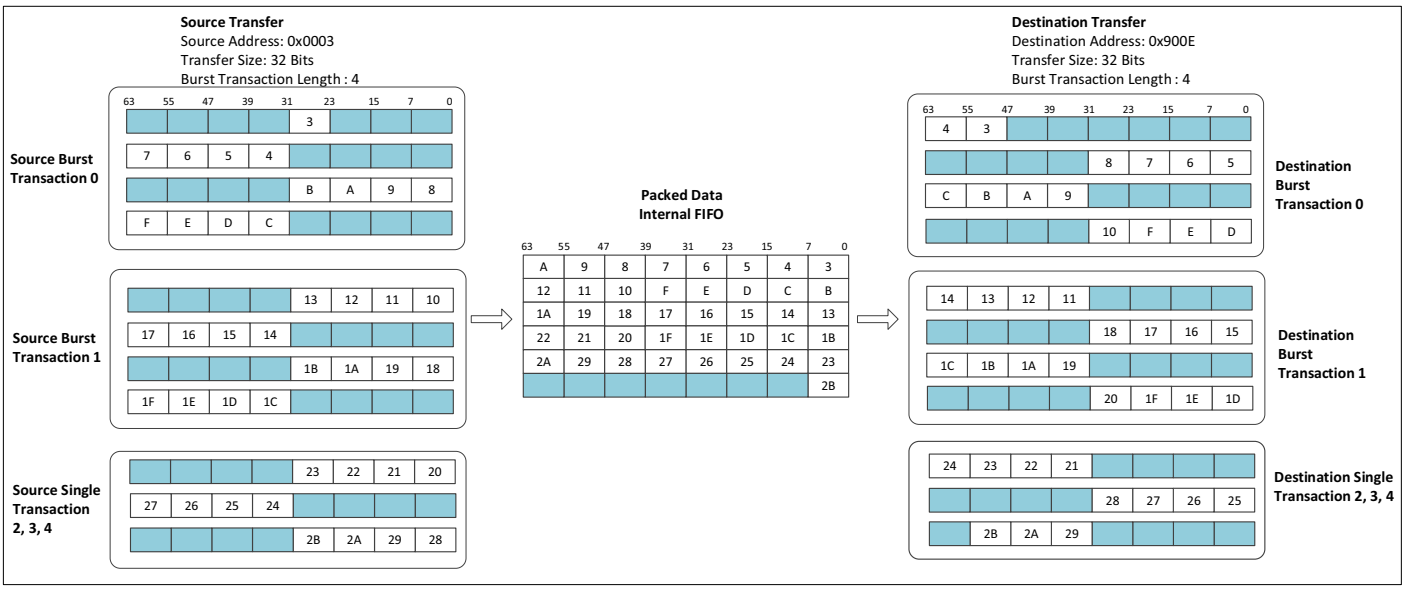
- the source address is unaligned to source transfer width
- the destination address is also unaligned to the destination transfer width
- both the source and the destination transfer widths are not the same as the Manager Interface data bus width with multiple source burst/single DMA transactions.

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
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 2-45 Example 6 - Unaligned DMA Transfer Handling



2.16.2.7 Example 7

Assume that,

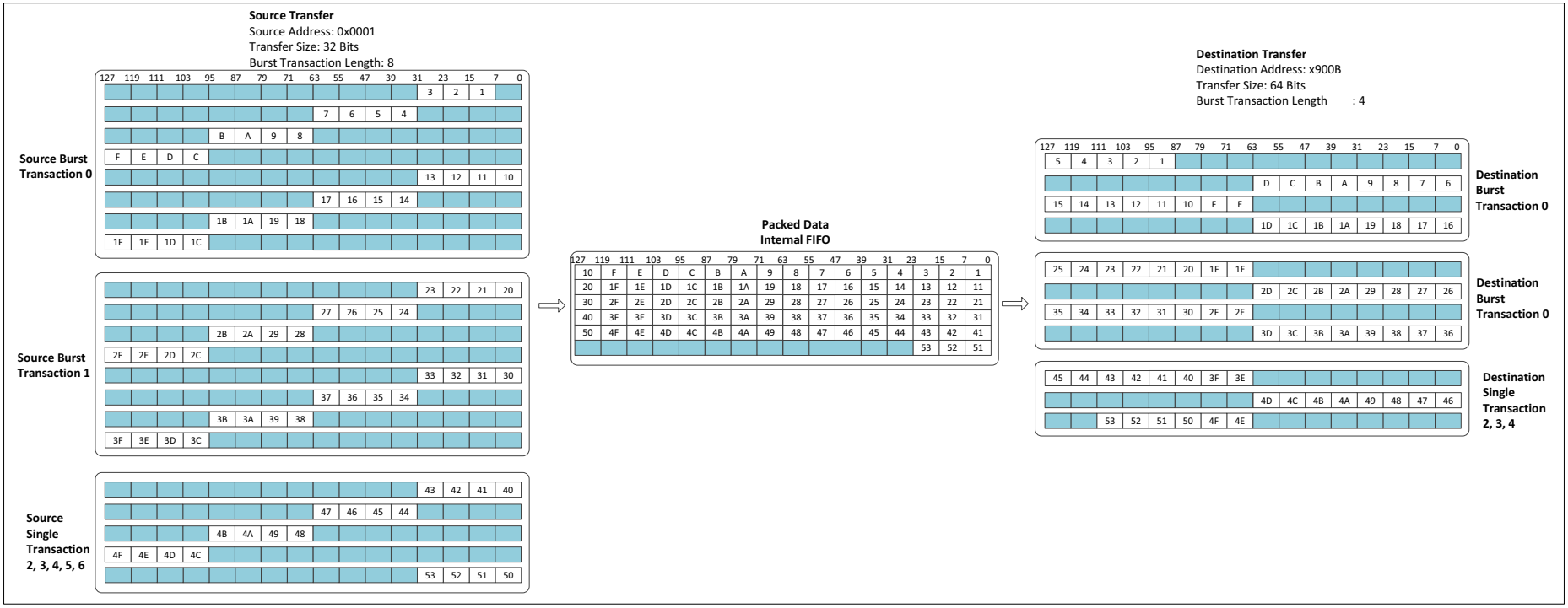
- the source address is unaligned to source transfer width
- the destination address is also unaligned to the destination transfer width
- both the source and the destination transfer widths are not the same as the Manager Interface data bus width with multiple source burst/single DMA transactions

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 2-46 Example 7 - Unaligned DMA Transfer Handling



2.16.2.8 Example 8

Assume that,

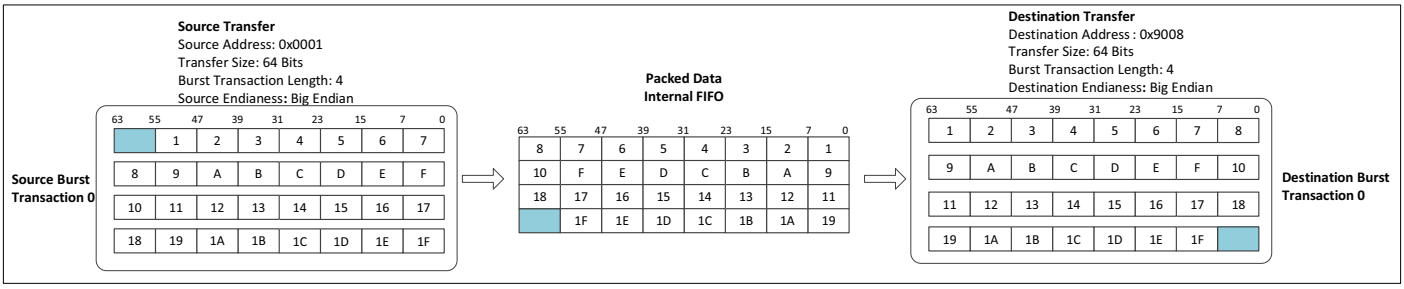
- the source address is unaligned to the source transfer width
- the destination address is aligned to destination transfer width
- the source and the destination endianness is big-endian

Table 2-12 on page 104 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 2-47 Example 8 - Unaligned DMA Transfer Handling



2.16.2.9 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

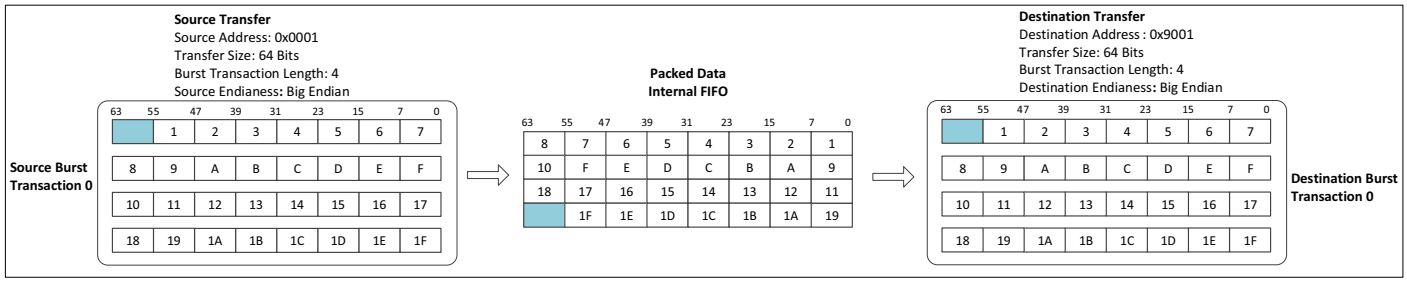
Table 2-13 describes all major DMA parameter settings considered for this example.

Table 2-13 Example 9 - Configuration and Programming Options

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

Configuration/Programming	Value	Description
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 2-48 Example 9 - Unaligned DMA Transfer Handling



2.16.2.10 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

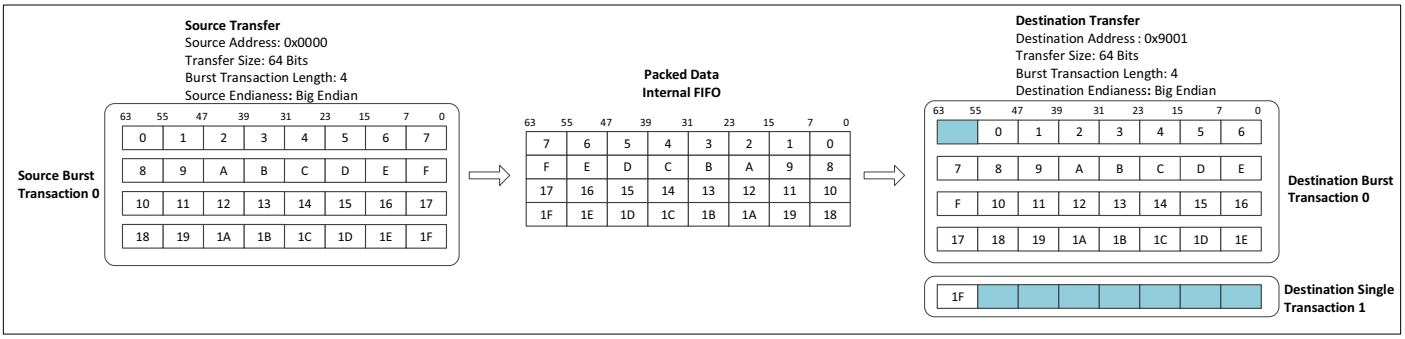
Table 2-14 describes all major DMA parameter settings considered for this example.

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
CHx_DAR	0x9001	Unaligned destination address
CHx_CTL.SRC_MSIZE	3b001	Source burst transaction length = 4

Configuration/Programming	Value	Description
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 2-49 Example 10 - Unaligned DMA Transfer Handling



2.16.2.11 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 Manager Interface data bus width with multiple source burst/single DMA transactions
- the source and destination endianness is big-endian

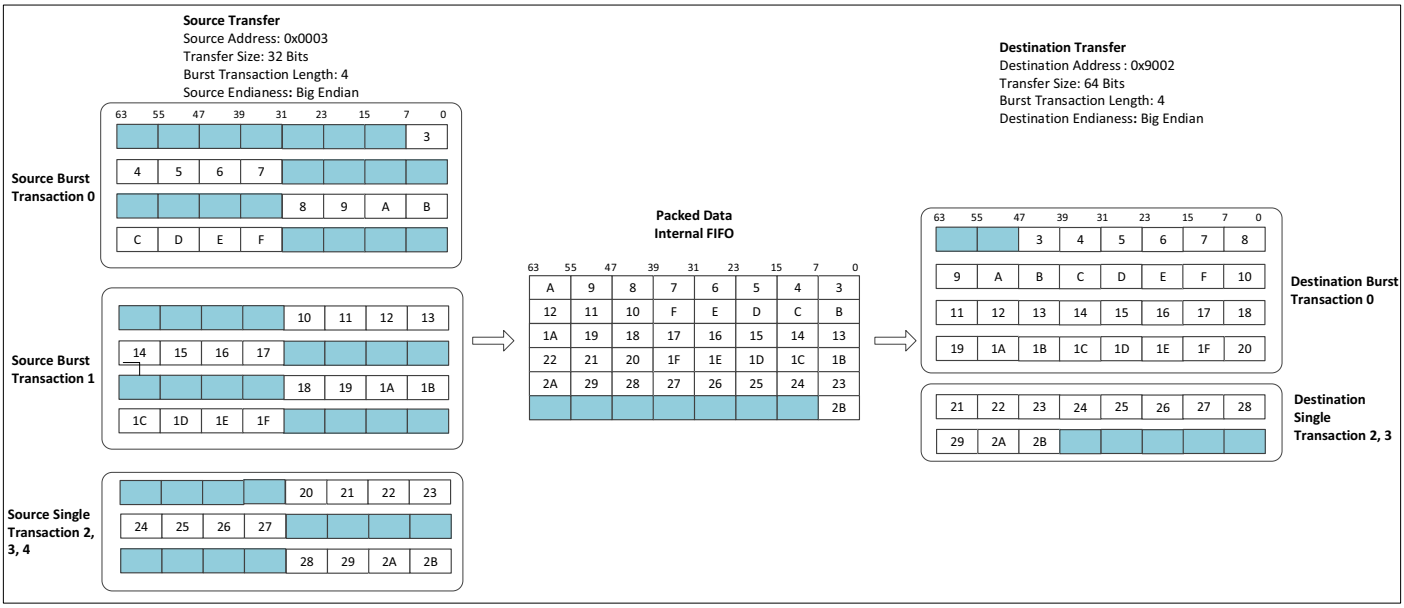
Table 2-15 describes all major DMA parameter settings considered for this example.

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
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

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

Figure 2-50 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 2-16 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)

2.16.2.12 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 Manager Interface data bus width
- the source and the destination endianness is big-endian

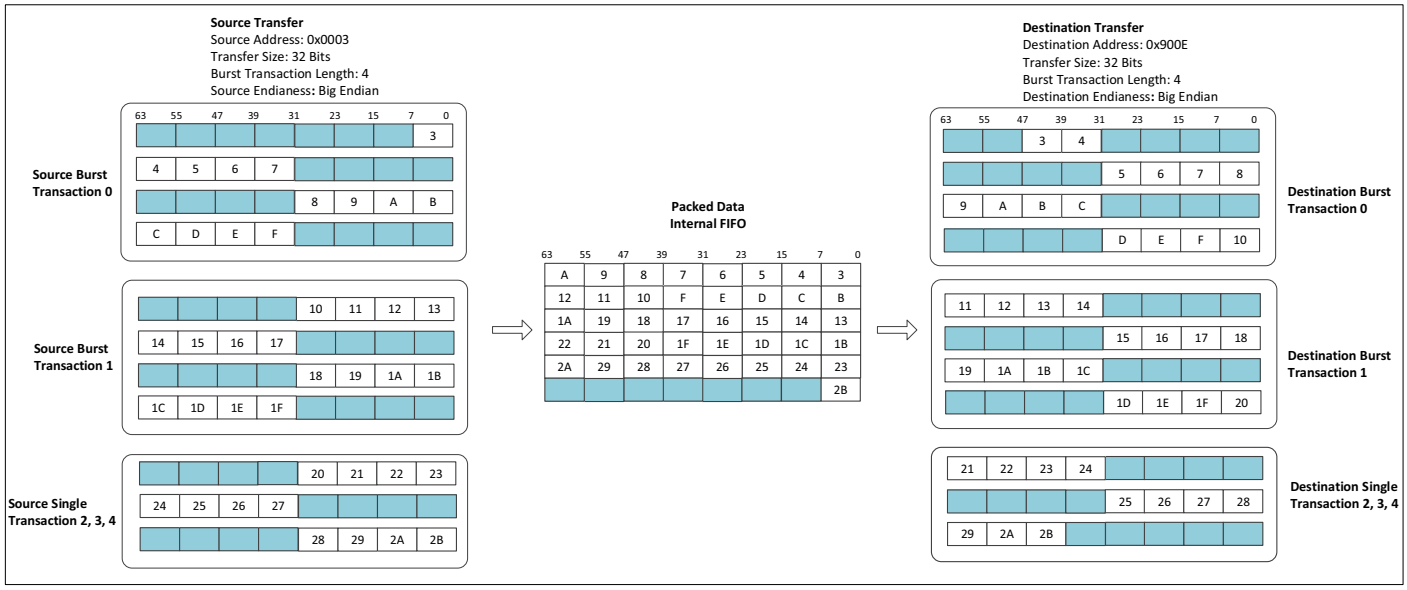
Table 2-16 describes all major DMA parameter settings considered for this example.

Table 2-16 Example 12 - Configuration and Programming Options (Continued)

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

Configuration/Programming	Value	Description
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 2-51 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 2-16 on page 48) that is all bytes of the data item are flipped. Similar operation is continued, on the subsequent bytes.

2.17 Channel Suspend, Disable, and Terminate

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 terminate a channel before a transfer completes. The suspend, disable, and terminate procedures are explained in the following sections.

2.17.1 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 Manager interface, which could result in the DMA channel going into waiting mode.
- The requests initiated by source/destination/LLI state machines and present in the Manager Interface Read Address Channel and Write Address Channel FIFOs to be sent on the AXI Manager Interface will be sent on the AXI Manager Interface even if the Channel Suspend request is initiated. Based on the Manager 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 is 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.

2.17.2 Channel Suspend and Resume

To suspend and resume a channel:

1. Follow Step 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.

2.17.3 Channel Suspend and Disable Prior to Transfer Completion

To suspend and disable a channel:

1. Follow Step 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.

2.17.4 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 Manager interface, which could lead to a deadlock.
> - The requests initiated by source/destination/LLI state machines and present in the Manager Interface Read Address Channel and Write Address Channel FIFOs to be sent on the AXI Manager Interface will be sent on the AXI Manager Interface even if the Channel Suspend request is initiated. Based on the Manager 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.

2.17.5 Abnormal Channel Terminate

Terminating 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 stops responding due to not receiving a response from the corresponding handshaking interface. Before terminating a channel, it is recommended that you first attempt to disable a channel.

To terminate a channel:

1. Software writes a 1 to the channel termination 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 terminated.
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.

2.18 DMA Channel Status Register – CHx_STATUSREG

The DW_axi_dmac supports the Channel Status Register and it is used to provide the additional debug capability. This register supports following two fields:

- Completed Block Transfer Size - `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE`
- Data Left in the FIFO - `CHx_STATUSREG.DATA_LEFT_IN_FIFO`

2.18.1 Completed Block Transfer Size

The Completed Block Transfer Size status indicates the total number of data items transferred from Source memory/peripheral to the Destination memory/peripheral at any point in the DMA block transfer. This field helps user to track the number of bytes has been transferred at any point in the DMA block transfer.

This status accessible through the register field `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE`. The completed block transfer size or data items is defined in terms of:

- When DMA or Source is a Flow controller this field is defined in terms of the `CHx_CTL.SRC_TR_WIDTH`.
- When Destination is a Flow Controller this field is defined in terms of the `CHx_CTL.DST_TR_WIDTH`.

This register field gets updated every time an AXI write transfer is generated to the destination. This field gets updated irrespective of whether a `SLVERR` or `DECERR` is received along with write response. The soft reset (`DMAC_ResetReg.DMAC_RST`) clears the `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE` field to 0.

Note

- In the use cases with multi-block transfer, the register field `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE` retains number of data items transferred in the previous block (at the end of block) until the next blocks first AXI write transfer is generated. This helps user to read the register field `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE` even after the last block is ended.
- The register field `CHx_STATUSREG.CMPLTD_BLK_TFR_SIZE` gets cleared when DMA transfer is completed and channel is enabled again.

2.18.2 Data Left in the FIFO – CHx_STATUSREG.DATA_LEFT_IN_FIFO

The data left in the FIFO status indicates the total number of data left in the channel FIFO after completing the current DMA block transfer or DMA block abnormal termination due to the following reasons:

- AXI Transfer Error response (`SLVERR` or `DECERR`) reception
- Channel Abort

This status is accessible through the register field `CHx_STATUSREG.DATA_LEFT_IN_FIFO`. The data left in the FIFO is defined in terms of the `CHx_CTL.SRC_TR_WIDTH`. This field helps user to know the number of bytes left in the Channel FIFO in the above-mentioned conditions.

This register field gets updated only in the following conditions:

- End of the block
- On reception of AXI Error (`SLVERR` or `DECERR`)
- Channel Abort

The normal block transfer completion without errors or abort this field should be equal to 0. If any AXI Error or Channel Abort 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. The soft reset (`DMAC_ResetReg.DMAC_RST`) clears the `CHx_STATUSREG.DATA_LEFT_IN_FIFO` field to 0.

Note

- In the use cases with multi-block transfer, the register field `CHx_STATUSREG.DATA_LEFT_IN_FIFO` retains last updated value of the previous block until the end of block or AXI Error reception or Channel abort received for the current block.
- The register field `CHx_STATUSREG.DATA_LEFT_IN_FIFO` gets cleared when DMA transfer is completed, and channel is enabled again.

2.19 Debug Interface

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

2.19.1 Register Bus Interface and Controller 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 Register Bus Interface is busy. This signal is asserted High if there is an active transfer on the Register Bus interface.
- `dmac_busy` – indicates whether the controller 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 Manager 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 Manager Interface from the arbiter.

2.19.2 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 (manager) 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_mN` output once `arready_mN` is received.
   - DW_axi_dmac de-asserts `rready_mN` 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 Manager 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_mN` and `wvalid_mN` output when `awready_mN` and `wready_mN` is received.
   - DW_axi_dmac de-asserts `bready_mN` output when data transfer on the AXI write data channel is completed.

Note There may be write requests from many channels on the Manager 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.

2.19.3 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 is 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.

3

Mission Critical Features

The DW_axi_dmac controller supports the following features:

- "Register Bus Interface Parity Feature" on page 118
- "Error Correction Code (ECC) Protection Feature" on page 122
- "ECC Protection Feature for FIFO Memory Interface" on page 125
- "ECC Protection Feature for UID Memory Interface" on page 129
- "ECC and Parity Protection Feature for AXI Manager Interface" on page 134
- "Lock Step Protection Feature" on page 150

3.1 Register Bus Interface Parity Feature

This section covers the following topics:

- "Overview of the Register Bus Interface Parity Feature" on page 118
- "AHB/APB4 Interface Address Parity" on page 118
- "AHB/APB4 Interface Data Parity" on page 118
- "Interface Parity Error Interrupt" on page 119
- "Interface Parity Error Interrupt Signal" on page 120
- "Enabling Register Bus Interface Parity Feature" on page 121
- "Signals Related to Register Bus Interface Parity Feature" on page 121
- "Registers Related to the Register Bus Interface Parity Feature" on page 121

3.1.1 Overview of the Register Bus Interface Parity Feature

The data integrity of Control and Status Register is critical for any application, including mission-critical applications such as automotive. The DW_axi_dmac controller implements the Parity Protection feature for all Common Registers, and Channel Specific Registers at the interface level to perform the data integrity checks through the parity.

The Parity protection feature provides higher level of data integrity check at the interface level for all Common Registers and Channel Specific Registers. This data integrity feature is available on both the AHB and APB4 Register Bus Interfaces. A configurable parameter `DMAX_PARITY_PROT_EN` is provided to enable the Parity Protection feature.

Use the PAR_EN bit of the `DMAC_PARITYCTLREG` register to enable or disable the Parity Protection feature through the software interface. Use the programming register field `PARITY_MODE` of the `DMAC_PARITYCTLREG` register to choose between the odd and even parity. Use the `DMAX_DEF_PARITY_MODE` parameter to configure the default value of the odd and even parity. Like any other register, the value of the `DMAC_PARITYCTLREG` register is resettable based on the software reset.

3.1.2 AHB/APB4 Interface Address Parity

The DW_axi_dmac controller receives parity per byte during the AHB/APB4 transfer. These parity signals are received through the side-band signals. For example, if the address bus width is 32 bits, then 4-bit parity is received. The locally-computed parity for the received address is compared against the received parity through the AHB/APB4 transfer. The comparison result is indicated through the `hresp/pslverr` signal depending on AHB or APB4 interface. In case of failure, Error response is returned through the `hresp/pslverr` signal, and the write/read operation will be unsuccessful.

3.1.3 AHB/APB4 Interface Data Parity

The DW_axi_dmac controller receives parity per byte during the AHB/APB4 write transfer. These parity signals are received through the side-band signals. For example, if the data bus width is 32 bits, then 4-bit parity is received. The locally-computed parity for the received write data is compared against the received parity through the AHB/APB4 write transfer. The comparison result is indicated through the `hresp/pslverr` signal depending on AHB or APB4 interface. In case of failure, Error response is returned through the `hresp/pslverr` signal, and the write operation will be unsuccessful.

The DW_axi_dmac controller provides parity per byte during the AHB/APB4 read transfer along with the read data. These parity signals are sent through the side-band signals. They are computed statically using

an internal parity computation logic. The receiving manager interface can use this read parity to check the data integrity of the read data. This read parity is applicable only for the readable registers.

3.1.4 Interface Parity Error Interrupt

If parity fails because of the parity error, the DW_axi_dmac controller generates the Interface Parity Error Interrupt along with the interface error on the `hresp/pslverr` signal.

The DW_axi_dmac controller generates the following Address Interface Parity Error Interrupt:

- **All Common Registers**: Address Interface Parity failure has an Address Interface Parity Error Interrupt (`SLVIF_CommonReg_ADDRPARITY_ERR_IntStat`). The respective interrupt bit is present in the following Common Interrupt registers:
  - `DMAC_COMMONREG_INTSTATUSREG`
  - `DMAC_COMMONREG_INTSTATUS_ENABLEREG`
  - `DMAC_COMMONREG_INTSIGNAL_ENABLEREG`
  - `DMAC_COMMONREG_INTCLEARREG`
- **All Channel-Specific Registers**: Address Interface Parity failure has an Address Interface Parity Error Interrupt (`SLVIF_ADDRPARITY_ERR_IntStat`). The respective interrupt bit is present in the following Channel-Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`

The DW_axi_dmac controller generates the following Write Interface Parity Error Interrupt:

- **All Common Registers**: Write Interface Parity failure has a Write Interface Parity Error Interrupt (`SLVIF_CommonReg_WRPARITY_ERR_IntStat`). The respective interrupt bit is present in the following Common Interrupt registers:
  - `DMAC_COMMONREG_INTSTATUSREG`
  - `DMAC_COMMONREG_INTSTATUS_ENABLEREG`
  - `DMAC_COMMONREG_INTSIGNAL_ENABLEREG`
  - `DMAC_COMMONREG_INTCLEARREG`
- **All Channel-Specific Registers:** Write Interface Parity failure has a Write Interface Parity Error Interrupt (`SLVIF_WRPARITY_ERR_IntStat`). The respective interrupt bit is present in the following Channel-Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`.

After Interface Parity Error interrupt is asserted, the software performs the following tasks:

1. Reads the `DMAC_COMMONREG_INTSTATUSREG` or `CHx_INTSTATUS` register.
2. Executes the necessary interrupt service routine.

3. Clears the respective interrupt through the DMAC_COMMONREG_INTCLEARREG or CHx_INTCLEARREG register.

Note

- The Interface Parity Error Interrupt (DMAC_COMMONREG_INTSTATUSREG.SLVIF_Common-Reg_WRPARITY_ERR_IntStat, or DMAC_COMMONREG_INTSTATUSREG.SLVIF_Common-Reg_ADDRPARITY_ERR_IntStat) is not combined into the DMAC_INTSTATUSREG.CommonReg_IntStat register field bit.
- If there is an Interface Parity error, the Interface Parity Error Interrupt is generated even if there are other errors on the register interface (for example, a write to read-only register, a read to write-only register, an invalid write on channel hold, and so on).

3.1.5 Interface Parity Error Interrupt Signal

The DW_axi_dmac controller has a separate set of Interface Parity Error signals at the top level. These signals are level signals and are de-asserted when the respective interrupt is cleared through the Interrupt Clear registers – DMAC_COMMONREG_INTCLEARREG and CHx_INTCLEARREG. The number of Interface Parity Error interrupt signals available on the top-level port depends on the DMAX_INTR_IO_TYPE parameter.

When DMAX_INTR_IO_TYPE = COMBINED_ONLY (0)

- Combined – Address Interface Parity Error (Combines Address Interface Parity Error from Common Register Space and Channel-Specific Register Space)
- Combined – Write Interface Parity Error (Combines Write Interface Parity Error from Common Register Space and Channel-Specific Register Space)

When DMAX_INTR_IO_TYPE = CHANNEL_AND_COMMONREG (1)

- Common Register Space – Address Interface Parity Error
- Channel-Specific Register Space – Address Interface Parity Error (DMAX_NUM_CHANNELS bits)
- Common Register Space – Write Interface Parity Error
- Channel-Specific Register Space – Write Interface Parity Error (DMAX_NUM_CHANNELS bits)

When DMAX_INTR_IO_TYPE = ALL_INTERRUPT_OUTPUTS (2)

- Combined – Address Interface Parity Error
- Common Register Space – Address Interface Parity Error
- Channel-Specific Register Space - Address Interface Parity Error (DMAX_NUM_CHANNELS bits)
- Combined – Write Interface Parity Error
- Common Register Space – Write Interface Parity Error
- Channel-Specific Register Space – Write Interface Parity Error (DMAX_NUM_CHANNELS bits)

Note

- If a parity error occurs while the corresponding interrupts have already been asserted and not cleared yet, then the dedicated Parity Error Interrupt signal remains asserted and the Parity error is communicated through the hresp/pslverr signals of the AHB/APB4 interface.
- If parity error does not occur for the next transfer, then the transfer goes through without any error response.

3.1.6 Enabling Register Bus Interface Parity Feature

To enable the Register Bus Interface Parity Feature, choose the "Yes" option for the "Register Bus Interface Parity Check Enable?" drop-down using the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.1.7 Signals Related to Register Bus Interface Parity Feature

The following signals are related to the Register Bus Interface Parity feature:

- **APB4 Register Bus Interface Signals**
  - `pauser[DMAX_S_ADDR_WIDTH/8-1:0]`
  - `pwuser[DMAX_S_DATA_WIDTH/8-1:0]`
  - `pruser[DMAX_S_DATA_WIDTH/8-1:0]`
- **AHB Register Bus Interface Signals**
  - `hauser[DMAX_S_ADDR_WIDTH/8-1:0]`
  - `hwuser[DMAX_S_DATA_WIDTH/8-1:0]`
  - `hruser[DMAX_S_DATA_WIDTH/8-1:0]`
- **Parity Protection Interrupt Signals**
  - `addrparity_prot_err_intr`
  - `addrparity_prot_err_intr_ch[DMAX_NUM_CHANNELS-1: 0]`
  - `addrparity_prot_err_intr_cmnreg`
  - `wrparity_prot_err_intr`
  - `wrparity_prot_err_intr_ch[DMAX_NUM_CHANNELS-1: 0]`
  - `wrparity_prot_err_intr_cmnreg`

For more information about these signals, see the "Signal Descriptions" chapter.

3.1.8 Registers Related to the Register Bus Interface Parity Feature

The following registers are related to the Register Bus Interface Parity feature:

- `DMAC_PARITYCTLREG`

For more information about these registers, see the "Register Descriptions" chapter.

3.2 Error Correction Code (ECC) Protection Feature

This section covers the following topics:

- "Overview of ECC Protection Feature" on page 122
- "Description of ECC Protection Feature" on page 122
- "Enabling the ECC Protection Feature" on page 124
- "Registers Related to ECC Protection Feature" on page 124

3.2.1 Overview of ECC Protection Feature

The Error Correction Code (ECC) Protection feature enables the protection against the data corruption. Data corruption takes place due to the soft errors that occur because of the charged particles or radiation. ECC Protection feature can correct single bit error and detect the double bit errors.

3.2.2 Description of ECC Protection Feature

ECC Protection feature is enabled through the parameter `DMAX_ECC_PROT_EN`. If enabled, this allows to add ECC Protection feature for:

- FIFO Memory Interface
- UID Memory Interface
- AXI Manager Interface

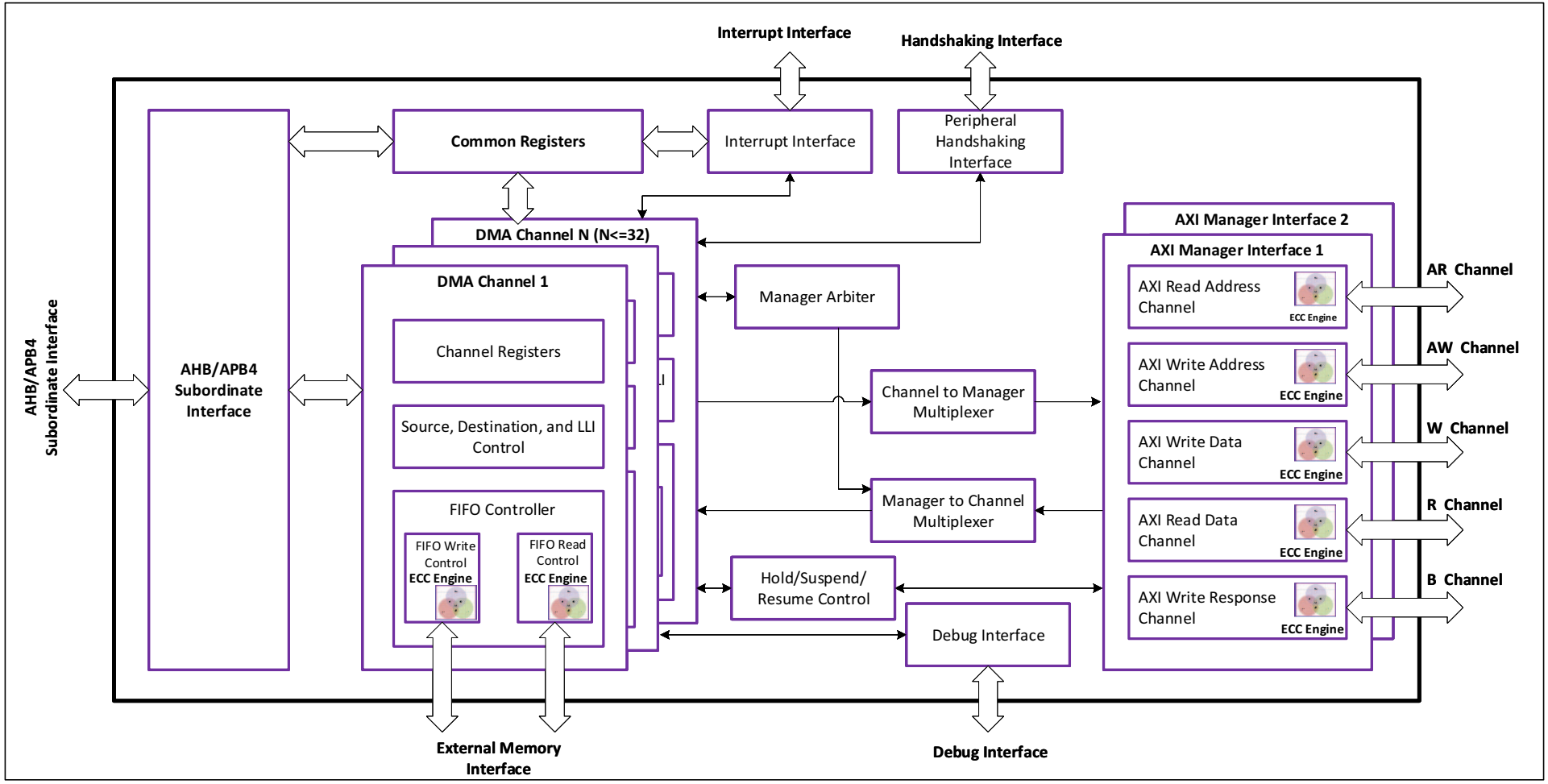
You can enable or disable the ECC Protection feature through the following parameters:

- `DMAX_ECC_MEMIF_EN` – to enable the ECC Protection for FIFO Memory Interface
- `DMAX_ECC_UIDMEMIF_EN` – to enable the ECC Protection feature for UID Memory Interface
- `DMAX_ECC_MXIF_EN` – to enable the ECC Protection for AXI Manager Interface
- `DMAX_ECC_DIAG_EN` – to enable ECC Protection related additional diagnostic features

Note AXI Manager Interface not only supports ECC Protection feature, it also supports highly configurable ECC/Parity feature combinations. For more information, see "ECC and Parity Protection Feature for AXI Manager Interface" on page 134.

Figure 3-1 describes the ECC Protection feature incorporated in the DW_axi_dmac controller.

Figure 3-1 ECC Protection Feature - Overall Block Diagram



- "ECC Protection Feature for FIFO Memory Interface" on page 125
- "ECC Protection Feature for UID Memory Interface" on page 129
- "ECC and Parity Protection Feature for AXI Manager Interface" on page 134

3.2.3 Enabling the ECC Protection Feature

To enable ECC Protection feature, click the "Include ECC Protection feature?" check-box in the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.2.4 Registers Related to ECC Protection Feature

The following registers are related to the ECC Protection feature:

- `DMAC_COMMONREG_ECCCTLSTATUSREG`

For more information about these registers, see the "Register Descriptions" chapter.

3.3 ECC Protection Feature for FIFO Memory Interface

This section covers the following topics:

- "Overview of ECC Protection Feature for FIFO Memory Interface" on page 125
- "Description of ECC Protection Feature for FIFO Memory Interface" on page 125
- "Error Injection Capabilities for FIFO Memory Interface" on page 126
- "Diagnostic Support for Error Management for FIFO Memory Interface" on page 126
- "Enabling the ECC Protection Feature for FIFO Memory Interface" on page 128

3.3.1 Overview of ECC Protection Feature for FIFO Memory Interface

Enable the ECC Protection feature on the FIFO Memory Interface to protect the FIFO Memory. This enables the ECC engine that computes ECC checkbits for every word written into the FIFO memory. The computed ECC checkbits are stored along with the data that is being written into the FIFO. The computation of ECC checkbits is done based on the data or payload that is being written into FIFO, and the FIFO address into which the data is written. The FIFO address consideration for ECC computation helps detecting the data integrity issues that occur because of address corruption.

3.3.2 Description of ECC Protection Feature for FIFO Memory Interface

While reading the data from the FIFO memory, ECC checkbits are recomputed using the ECC engine. The ECC checkbits computation is done based on the read data and FIFO address from which the data is read. The recomputed ECC checkbits are compared against the read ECC checkbits to form an error syndrome. Then, the ECC Engine corrects the single-bit error, if any, and indicates the multiple bit error through the error syndrome. The `CHx_INTSTATUS` register indicates these single-bit errors and multi-bit errors through the dedicated error signal and through the Error interrupt. "Diagnostic Support for Error Management for FIFO Memory Interface" on page 126 provides more information about the error management.

The number of ECC checkbits depends on the FIFO data width and FIFO memory address width (that is dependent on the FIFO depth). Table 3-1 provides the few typical FIFO memory widths, FIFO memory depths, ECC engine input data width, and checkbits size examples.

Table 3-1 ECC Protection Feature - FIFO Memory Interface Typical Examples

FIFO Memory Width	FIFO Memory Depth	ECC Engine Input Data Width	Required ECC Checkbits	Total Data Width (with ECC)
32	8	35	7	42
32	32	37	7	44
64	8	67	8	75
64	32	69	8	77
64	256	72	8	80
128	32	133	9	142
128	64	134	9	143
256	32	261	10	271

FIFO Memory Width	FIFO Memory Depth	ECC Engine Input Data Width	Required ECC Checkbits	Total Data Width (with ECC)
512	32	517	11	528

Table 3-1 shows the typical combinations. You can choose other combinations as well, provided the FIFO memory width and depth are within the permissible range of the DW_axi_dmac controller. The width of the FIFO increases with ECC Protection feature because you need to store the ECC checkbits along with the data in the FIFO.

3.3.3 Error Injection Capabilities for FIFO Memory Interface

The DW_axi_dmac controller supports error injection capabilities when the `DMAX_ECC_ERR_INJ_EN` parameter is set to 1. The DW_axi_dmac controller enables ECC error injection by programming the `SFTY_EI_EN` register bit of the `DMAC_SFTY_ERR_INJ_CTRL` register to 1. After the error injection logic inserts error in the address, data, or ECC checkbits, the error is generated at the subsequent ECC checker. The correctable or uncorrectable safety error is indicated by the respective fields of the `CHx_INTSTATUS` register. The correctable or uncorrectable safety interrupt is also asserted when an ECC error is detected.

Note For the FIFO memory interface, the error is injected before the data is written into the FIFO memory and the error is reported at the ECC checker. The FIFO memory interface for error injection can be selected by programming the `SFTY_EI_FEATURE` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register to 0.

The DW_axi_dmac controller must undergo the following programming sequence to inject the data path error.

- Enable the error injection using the `SFTY_EI_EN` register bit in the `DMAC_SFTY_ERR_INJ_CTRL` register).
- Select the target channel memory interface for error injection using the `SFTY_EI_FEATURE` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the target channel for error injection using the `SFTY_EI_CH` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the point of error insertion (address, data, or ECC checkbits) using the `ECC_EI_POINTS` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- If the `SFTY_EI_EN` register bit is 1, choose 1-bit or 2-bit error injection using the `ECC_EI_TYPE` register bit in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the bit position where the error needs to be injected. In case of 2-bit error injection, bit 0 is always included in error injection and the second bit location is selected using the `SFTY_BIT_SEL` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.

3.3.4 Diagnostic Support for Error Management for FIFO Memory Interface

Diagnostic features provide the following statistics to monitor the ECC error behavior in the FIFO Memory interface.

3.3.4.1 ECC Protection – Correctable Error Interrupt

The ECC Protection Correctable Error interrupt is generated when a single bit error is detected on the data, and it is corrected through the ECC engine.

The following ECC Protection Correctable Error Interrupt is generated:

- All Channel Specific Register ECC Protection has a Correctable Error Interrupt (`ECC_PROT_CH-Mem_CorrERR_IntStat`). The respective interrupt bit is added in the following Channel Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`

After this interrupt is asserted, the software

1. Read the `CHx_INTSTATUS` register
2. Executes the necessary interrupt service routine
3. Clears the respective interrupt through the `CHx_INTCLEARREG` register

3.3.4.2 ECC Protection – Uncorrectable Error Interrupt

The ECC Protection Uncorrectable Error interrupt is generated when a multiple bit error has been detected on the data that is not correctable through the ECC engine.

The following ECC Protection Uncorrectable Error Interrupt is generated:

- All Channel Specific Register ECC Protection has an Uncorrectable Error Interrupt (`ECC_PROT_CH-Mem_UnCorrERR_IntStat`). The respective interrupt bit is added in the following Channel Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`

After this interrupt is asserted, the software

1. Reads the `CHx_INTSTATUS` register
2. Executes the necessary interrupt service routine is executed
3. Clear the respective interrupt through the `CHx_INTCLEARREG` register

3.3.4.3 ECC Protection Error Interrupt Signal

The DW_axi_dmac controller adds a separate set of ECC Protection Error signals at the top level. These signals are level signals and are de-asserted when the respective interrupt is cleared through the Interrupt clear register `CHx_INTCLEARREG`.

The number of ECC Protection Error interrupt signals available on the top-level port depends on the `DMAX_INTR_IO_TYPE` parameter.

- When `DMAX_INTR_IO_TYPE = COMBINED_ONLY (0)`:
  - Combined – ECC Protection Correctable Error (Combines ECC Protection Correctable Error from all Channel FIFO Memory interface).
  - Combined – ECC Protection Uncorrectable Error (Combines ECC Protection Uncorrectable Error from all Channel FIFO Memory interface).
- When `DMAX_INTR_IO_TYPE = CHANNEL_AND_COMMONREG (1)`
  - Channel FIFO Memory Interface - ECC Protection Correctable Error (`DMAX_NUM_CHANNELS` bits)
  - Channel FIFO Memory Interface - ECC Protection Uncorrectable Error (`DMAX_NUM_CHANNELS` bits)
- When `DMAX_INTR_IO_TYPE = ALL_INTERRUPT_OUTPUTS (2)`
  - Combined – ECC Protection Correctable Error
  - Combined – ECC Protection Uncorrectable Error
  - Channel FIFO Memory Interface - ECC Protection Correctable Error (`DMAX_NUM_CHANNELS` bits)
  - Channel FIFO Memory Interface - ECC Protection Uncorrectable Error (`DMAX_NUM_CHANNELS` bits)

3.3.4.4 ECC Protection Error Counter Registers

The ECC Protection Control and Status register `DMAC_COMMONREG_ECCCTLSTATUSREG` is provided to support the enhanced error management capabilities. You can enable this additional diagnostic feature through the `DMAX_ECC_DIAG_EN` parameter. The `DMAC_COMMONREG_ECCCTLSTATUSREG` register:

- Provides access to all Channel FIFO Memory Interface related to ECC correctable and uncorrectable error and counters.
- Provides access to a channel related to ECC correctable and uncorrectable counters using the control fields `ECC_INTFC_TYPE` and `ECC_INTFC_NO`.
- Provides the capability to reset all the ECC counters without resetting any other functionality.

The ECC correctable error counter is incremented whenever a single-bit error is detected. The ECC uncorrectable error counter is incremented whenever a multi-bit error detected by the ECC engine.

See the `DMAC_ECCPROT_CTLSTATUSREG` Register in the Register Description chapter for more information about this register.

Note The ECC Channel Memory correctable and uncorrectable error counter register is not cleared with the Soft Reset or when Channel is disabled. It can be cleared only with the ECC Error Count Reset that is `DMAC_COMMON_ECCCTLSTATUSREG.ECC_ERR_COUNT_RST`.

3.3.5 Enabling the ECC Protection Feature for FIFO Memory Interface

To enable the ECC Protection Feature for FIFO Memory Interface, click the “Include ECC Protection for the FIFO Memory Interface?” field using the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.4 ECC Protection Feature for UID Memory Interface

This section covers the following topics:

- "Overview of ECC Protection Feature for UID Memory Interface" on page 129
- "Description of ECC Protection Feature for UID Memory Interface" on page 129
- "Error Injection Capabilities for UID Memory Interface" on page 130
- "Diagnostic Support for Error Management for UID Memory Interface" on page 130
- "Enabling the ECC Protection Feature for UID Memory Interface" on page 133

3.4.1 Overview of ECC Protection Feature for UID Memory Interface

Enable the ECC Protection feature on the UID Memory Interface to protect the UID Memory. This enables the ECC engine, that computes ECC checkbits for every word written into the UID memory. The computed ECC checkbits are stored along with the data that is being written into the UID memory. The computation of ECC checkbits is done based on the data or payload that is being written into UID memory, and UID address into which the data is written. The UID address consideration for ECC computation helps detecting the data integrity issues that occur because of address corruption.

3.4.2 Description of ECC Protection Feature for UID Memory Interface

While reading the data from the UID memory, ECC checkbits are recomputed using the ECC engine. The ECC checkbits computation is done based on the read data and UID address from which the data is read. The recomputed ECC checkbits are compared against the read ECC checkbits to form an error syndrome. Then, the ECC Engine corrects the single bit error, if any, and indicates the multiple bit error through the error syndrome. The `CHx_INTSTATUS` register indicates these single-bit errors and multi-bit errors through the dedicated error signal and through the Error interrupt. The "Diagnostic Support for Error Management for UID Memory Interface" on page 130 provides the more information about the error management.

The number of ECC checkbits depends on the UID data width and UID memory address width (that is dependent on the UID memory depth). Table 3-2 provides the few typical memory widths, memory depths, ECC engine input data width, and checkbits size examples.

Table 3-2 ECC Protection Feature - UID Memory Interface Typical Examples

UID Memory Width	UID Memory Depth	ECC Engine Input Data Width	Required ECC Checkbits	Total Data Width (with ECC)
32	8	35	7	42
32	32	37	7	44
64	8	67	8	75
64	32	69	8	77
64	256	72	8	80
128	32	133	9	142
128	64	134	9	143
256	32	261	10	271

UID Memory Width	UID Memory Depth	ECC Engine Input Data Width	Required ECC Checkbits	Total Data Width (with ECC)
512	32	517	11	528

Table 3-2 shows the typical combinations. You can choose other combinations as well, provided the UID memory width and depth are within the permissible range of DW_axi_dmac. The width of the UID memory increases with ECC Protection feature because you need to store the ECC checkbits along with the data in the UID memory.

3.4.3 Error Injection Capabilities for UID Memory Interface

The DW_axi_dmac supports error injection capabilities when the `DMAX_ECC_ERR_INJ_EN` parameter is set to 1. The DW_axi_dmac enables ECC error injection by programming the `SFTY_EI_EN` register bit in the `DMAC_SFTY_ERR_INJ_CTRL` register to 1. After the error injection logic inserts error in the address, data, or ECC checkbits, the error is generated at the subsequent ECC checker. The correctable or uncorrectable safety error is indicated by the respective fields of the `CHx_INTSTATUS` register. The correctable or uncorrectable safety interrupt is asserted when an ECC error is detected.

Note For the UID memory interface, the error is injected before the data is written into the UID memory and the error is reported at the ECC checker. The UID memory interface for error injection can be selected by programming the `SFTY_EI_FEATURE` register field in `DMAC_SFTY_ERR_INJ_CTRL` to 1.

The DW_axi_dmac controller must undergo the following programming sequence to inject the data path error.

- Enable the error injection using the `SFTY_EI_EN` register bit in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the target channel memory interface for error injection using the `SFTY_EI_FEATURE` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the target channel for error injection using the `SFTY_EI_CH` register field in the `DMAC_SFTY_ER-R_INJ_CTRL` register.
- Select the point of error insertion (address, data, or ECC checkbits) using the `ECC_EI_POINTS` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- If the `SFTY_EI_EN` register bit is 1, choose 1-bit or 2-bit error injection using the ECC_EI_TYPE register bit in the `DMAC_SFTY_ERR_INJ_CTRL` register.
- Select the bit position where the error needs to be injected. In case of 2-bit error injection, bit 0 is always included in error injection and the second bit location is selected using the `SFTY_BIT_SEL` register field in the `DMAC_SFTY_ERR_INJ_CTRL` register.

3.4.4 Diagnostic Support for Error Management for UID Memory Interface

Diagnostic features provide the following statistics to monitor the ECC error behavior in the UID Memory interface.

DW_axi_dmac generates the ECC Protection interrupts to indicate if the error is correctable or uncorrectable.

3.4.4.1 ECC Protection – Correctable Error Interrupt

The ECC Protection Correctable Error interrupt is generated when a single bit error is detected on the data, and it is corrected through the ECC engine.

The following ECC Protection Correctable Error Interrupt is generated:

- All Channel Specific Register ECC Protection has a Correctable Error Interrupt (`ECC_PROT_UID-Mem_CorrERR_IntStat`). The respective interrupt bit is added in the following Channel Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`

After this interrupt is asserted, the software

1. Read the `CHx_INTSTATUS` register
2. Executes the necessary interrupt service routine
3. Clear the respective interrupt through the `CHx_INTCLEARREG` register

3.4.4.2 ECC Protection – Uncorrectable Error Interrupt

The ECC Protection Uncorrectable Error interrupt is generated when a multiple bit error has been detected on the data that is not correctable through the ECC engine.

The following ECC Protection Uncorrectable Error Interrupt is generated:

- All Channel Specific Register ECC Protection has an Uncorrectable Error Interrupt (`ECC_PROT_UID-Mem_UnCorrERR_IntStat`). The respective interrupt bit is added in the following Channel Specific Interrupt registers:
  - `CHx_INTSTATUS`
  - `CHx_INTSTATUS_ENABLEREG`
  - `CHx_INTSIGNAL_ENABLEREG`
  - `CHx_INTCLEARREG`

After this interrupt is asserted, the software

1. Reads the `CHx_INTSTATUS` register
2. Executes the necessary interrupt service routine
3. Clears the respective interrupt through the `CHx_INTCLEARREG` register

3.4.4.3 ECC Protection Error Interrupt Signal

The DW_axi_dmac adds a separate set of ECC Protection Error signals at the top level. These signals are level signals and are de-asserted when the respective interrupt is cleared through the Interrupt clear register `CHx_INTCLEARREG`.

The number of ECC Protection Error interrupt signals available on the top-level port depends on the `DMAX_INTR_IO_TYPE` parameter.

- When `DMAX_INTR_IO_TYPE = COMBINED_ONLY (0)`:
  - Combined – ECC Protection Correctable Error (Combines ECC Protection Correctable Error from all Channel UID Memory interface).
  - Combined – ECC Protection Uncorrectable Error (Combines ECC Protection Uncorrectable Error from all Channel UID Memory interface).
- When `DMAX_INTR_IO_TYPE= CHANNEL_AND_COMMONREG (1)`:
  - Channel UID Memory Interface - ECC Protection Correctable Error (`DMAX_NUM_CHANNELS` bits)
  - Channel UID Memory Interface - ECC Protection Uncorrectable Error (`DMAX_NUM_CHANNELS` bits)
- When `DMAX_INTR_IO_TYPE= ALL_INTERRUPT_OUTPUTS (2)`:
  - Combined – ECC Protection Correctable Error
  - Combined – ECC Protection Uncorrectable Error
  - Channel UID Memory Interface - ECC Protection Correctable Error (`DMAX_NUM_CHANNELS` bits)
  - Channel UID Memory Interface - ECC Protection Uncorrectable Error (`DMAX_NUM_CHANNELS` bits)

3.4.4.4 ECC Protection Error Counter Registers

The ECC Protection Control and Status register `DMAC_COMMONREG_ECCCTLSTATUSREG` is provided to support the enhanced error management capabilities. You can enable this additional diagnostic feature through the `DMAX_ECC_DIAG_EN` parameter.

- Provides access to all Channel UID Memory Interface related to ECC correctable and uncorrectable error counters.
- Provides access to a channel related to ECC correctable and uncorrectable error using the control fields `ECC_INTFC_TYPE` and `ECC_INTFC_NO`.
- Provides the capability to reset all the ECC counters without resetting any other functionality.

The ECC correctable error counter is incremented whenever a single-bit error is detected. The ECC uncorrectable error counter is incremented whenever a multi-bit error detected by the ECC engine.

See the `DMAC_ECCPROT_CTLSTATUSREG` Register in the Register Description chapter for more information about this register.

Note

- The ECC UID Memory correctable and uncorrectable error counter register is not cleared with the Soft Reset or when Channel is disabled. It can be cleared only with the ECC Error Count Reset that is `DMAC_COMMON_ECCCTLSTATUSREG.ECC_ERR_COUNT_RST`.
- The ECC UID Memory Correctable and Uncorrectable interrupts are set only when respective channel is enabled.
- But under few scenarios such as outstanding read from UID Memory, UID Memory Correctable and Uncorrectable counter Register may get updated (due to ECC error) when Channel is disabled (due to an AXI Subordinate ERROR, Decode ERROR).

3.4.5 Enabling the ECC Protection Feature for UID Memory Interface

To enable the ECC Protection Feature for UID Memory Interface, click the “Include ECC Protection for the UID Memory Interface?” field using the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.5 ECC and Parity Protection Feature for AXI Manager Interface

This section covers the following topics:

- "Overview of ECC and Parity Protection Feature for AXI Manager Interface" on page 134
- "ECC and Parity Generator for AXI Interface" on page 134
- "ECC and Parity Monitor for AXI Interface" on page 135
- "AXI Channel Organization – ECC and Parity" on page 137
- "ECC Mode" on page 139
- "ECC Granularity" on page 141
- "Parity Mode" on page 143
- "Diagnostic Support for Error Management for AXI Manager Interface (with ECC and Parity Protection Feature)" on page 144
- "AXI Interface VALID and READY Parity Generator and Checker" on page 147
- "Enabling ECC and Parity Protection Feature for AXI Manager Interface" on page 149

3.5.1 Overview of ECC and Parity Protection Feature for AXI Manager Interface

The AXI Manager Interface can be protected by enabling the ECC and Parity Protection feature. This enables the ECC engine, Parity logic, or both on each of the following AXI interface channels:

- AR channel/AW channel
- W channel
- R channel
- B channel

The AXI Manager Interface ECC and Parity Protection feature is implemented by using the AXI Interface ECC and Parity Generator and Monitor blocks.

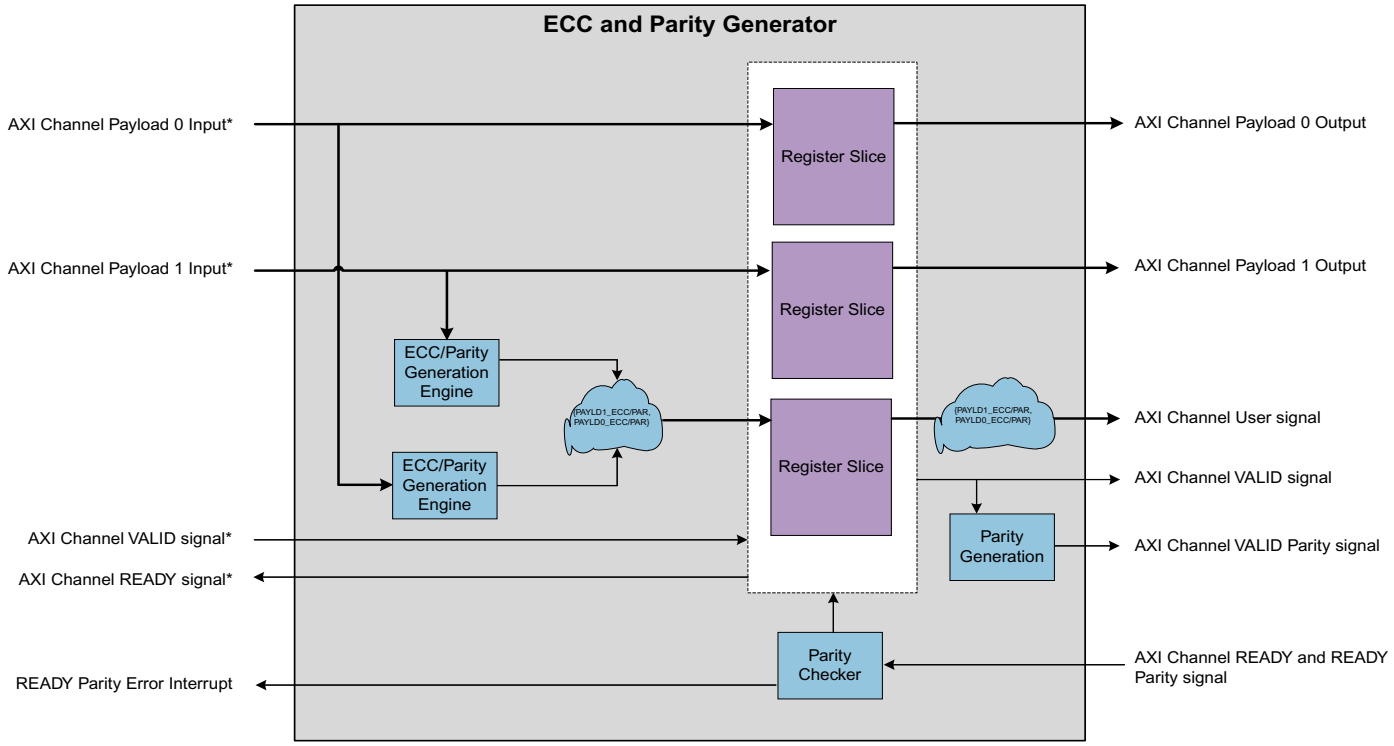
Note

- The ECC Protection feature for the FIFO Memory Interface and the UID Memory Interface supports only the DesignWare ECC (DW_ecc) Engine.
- The ECC Protection feature for the AXI Manager Interface supports DesignWare ECC, ARC Compatible ECC, and User-Defined ECC Engine.
- The Parity Protection feature for the AXI Manager Interface supports ARC Compatibility Parity Mode.

3.5.2 ECC and Parity Generator for AXI Interface

Figure 3-2 shows the block diagram of the AXI Interface ECC and Parity Generator. It illustrates how the ECC/Parity Engine and Register Slices are organized in the DW_axi_dmac controller for AR, AW, and W channels.

Figure 3-2 ECC and Parity Generator for AXI Interface – Block Diagram



In Figure 3-2,

* - indicates that these signals are coming from the internal AXI Manager Interface logic.

The AXI Payload 0 Input is defined as follows:

- AR/AW channel – Address of AR and AW channel
- W channel – Write Data of the W channel

The AXI Payload 1 Input is defined as follows:

- AR/AW channel – Control signals
- W channel – Control signals

The ECC Engine Logic is configurable using the following parameters:

- `DMAX_MXIF_ECC_MODE`
- `DMAX_MXIF_ECC_GRAN_TYPE`

The Parity Logic is configurable using the following parameters:

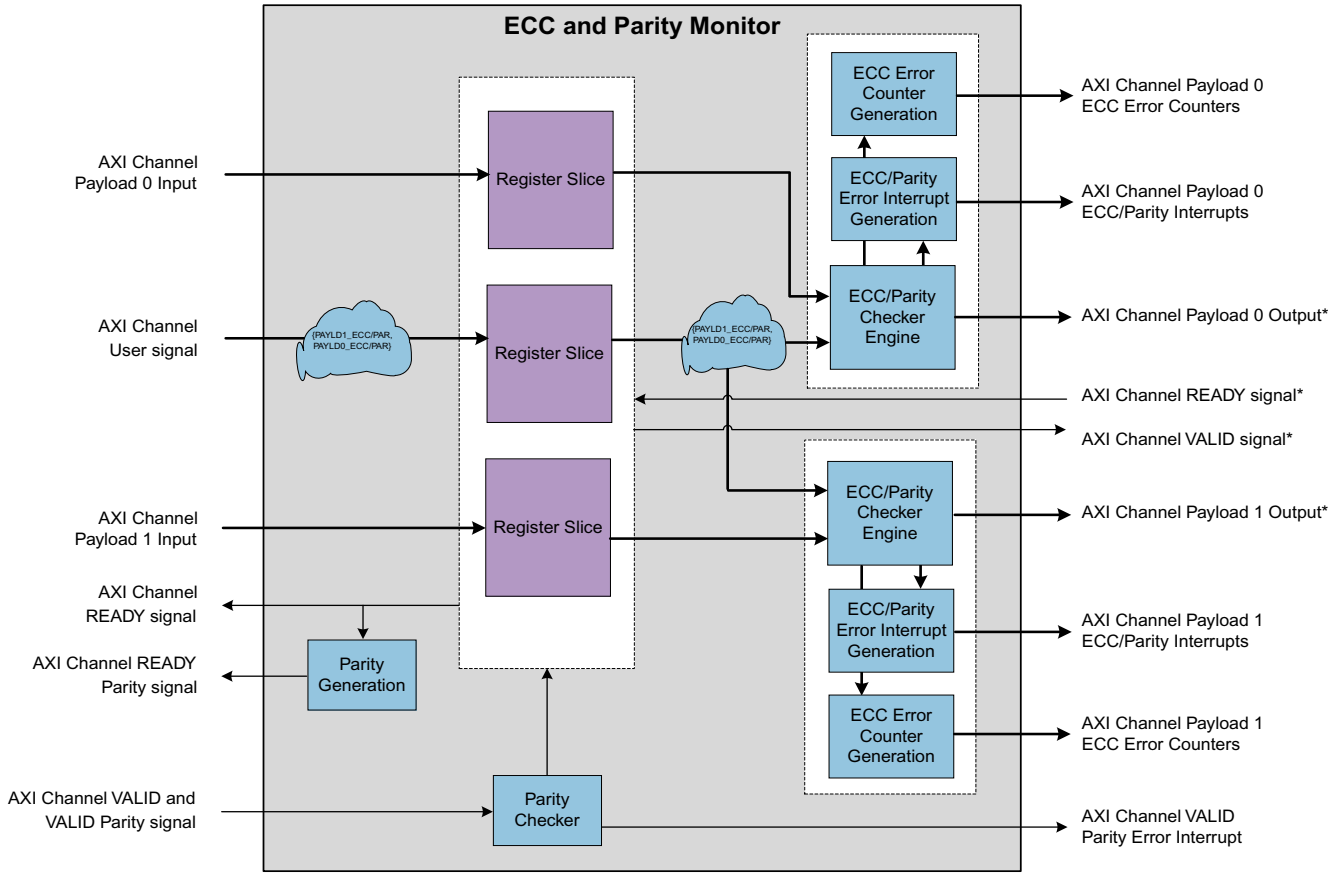
- `DMAX_MXIF_PAR_MODE`
- `DMAX_MXIF_PAR_TYPE`

For details on these parameters, see the “Parameter Descriptions” chapter.

3.5.3 ECC and Parity Monitor for AXI Interface

Figure 3-3 shows the block diagram of the AXI Interface ECC and Parity Monitor. It illustrates how the ECC/Parity Engine and Register slices are organized in the DW_axi_dmac controller for R and W channels.

Figure 3-3 ECC and Parity Monitor for AXI Interface – Block Diagram



In Figure 3-3,

* - indicates that these signals are going to the internal AXI Manager Interface logic.

The AXI Payload 0 Input is defined as follows:

- R channel – Read Data of R Channel
- B channel – Write Response Control signals

The AXI Payload 1 Input is defined as follows:

- R channel – Control signals
- B channel – Not Applicable as all the signals considered in the Payload 0.

The ECC Engine Logic is configurable using the following parameters:

- `DMAX_MXIF_ECC_MODE`
- `DMAX_MXIF_ECC_GRAN_TYPE`

The Parity Logic is configurable using the following parameters:

- `DMAX_MXIF_PAR_MODE`
- `DMAX_MXIF_PAR_TYPE`

For details on these parameters, see “Parameter Descriptions” chapter.

3.5.4 AXI Channel Organization – ECC and Parity

3.5.4.1 AR/AW Channel Organization

The AXI Manager Interface AR/AW Channel Safety block is implemented using the AXI Interface ECC and Parity Generator. For more details on the AXI Interface ECC and Parity Generator, see "ECC and Parity Generator for AXI Interface" on page 134.

The AXI Manager Interface AR/AW Channel signals are grouped into the following two payloads for the computation of ECC and Parity:

- AR/AW Channel Address (`ARADDR/AWADDR`)
- AR/AW Channel Control Signals (`ARCTRL/AWCTRL`)

The AR Channel Payload 0 and Payload 1 used for computation of ECC and Parity is packed as follows:

- `AR Channel Payload 0 = ARADDR`
- `AR Channel Payload 1 = {ARQOS, ARPROT, ARCACHE, ARLOCK, ARBURST, ARSIZE, ARLEN, ARID}`

Similarly, the AW Channel Payload 0 and Payload 1 used for computation of ECC and Parity is packed as follows:

- `AW Channel Payload 0 = AWADDR`
- `AW Channel Payload 1 = {AWQOS, AWPROT, AWCACHE, AWLOCK, AWBURST, AWSIZE, AWLEN, AWID}`

Note Depending on the configuration,

- `ARQOS/AWQOS` signal may or may not be present based on the AXI4 or AXI3 configuration.
- AR/AW Channel Control signal width depends on the configuration.

The AXI Manager Interface AR/AW Channel handshake signals `ARVALID/AWVALID` and `ARREADY/AWREADY` are protected with parity as described in the section "AXI Channel Parity/ECC Error Interrupts" on page 144.

The computed ECC and Parity signals are packed and sent over the ARUSER/AWUSER as follows:

- `ARUSER = {AR Channel Payload 1 ECC/Parity, AR Channel Payload 0 ECC/Parity}`
- `AWUSER = {AW Channel Payload 1 ECC/Parity, AW Channel Payload 0 ECC/Parity}`

The `ARVALID/AWVALID/ARREADY/AWREADY` parity signal is sent over the non-standard sideband signal.

3.5.4.2 W Channel Organization

The AXI Manager Interface W Channel Safety block is implemented using the AXI Interface ECC and Parity Generator. For more details on the AXI Interface ECC and Parity Generator, see "ECC and Parity Generator for AXI Interface" on page 134.

Note For the narrow transfer, the ECC computation is done based on the entire WDATA driven on the Write data channel that is both valid byte lanes and invalid byte lanes (may have random write data) are considered. User needs to consider this requirement in the subsystem while computing the ECC at the receiving end.

The AXI Manager Interface W Channel signals are grouped into the following two payloads for the computation of ECC and Parity:

- W Channel Data (`WDATA`)
- W Channel Control Signals (`WCTRL`)

The W Channel Payload 0 and Payload 1 used for computation of ECC and Parity is packed as follows:

- `W Channel Payload 0 = WDATA`
- `W Channel Payload 1 = {WLAST, WSTRB, WID}`

Note Depending on the configuration,

- WID signal may or may not be present based on the AXI4 or AXI3 configuration.
- W Channel Control signal width changes depending on the configuration.

The AXI Manager Interface W Channel handshake signals WVALID and WREADY are protected with parity as described in section "AXI Channel Parity/ECC Error Interrupts" on page 144.

The computed ECC and Parity signals are packed and sent over the WUSER as follows:

`WUSER = {W Channel Payload 1 ECC/Parity, W Channel Payload 0 ECC/Parity}`

The `WVALID/WREADY` parity signal is sent over the non-standard sideband signal.

3.5.4.3 R Channel Organization

The AXI Manager Interface R Channel Safety block is implemented using the AXI Interface ECC and Parity Monitor. For more details on the AXI Interface ECC and Parity Monitor, see "ECC and Parity Monitor for AXI Interface" on page 135.

Note For the narrow transfer, the ECC computation need to be done based on the entire RDATA driven on the Read data channel that is both valid byte lanes and invalid byte lanes (may have random read data) are considered. User needs to consider this requirement in the subsystem while computing the ECC at the receiving end.

The AXI Manager Interface R Channel signals are grouped into the following two payloads for the validation of ECC and Parity:

- R Channel Data (`RDATA`)
- R Channel Control Signals (`RCTRL`)

The R Channel Payload 0 and Payload 1 used for validation of ECC and Parity is packed as follows:

- `R Channel Payload 0 = RDATA`
- `R Channel Payload 1 = {RLAST, RRESP, RID}`

Note R Channel Control signals width changes based on the configuration.

The AXI Manager Interface R Channel handshake signals RVALID and RREADY are protected with parity as described in section "AXI Interface VALID and READY Parity Generator and Checker" on page 147.

The computed ECC and Parity signals are packed and sent over the RUSER as follows:

`RUSER = {R Channel Payload 1 ECC/Parity, R Channel Payload 0 ECC/Parity}`

The `RVALID/RREADY` parity signal is sent over the non-standard sideband signal.

3.5.4.4 B Channel Organization

The AXI Manager Interface B Channel Safety block is implemented using the AXI Interface ECC and Parity Monitor. For more details on the AXI Interface ECC and Parity Monitor, see "ECC and Parity Monitor for AXI Interface" on page 135.

The AXI Manager Interface B Channel signals are grouped into the following payload for the validation of ECC and Parity:

- B Channel Control Signals (`BCTRL`)

The B Channel Payload 0 used for validation of ECC and Parity is packed as follows:

- `B Channel Payload 0 = {BRESP, BID}`

Note B Channel Control signals width changes based on the configuration.

The AXI Manager Interface B Channel handshake signals `BVALID` and `BREADY` are protected with parity as described in section "AXI Interface VALID and READY Parity Generator and Checker" on page 147".

The computed ECC and Parity signals are packed and sent over the BUSER as follows:

`BUSER = {B Channel Payload 0 ECC/Parity}`

The `BVALID/BREADY` parity signal is sent over the non-standard sideband signal.

3.5.5 ECC Mode

3.5.5.1 DesignWare ECC Engine

The DesignWare ECC engine is implemented based on the Single-bit Error Correction – Double-bit Error Detection modified Hamming code. The ECC architecture is same as that of DW_ecc (DesignWare part), which is widely used in the industry. For more information, see the databook of the DW_ecc.

Note For information related to ECC checkbits and syndrome generation, see the following article: Generating checkbit equations and error syndrome map for DW_ecc

Table 3-3 shows how the ECC checkbits width changes against the input ECC Payload width.

Table 3-3 DesignWare ECC – ECC Payload Width vs ECC Checkbits Width

Payload Width	ECC Checkbits Width
Up to 11 bits	5
Up to 26 bits	6
Up to 57 bits	7
Up to 120 bits	8
Up to 247 bits	9
Up to 502 bits	10
Up to 1013 bits	11

3.5.5.2 ARC-Compatible ECC Engine

This implementation is done based on the ARC Processor Compatible ECC Algorithm, which is designed specifically for the mission-critical applications. This ECC Algorithm is also implemented based on the modified Hamming code. For more information about the ARC Processor Compatible ECC Algorithm refer the ARC EV documentation.

Note When ARC Compatible ECC Engine is enabled, the `DMAX_MXIF_ECC_GRAN_TYPE` parameter can only be configured to 32-bit ECC Granularity (1) or 64-bit ECC Granularity (2).

3.5.5.3 User-Defined ECC Engine

The ECC checkbits generation architecture can be implemented with various SEC-DED hamming codes or other error correction and detection codes. The ECC generation and checker architecture chosen by the user can be different compared to the DW_ecc or ARC Compatible ECC, in such cases user can opt to use the User-Defined ECC Engine.

The DW_axi_dmac provides as wrapper files for User-Defined ECC Engine:

- DW_axi_dmac_usr_eccgen.v, and
- DW_axi_dmac_usr_eccmon.v

Table 3-4 DW_axi_dmac_usr_eccgen Port Definition

Pin Name	Payload Width (bits)	Description
datain	DATA_WIDTH	Input payload/data for which ECC is computed.
chkout	CHKB_WIDTH	ECC checkbits computed by using the user-defined ECC Engine

Table 3-5 DW_axi_dmac_usr_eccmon Port Definition

Pin Name	Payload Width (bits)	Description
datain	DATA_WIDTH	Input payload/data for which ECC is computed.
chkin	CHKB_WIDTH	ECC checkbits received for the datain.
err_detect	1	Active high Error Detect output, indicates that either single-bit or multi-bit error is detected.
err_multpl	1	Active high Multiple bit Error Detect output, indicates that multi-bit error is detected.
dataout	DATA_WIDTH	Output corrected (Single bit error) payload/data.
chkout	CHKB_WIDTH	ECC checkbits computed by using the user-defined ECC Engine
syndout	CHKB_WIDTH	Error Syndrome output for error logging purpose.

Inside these modules,

- User can insert User-Defined ECC Generator and Monitor logic in respective DW_axi_dmac_usr_eccgen and DW_axi_dmac_usr_eccmon modules, or
- User can instantiate the User-Defined ECC Generator and Monitor module in respective DW_axi_dmac_usr_eccgen and DW_axi_dmac_usr_eccmon modules.

3.5.6 ECC Granularity

The ECC Generation is a combinational logic, as the payload size increases for ECC computation it becomes difficult to meet the higher timing requirements. ECC Granularity feature improves the timing characteristics of the ECC Generator logic.

The ECC Generator supports following combination of ECC Granularities for different payloads which are multiple 32-bits shown in the Table 3-6.

Table 3-6 ECC Payload and ECC Checkbits with Different ECC Granularity (Multiple 32 Bits)

Payload Width	ECC Granularity		
	Full Width DMAX_MXIF_ECC_GRAN_TYPE = 0	32 DMAX_MXIF_ECC_GRAN_TYPE = 1	64 DMAX_MXIF_ECC_GRAN_TYPE = 2
32	7	7	7*

Payload Width	ECC Granularity		
	Full Width DMAX_MXIF_ECC_GRAN_TYPE = 0	32 DMAX_MXIF_ECC_GRAN_TYPE = 1	64 DMAX_MXIF_ECC_GRAN_TYPE = 2
64	8	2x7	8
128	9	4x7	2x8
256	10	8x7	4x8
512	11	16x7	8x8

* - indicates that if the payload width is less than Granularity (that is, 64 in above example), then ECC is generated with Full width granularity.

The ECC Generator supports following combination of ECC Granularities for different payloads (General) shown in Table 3-7.

Table 3-7 ECC Payload and ECC Checkbits with Different ECC Granularity (General)

Payload Width	ECC Granularity		
	Full Width DMAX_MXIF_ECC_GRAN_TYPE = 0	32 DMAX_MXIF_ECC_GRAN_TYPE = 1	64 DMAX_MXIF_ECC_GRAN_TYPE = 2
8	5	5*	5*
16	6	6*	6*
24	6	6*	6*
32	7	7	7*
40	7	2x7$	7*
58	8	2x7$	8*
64	8	2x7	8
96	8	3x7	2x8$
128	9	4x7$	2x8
192	9	6x7	3x8
256	10	8x7	4x8
384	10	12x7	6x8
512	11	16x7	8x8

* - indicates that, the payload width is less than Granularity, then ECC is generated with Full width granularity.

$ - indicates that, the payload width is divisible by Granularity more than once, but not multiple of Granularity.

In this case, ECC is computed for each whole Granularity width payload bits. The rest of the bits (which is not multiple of granularity) are prefixed with 0's to pack it to Granularity width and then ECC is computed.

For example,

- Payload Width = 58 and Granularity = 32

  For the lower 32-bits ECC Computed. For the upper 26-bits, zeros are appended to make it 32-bit width and then ECC is computed. The total ECC checkbits required are 2x7 = 14 bits.

- Payload Width = 96 and Granularity = 64

  For the lower 64-bits ECC Computed. For the upper 32-bits, zeros are appended to make it 64-bit width and then ECC is computed. The total ECC checkbits required are 2x8 = 16 bits.

3.5.7 Parity Mode

3.5.7.1 ARC Compatibility Parity Mode

The ARC Compatibility Parity mode is provided to align the Parity Generation implementation with ARC Processor Parity implementation. This Parity mode provides the interoperability between the ARC Processor's and Synopsys DesignWare AXI DMA Controller with Mission Critical features.

Table 3-8 provides details of the Parity computation in the case of ARC Compatibility Parity Mode.

Table 3-8 AXI Interface Parity Generator – ARC Compatibility Parity Mode

S.No.	AXI Channel Payload	ARC Compatibility Parity Mode
1	ARADDR/AWADDR	The Parity Generator computes the ARADDR/AWADDR parity per 8-bits. **Note**: If the payload is not multiple of 8, then zeros are appended to make multiple of 8-bits.
2	AR/AW Channel Control (ARCTRL/AWCTRL)	The Parity Generator computes the Parity for AR/AW Channel Control signals as follows: ■ [0] – Parity bit for ARID/AWID ■ [1] – Parity bit for ARLEN/AWLEN ■ [2] – Parity bit for {ARBURST, ARLOCK, ARSIZE}/ {AWBURST, AWLOCK, AWSIZE} ■ [3] – Parity bit for {ARCACHE, ARPROT}/ {AWCACHE, AWPROT} ■ [4] – Parity bit for ARQOS/AWQOS
3	WDATA	The Parity Generator computes the WDATA parity per 8-bits.

S.No.	AXI Channel Payload	ARC Compatibility Parity Mode
4	W Channel Control (WCTRL)	The Parity Generator computes the Parity for W Channel Control signals as follows: For AXI3, the parity generated is as follows: ■ [0] – Parity bit for WID ■ [1] – Parity bit for WSTRB ■ [2] – Parity bit for WLAST For AXI4, the parity generated is as follows: ■ [0] – Parity bit for WSTRB ■ [1] – Parity bit for WLAST
5	RDATA	The Parity Generator computes the WDATA parity per 8-bits.
6	R Channel Control (RCTRL)	The Parity Generator computes the Parity for R Channel Control signals as follows: ■ [0] – Parity bit for RID ■ [1] – Parity bit for RRESP and RLAST
7	B Channel Control (BCTRL)	The Parity Generator computes the Parity for B Channel Control signals as follows: ■ [0] – Parity bit for BID ■ [1] – Parity bit for BRESP

Note In ARC Compatibility mode, only 1-bit parity is computed for ARID, AWID, WID, RID, BID, and WSTRB irrespective of whether their width is:

- Less than or equal to 8 bits, or
- More than 8 bits.

3.5.8 Diagnostic Support for Error Management for AXI Manager Interface (with ECC and Parity Protection Feature)

3.5.8.1 AXI Channel Parity/ECC Error Interrupts

The Parity/ECC Monitor generates the ECC Protection interrupts to indicate the correctable and uncorrectable error has been detected.

ECC Protection – AXI Manager (y) (z) Channel Correctable Error Interrupt

The ECC Protection – AXI Manager (y) (z) Channel Correctable Error interrupt is generated, in the case where a single bit error has been detected on the data and it is corrected with the help of ECC engine.

The following ECC Protection – (z) Channel Correctable Error Interrupt will be generated:

- AXI Manager (y) – (z) Channel has a Correctable Error Interrupt (MXIF(y)_(z)CH_PROT_CorrERR_IntStat). The respective interrupt bit is added in the Common Interrupt registers:
- DMAC_COMMONREG_INTSTATUSREG

- DMAC_COMMONREG_INTSTATUS_ENABLEREG
- DMAC_COMMONREG_INTSIGNAL_ENABLEREG
- MAC_COMMONREG_INTCLEARREG

After this interrupt is asserted, the software

- Reads the DMAC_COMMONREG_INTSTATUSREG
- Executes the necessary interrupt service
- Clears the respective interrupt through the DMAC_COMMONREG_INTCLEARREG register

Where y = 1 to DMAX_NUM_MASTER_IF

z = R/B

Note AXI Interface Parity Protection – Parity protection will not have correctable interrupt.

Parity/ECC Protection – AXI Manager (y) (z) Channel Uncorrectable Error Interrupt

The Parity/ECC Protection AXI Manager (y) (z) Channel Uncorrectable Error interrupt is generated either when ECC engine detects a 2-bit error or when Parity engine detects an error.

The following Parity/ECC Protection – (z) Channel Uncorrectable Error Interrupt is generated:

- AXI Manager (y) – (z) channel has an Uncorrectable Error Interrupt (MXIF(y)_(z)CH_PROT_UnCorrERR_IntStat). The respective interrupt bit is added in the Common Interrupt registers:
- DMAC_COMMONREG_INTSTATUSREG
- DMAC_COMMONREG_INTSTATUS_ENABLEREG
- DMAC_COMMONREG_INTSIGNAL_ENABLEREG
- DMAC_COMMONREG_INTCLEARREG

After this interrupt is asserted, the software

- Reads the DMAC_COMMONREG_INTSTATUSREG register
- Executes the necessary interrupt service routine
- Clears the respective interrupt through the DMAC_COMMONREG_INTCLEARREG register

Where y = 1 to DMAX_NUM_MASTER_IF

z = R/ B

Parity/ECC Protection – Error Interrupt Signal

DW_axi_dmac adds a separate set of Parity/ECC Protection Error signals at the top level. These signals are level signals and are de-asserted based on the clearing of respective interrupt through the Interrupt clear registers DMAC_COMMONREG_INTCLEARREG.

The number of Parity/ECC Protection Error interrupt signals available on the top-level port depends on the DMAX_INTR_IO_TYPE parameter.

When DMAX_INTR_IO_TYPE=COMBINED_ONLY (0):

- Combined – ECC Protection Correctable Error (Combines ECC Protection Correctable Error from all AXI Manager Interfaces all Channels - Read Data and Write Response).
- Combined – Parity/ECC Protection Uncorrectable Error (Combines Parity/ECC Protection Uncorrectable Error from all AXI Manager Interface's all Channels - Read Data and Write Response).

When `DMAX_INTR_IO_TYPE= CHANNEL_AND_COMMONREG (1)`:

- AXI Manager Interface (y) Read Data Channel - ECC Protection Correctable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Write Response Channel - ECC Protection Correctable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Read Data Channel – Parity/ECC Protection Uncorrectable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Write Response Channel – Parity/ECC Protection Uncorrectable Error (DMAX_NUM_MASTER_IF bits) Where y = 1 to `DMAX_NUM_MASTER_IF`.

When `DMAX_INTR_IO_TYPE= ALL_INTERRUPT_OUTPUTS (2)`:

- Combined – ECC Protection Correctable Error
- Combined – Parity/ECC Protection Uncorrectable Error
- AXI Manager Interface (y) Read Data Channel - ECC Protection Correctable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Write Response Channel - ECC Protection Correctable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Read Data Channel – Parity/ECC Protection Uncorrectable Error (`DMAX_NUM_MASTER_IF` bits)
- AXI Manager Interface (y) Write Response Channel – Parity/ECC Protection Uncorrectable Error (`DMAX_NUM_MASTER_IF` bits)

Safe State Management

Irrespective of Correctable or Uncorrectable error type, the DW_dmac controller does not stop the AXI transfer. It also does not disable the DMA channel. However, if a multi-bit uncorrectable error on the RRESP signal or the BRESP signal results in the AXI error, then the DW_dmac controller might disable the DMA channel.

After detection of an ECC error, the Mission Critical Management software can do one of the following:

- Disable the DMA Channel (soft reset of the channel).
- Soft reset the entire DMA Controller.
- Hard reset the entire DMA Controller.

3.5.8.2 AXI Channel ECC Error Counter

The ECC Protection Control and status register `DMAC_ECCPROT_CTLSTATUSREG` is provided to support the enhanced error management capabilities. This additional diagnostic feature can be enabled through the `DMAX_ECC_DIAG_EN` parameter. The `DMAC_ECCPROT_CTLSTATUSREG` register,

- Provides access to all AXI Manager Interface related to Read Data Channel and Write Response Channel ECC Correctable and Uncorrectable counters.

- Provides access to AXI Manager Interface R/B Channel related to ECC counters using the control fields `INTFC_TYPE` and `INTFC_NO`.
- Provides the capability to reset all the ECC counters without resetting any other functionality.

The ECC correctable error counter is incremented whenever single-bit error is detected. The ECC uncorrectable error counter is incremented whenever multi-bit error detected by the ECC engine.

For more information about this register, see the `DMAC_ECCPROT_CTLSTATUSREG` in the "Register Descriptions" chapter.

Note The ECC AXI Manager Interface correctable/uncorrectable error counter register will not be cleared with the Soft Reset or when Channel is disabled. It can be cleared only with the ECC Error Count Reset, that is, `DMAC_COMMON_ECCCTLSTATUSREG.ECC_ERR_COUNT_RST`.

3.5.9 AXI Interface VALID and READY Parity Generator and Checker

The AXI Interface handshake signals `VALID` and `READY` are protected with the parity signal. This feature is a configurable option enabled through the parameter `DMAX_MXIF_READY_VALID_PAR_EN`. The parity type, that is, Odd or Even parity is configurable through the parameter `DMAX_MXIF_READY_VALID_PAR_EN`.

The VALID/READY Parity Generator computes the parity for the `VALID/READY` signal and drives the same along with the non-standard sideband signal.

This logic block computes the parity for the following `VALID/READY` signals depending on which AXI Channel it is used.

- AXI Manager Interface: `ARVALID/AWVALID/WVALID/RREADY/BREADY`

The VALID/READY Parity Checker validates the parity of the VALID/READY signal. The VALID/READY signal parity is received through the respective non-standard sideband signal.

If the VALID/READY Parity checker validation fails, then VALID/READY Parity Check Error Interrupt is asserted.

This block validates the parity for the following `VALID/READY` signals, depending on which AXI Channel it is used.

- AXI Manager Interface: `ARREADY/AWREADY/WREADY/RVALID/BVALID`

Note If the VALID/READY Parity check fails, then respective payload will be forwarded based on the active state of VALID/READY signal. But VALID/READY Parity Check Error Interrupt is asserted.

VALID and READY Parity Protection – AXI Manager (y) (z) Channel VALID and READY Parity Error Interrupt

The VALID and READY Parity Protection AXI Manager (y) (z) Channel VALID and READY Parity Error interrupt is generated when READY or VALID Parity checker validation fails.

The following VALID and READY Parity Protection – (z) Channel VALID and READY Parity Error Interrupt is generated:

- AXI Manager (y) – (z) channel has VALID or READY Parity Error Interrupt (MXIF(y)_(z)CH_PROT_(z)VALID/READY_ParErr _IntStat). The respective interrupt bit is added in the Common Interrupt registers:
- `DMAC_COMMONREG_INTSTATUSREG`
- `DMAC_COMMONREG_INTSTATUS_ENABLEREG`
- `DMAC_COMMONREG_INTSIGNAL_ENABLEREG`
- `DMAC_COMMONREG_INTCLEARREG`

After this interrupt is asserted, the software

- Reads the `DMAC_COMMONREG_INTSTATUSREG` register
- Executes the necessary interrupt service routine
- Clears the respective interrupt through the `DMAC_COMMONREG_INTCLEARREG` register

Where y = 1 to `DMAX_NUM_MASTER_IF`

z = AR/AW/W/R/B

VALID and READY Parity Protection – Error Interrupt Signal

DW_axi_dmac adds a separate set of VALID and READY Parity Protection Error signals at the top level. These signals are level signals and are de-asserted based on the clearing of respective interrupt through the Interrupt clear registers `DMAC_COMMONREG_INTCLEARREG`.

The number of VALID and READY Parity Protection Error interrupt signals available on the top-level port depends on the `DMAX_INTR_IO_TYPE` parameter.

When `DMAX_INTR_IO_TYPE=COMBINED_ONLY (0)`:

- Combined – VALID and READY Parity Protection Error (Combines VALID and READY Parity Protection Error from all AXI Manager Interfaces all Channels).

When `DMAX_INTR_IO_TYPE= CHANNEL_AND_COMMONREG (1)`:

- AXI Manager Interface (y) Read Address Channel - READY Parity Protection Error
- AXI Manager Interface (y) Write Address Channel - READY Parity Protection Error
- AXI Manager Interface (y) Write Data Channel - READY Parity Protection Error
- AXI Manager Interface (y) Read Data Channel - VALID Parity Protection Error
- AXI Manager Interface (y) Write Response Channel - VALID Parity Protection Error

When `DMAX_INTR_IO_TYPE= ALL_INTERRUPT_OUTPUTS (2)`:

- Combined – VALID and READY Parity Protection Error (Combines VALID and READY Parity Protection Error from all AXI Manager Interfaces all Channels).
- AXI Manager Interface (y) Read Address Channel - READY Parity Protection Error
- AXI Manager Interface (y) Write Address Channel - READY Parity Protection Error
- AXI Manager Interface (y) Write Data Channel - READY Parity Protection Error
- AXI Manager Interface (y) Read Data Channel - VALID Parity Protection Error
- AXI Manager Interface (y) Write Response Channel - VALID Parity Protection Error

3.5.10 Enabling ECC and Parity Protection Feature for AXI Manager Interface

To enable ECC Protection feature for AXI Manager Interface, click the “Include Safety Protection feature on the AXI Manager Interface?” check-box in the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.6 Lock Step Protection Feature

This section covers the following topics:

- "Overview of Lock Step Protection Feature" on page 150
- "Description of the Lock Step Protection Feature" on page 150
- "Diagnostic Support for Error Management" on page 154
- "Enabling the Lock Step Protection Feature" on page 155
- "Signals Related to the Lock Step Protection Feature" on page 156

3.6.1 Overview of Lock Step Protection Feature

The Lock Step feature is implemented for the mission-critical applications such as automotive; where safety and reliability of the system is very important. The Lock Step Protection feature helps to implement the DW_axi_dmac design with more safety and reliability against unfavorable operating conditions with Single Event Upset (SEU), and extreme EMI/EMC.

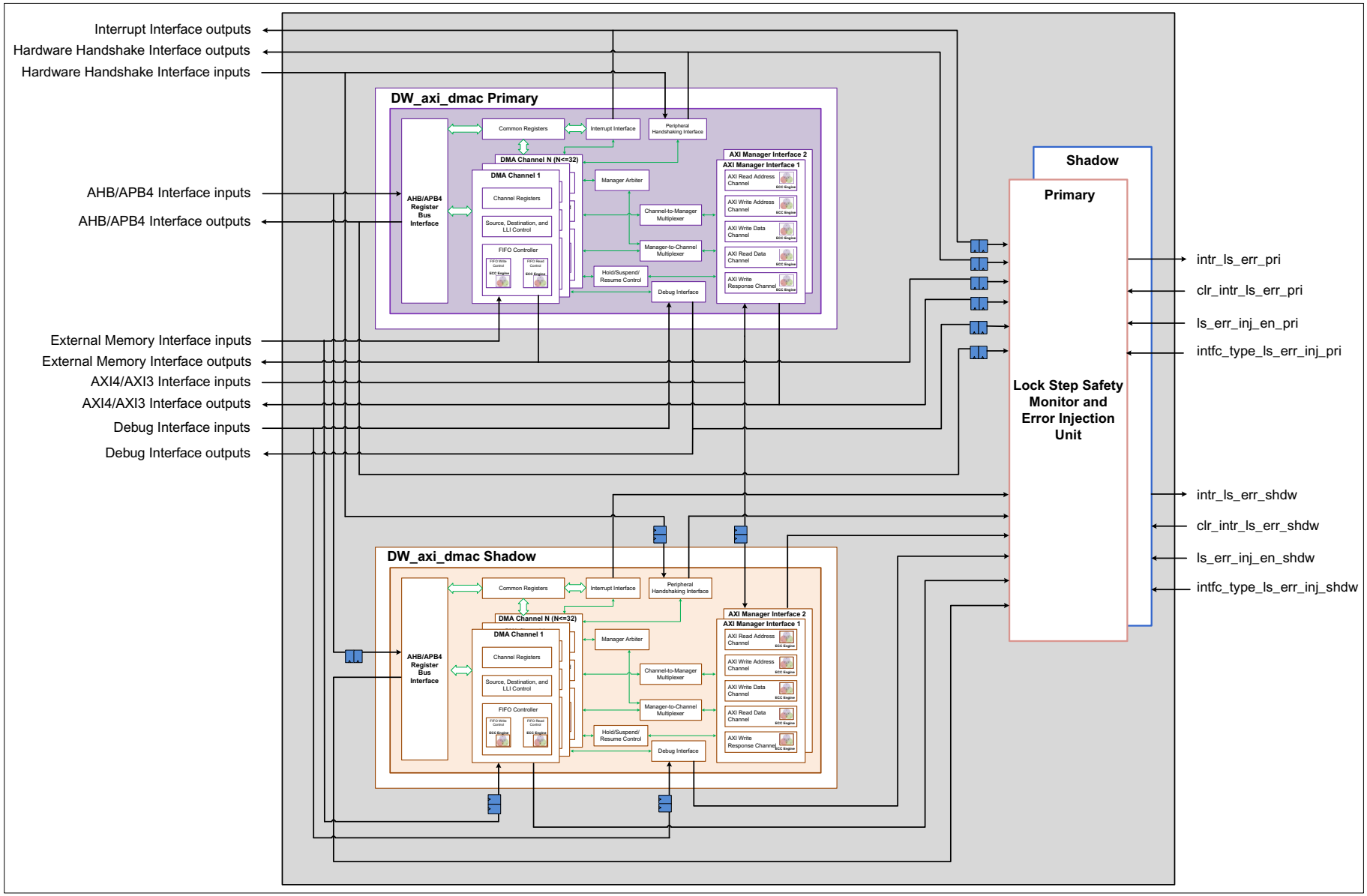
3.6.2 Description of the Lock Step Protection Feature

DW_axi_dmac implements the Lock Step feature by creating a duplicate instance of DW_axi_dmac. The duplication is done for the major part of DW_axi_dmac except:

- The Channel FIFO Memory
- Any block that needs to be kept outside the duplication due to asynchronous clock such as Asynchronous Hardware handshake related logic.

Figure 3-4 describes the typical configuration where the Lock Step Protection feature is implemented in DW_axi_dmac.

Figure 3-4 Lock Step Protection Feature – Typical Configuration



The main DW_axi_dmac instance is called the Primary DMA Controller, and redundant instance is called the Shadow/Secondary DMA Controller. The Primary and Shadow DMA Controller instances operation is separated two cycles delay, where shadow instance operation delayed by two cycles with respect to primary instance using the delay flip-flops.

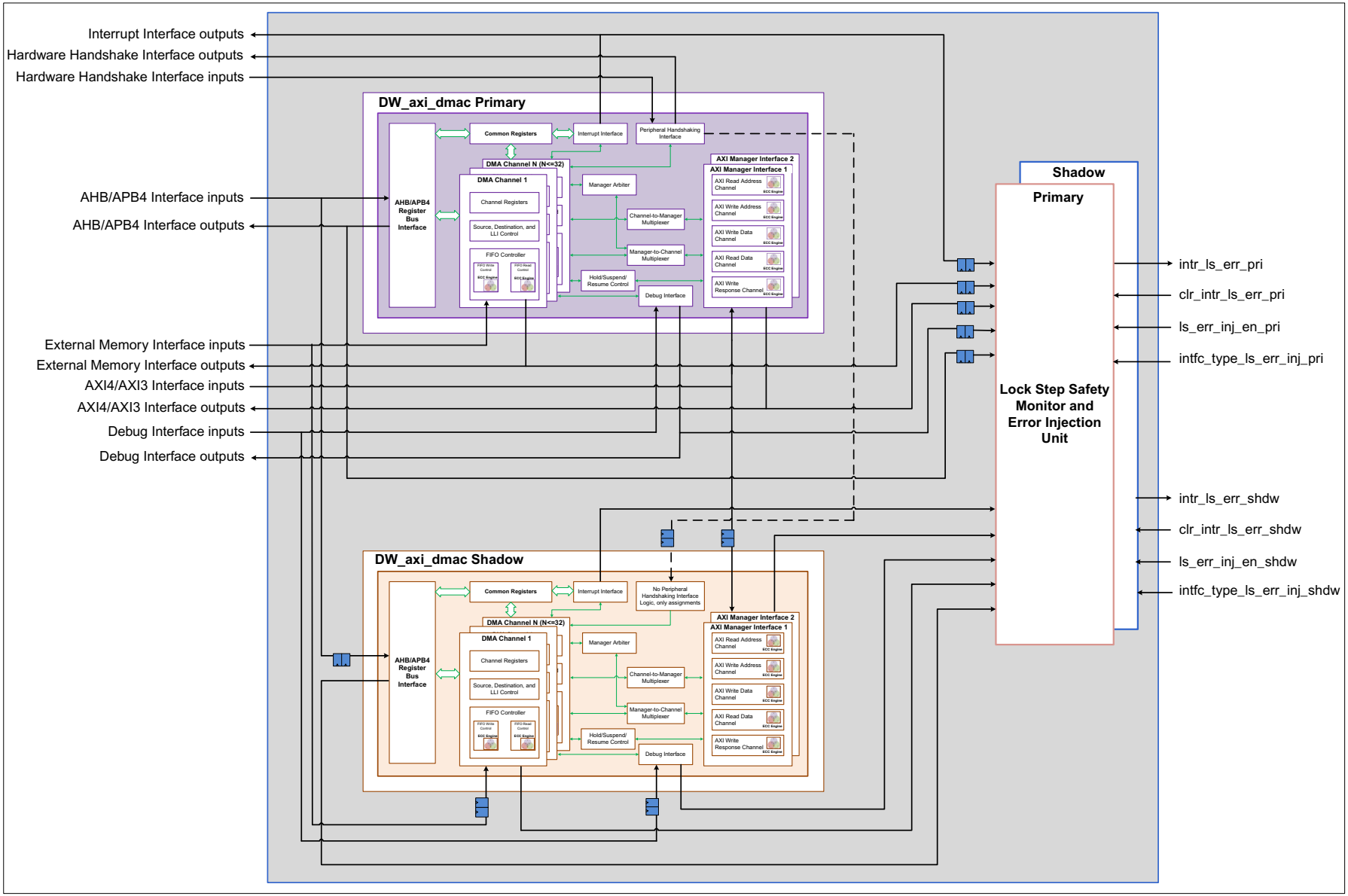
- Inputs of the Shadow DMA Controller are delayed by two clock cycles and
- Outputs of the Primary DMA Controller connected to the safety monitor are delayed by two clock cycles

Due to the delayed operation of the Primary and Shadow DMA Controllers, this feature is also called as the delayed Lock Step Protection.

The Lock Step Safety monitor tracks the error that can occur during the operation of the DMA Controller. See "Diagnostic Support for Error Management" on page 154.

Figure 3-5 on page 153 describes the typical configuration where the Lock Step Protection feature is implemented. In this example, all blocks except the Channel FIFO Memory and Hardware Handshake interface module are duplicated.

Figure 3-5 Lock Step Protection Feature – Typical Configuration and Asynchronous Hardware Handshake Enable



Dotted line in Figure 3-5 on page 153 refers to the internal signal connection (related to Asynchronous Handshake Module of the Primary DW_axi_dmac instance) to the Shadow instance of DW_axi_dmac.

This configuration is the same as a typical configuration shown in the Figure 3-4 on page 151 except that Asynchronous Hardware Handshake feature is enabled. In this configuration, the logic related to the Asynchronous Hardware handshake feature is not duplicated rather it is retained only in the Primary DMA Controller instance. The Primary DMA Controller instance supplies all hardware handshake logic related inputs required by the DMA Channel logic of the Shadow DMA Controller instance. The operation of this configuration is the same as that of the typical configuration.

Note The Lock Step Protection feature is supported only for the configuration with all clocks synchronous or all clocks synchronous when Asynchronous Hardware Handshake feature is enabled.

3.6.3 Diagnostic Support for Error Management

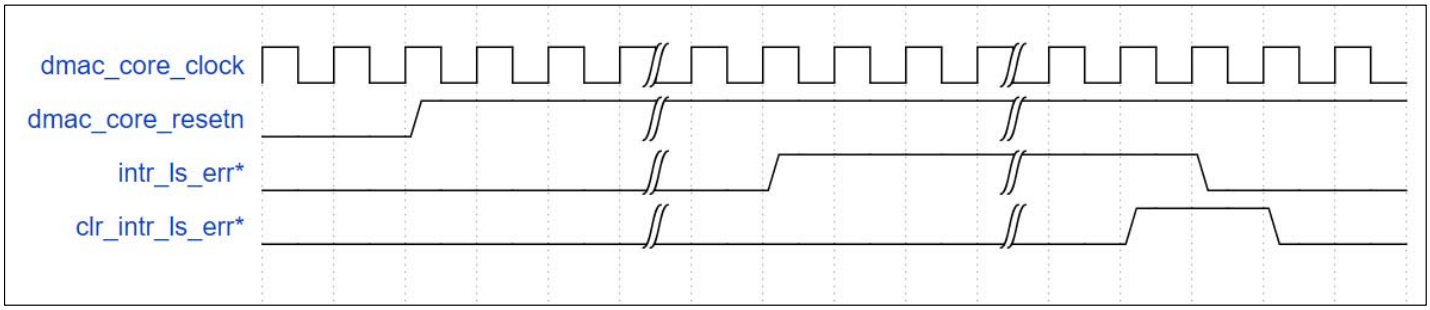
This section provides more information about the diagnostic features implemented in DW_axi_dmac as part of the Lock Step Protection feature.

3.6.3.1 Lock Step Protection – Safety Monitor

The Lock Step Safety Monitor is a monitor that checks the coherency of operation between the Primary and the Shadow DMA Controller instances. If the comparison fails and error occurs, then it is notified through a dedicated error signal (`intr_ls_err*`). This `intr_ls_err*` error signal is a level sensitive signal, and it can be cleared only through the Error Interrupt Clear signal (`clr_intr_ls_err*`).

Figure 3-6 is a timing diagram of the Lock Step Protection safety monitor signal.

Figure 3-6 Lock Step Protection - Safety Monitor Timing Diagram



Where, * - pri or shdw

The Safety Monitor Comparator logic is also duplicated for added protection on the safety monitor logic.

3.6.3.2 Lock Step Protection – Error Injection

The Lock Step Protection also supports Hardware Error Injection feature on the Lock Step Error signal (`intr_ls_err*`) through the level sensitive Error Injection Enable signal (`ls_err_inj_*_en`). This error injection introduces the artificial error into the Lock Step Error signal during the normal operating

conditions. This enables system level validation of the Lock step error condition even without having the design under unfavorable operating conditions with Single Event Upset (SEU), and extreme EMI/EMC.

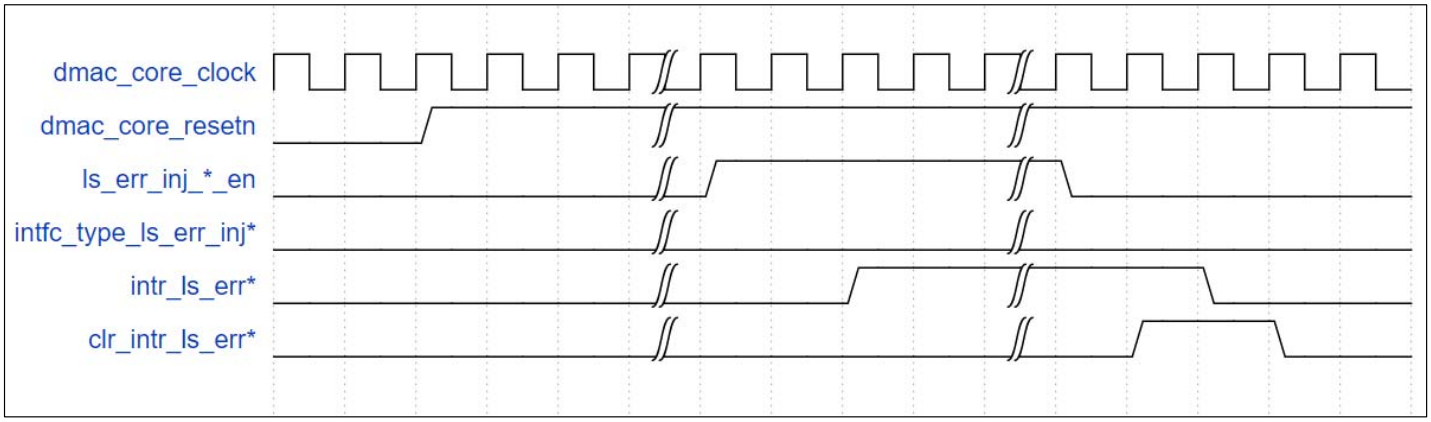
When the error injection enable signal (`ls_err_inj_*_en`) is asserted, error is injected to all the bits which are inputs to the comparator at once. This will result in the comparator mismatch for all the bits and a lock step error is triggered. There is a flexibility to choose between the primary or shadow inputs to the lock step comparator.

- When `intfc_type_ls_err_inj*` is 0, the error is inserted for all the primary inputs.
- When `intfc_type_ls_err_inj*` is 1, the error is inserted for all the shadow inputs.

The error injection triggers all the comparator outputs at once to indicate the error and ensures that all the comparators are working fine as expected.

Figure 3-7 shows the hardware error injection error waveform for the lock step monitors present in the IP. Primary or shadow inputs of the monitor are corrupted when the error injection is enabled and the corresponding interface type is chosen. The lock step error interrupt can be cleared using the respective clear input (`clr_intr_ls_err*`) as described below.

Figure 3-7 Lock Step Protection - Error Injection Timing Diagram



Where, * - pri or shdw

Note

- When `DMAX_ADV_LS_ERR_CAP = 0`, the `intr_ls_err*` is asserted only after single cycle delay with respect to `ls_err_inj_*_en`.
- When `DMAX_ADV_LS_ERR_CAP = 1`, the `intr_ls_err*` is asserted after 2 cycles delay with respect to `ls_err_inj_*_en` as shown in Figure 3-7.

3.6.4 Enabling the Lock Step Protection Feature

To enable the Lock Step Protection Feature, click the “Include Lock Step Protection feature?” field using the **Safety Feature Configuration** tab during the **Specify Configuration** Activity in coreConsultant.

3.6.5 Signals Related to the Lock Step Protection Feature

The following signals are related to the Lock Step Protection feature:

Lock Step Protection Signals

- `dmac_core_clock`
- `dmac_core_resetn`
- `dmac_shdw_clock`
- `dmac_shdw_resetn`
- `dmac_lsmon_clock`
- `dmac_lsmon_resetn`

Interrupt Signals for Lock Step Protection Feature

- `intr_ls_err_pri`
- `intr_ls_err_shdw`
- `clr_intr_ls_err_pri`
- `clr_intr_ls_err_shdw`
- `ls_err_inj_pri_en`
- `ls_err_inj_shdw_en`
- `intfc_type_ls_err_inj_pri`
- `intfc_type_ls_err_inj_shdw`

For more information about these signals, see the “Signals Description” chapter.

4

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

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

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

- "Top Level Parameters" on page 158
- "Manager Interface Configuration Parameters" on page 163
- "Register Bus Interface Configuration Parameters" on page 166
- "Clocking Parameters" on page 167
- "Safety Feature Configuration Parameters" on page 171
- "Low Power Configuration Parameters" on page 176
- "Channel x Configuration Parameters" on page 178

4.1 Top Level Parameters

Table 4-1 Top Level Parameters

Label	Description
Top Level Parameters	
AXI DMAC ID	A 32-bit value that is hardwired and read back by a read to the DW_axi_dmac ID Register (DMAC_IDReg). **Values:** 0x0, ..., 0xffffffff **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

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 Register Bus Interface?	By default, this option creates a register bus interface status indication (Busy/Idle) signals on the I/O, which is synchronous to the register bus 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

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_hold_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
Configure the Arbiter Type?	This parameter allows user to configure the arbitration type: ■ Dynamic Priority Arbitration with Fair-Among-Equals ■ Dynamic Priority Arbitration with Round Robin User can configure Dynamic Priority as first-tier arbitration with option to have second-tier arbitration as "Fair Among Equals" or "Round Robin". **Values:** ■ Dynamic Priority Arbitration with Fair-Among-Equals (0) ■ Dynamic Priority Arbitration with Round Robin (1) **Default Value:** Dynamic Priority Arbitration with Fair-Among-Equals **Enabled:** DMAX_NUM_CHANNELS > 1 && <DWC-AXI-DMAC-Plus feature authorize> == 1 **Parameter Name:** DMAX_ARB_TYPE
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 Manager interface. Enabling this feature helps to improve the QoR for the implemented design. **Values:** ■ No (0) ■ Yes (1) **Default Value:** ((DMAX_NUM_CHANNELS>8) && (DMAX_ARB_TYPE == 0)) ? 1 : 0 **Enabled:** (DMAX_NUM_CHANNELS>8) && (DMAX_ARB_TYPE == 0) **Parameter Name:** DMAX_MULT_ARB_EN

Label	Description
Include Channel Terminate Feature?	When set to Yes (1), enables logic to support Channel Terminate feature in DW_axi_dmac. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** Always **Parameter Name:** DMAX_CH_ABORT_EN
Include Transfer Completion Indication Signal on Manager Interface?	When set to Yes (1), enables the additional handshaking signal last_write_mi on all AXI manager 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
Use External memory?	This parameter is used for selecting Channel FIFO Memory and Unique ID Memory - internal or external or Top Level of the DW_axi_dmac. All channels share this parameter. The default value is 0, then internal channel memory interface scheme is used. The internal channel FIFO/Unique ID Memory uses DesignWare flip-flop based memory. When external channel memory interface scheme is used, SoC designer needs to implement the memories based on memory size and memory library external to DW_axi_dmac. When Top Level channel memory interface scheme is used, then internal channel FIFO/Unique ID Memory uses DesignWare flip-flop based memory and instantiates them at the top level of the DW_axi_dmac. **Values:** ■ Internal (0) ■ External (1) ■ TopLevel (2) **Default Value:** Internal **Enabled:** Always **Parameter Name:** DMAX_CH_MEM_EXT
External Memory Register Output	This parameter decides if to add pipeline register after channel FIFO/UID memory. It is only valid when external memory is enabled. If enabled, pipeline register is added in DMAC IP and memory read data timing will be improved. **Values:** ■ No (0) ■ Yes (1) **Default Value:** DMAX_ECC_MEMIF_EN == 1 ? 1 : 0 **Enabled:** (DMAX_CH_MEM_EXT>= 1) && (DMAX_ECC_MEMIF_EN == 0) **Parameter Name:** DMAX_CH_MEM_REGOUT

Label	Description
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

4.2 Manager Interface Configuration Parameters

Table 4-2 Manager Interface Configuration Parameters

Label	Description
Manager Interface Configuration	
DW_axi_dmac Manager Interface Protocol	The protocol used for the AXI Manager 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 Manager Interfaces	Creates the specified number of AXI manager interfaces. A channel source or destination device can be programmed to be on any of the configured AXI layers attached to the AXI manager interface. This setting determines if AXI manager 2 interface signal set is present on the I/O. AXI manager interface signals are always present. **Values:** 1, 2 **Default Value:** 1 **Enabled:** Always **Parameter Name:** DMAX_NUM_MASTER_IF
Manager 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 Manager 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 Manager 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

Label	Description
Statically Configure Endian Scheme of Manager Interfaces?	The endian scheme of the DW_axi_dmac manager 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 manager interfaces. ■ For the dynamic case, there is an individual pin for each of the AXI manager 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 Manager 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 Manager interface. LLI access on each AXI Manager interface can be independently configured to support Big Endian scheme (BE-8) based on the endian scheme selected for that particular manager interface or Little Endian scheme irrespective of the endian scheme selected for that particular manager 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
Manager Interface Endian Format?	DMAX_ENDIAN_FORMAT_MSTIF values for different AXI manager 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

Label	Description
Manager Interface Adddress Bus Width	AXI Manager 1 interface address bus width. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** DMAX_M_ADDR_WIDTH
Manager Interface Data Bus Width	AXI Manager interface data bus width. **Values:** 32, 64, 128, 256, 512, 1024 **Default Value:** 32 **Enabled:** Always **Parameter Name:** DMAX_M_DATA_WIDTH
Manager Interface ID Bus Width	AXI manager 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
Manager Interface Burst Length Width	AXI manager 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

4.3 Register Bus Interface Configuration Parameters

Table 4-3 Register Bus Interface Configuration Parameters

Label	Description
Register Bus Interface Configuration	
DW_axi_dmac Register Bus Interface Protocol	The protocol used for Register Bus Interface. *Only AHB and APB4 is supported in this version.* **Values:** ■ AHB (0) ■ APB4 (2) **Default Value:** AHB **Enabled:** Always **Parameter Name:** DMAX_SLVIF_MODE
Synchronize Interrupt Outputs to Register Bus 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 register bus 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
Register Bus Interface Data Bus Width	Specifies the data bus width for the AHB/APB4 register bus interface. **Values:** 32, 64 **Default Value:** 32 **Enabled:** DMAX_SLVIF_MODE == 0 **Parameter Name:** DMAX_S_DATA_WIDTH

4.4 Clocking Parameters

Table 4-4 Clocking Parameters

Label	Description
Synchronization options	
DW_axi_dmac Register Bus Interface Clocking Mode	Selects the relationship between the register bus interface clock (AHB/APB4) and the Core clock. ■ 0: Register Bus interface clock is synchronous to the core clock. ■ 1: Register Bus interface clock is asynchronous to the core clock. **Values:** ■ Synchronous (0) ■ Asynchronous (1) **Default Value:** Synchronous **Enabled:** DMAX_LS_PROT_EN==0 **Parameter Name:** DMAX_SLVIF_CLOCK_MODE
Subordinate to Core Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Register Bus Interface clock domain to the DW_axi_dmac Core Clock domain. ■ 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:** 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_SLVIF_CLOCK_MODE==1 **Parameter Name:** DMAX_S_2_C_SYNC_DEPTH
Core to Subordinate Synchronization Depth	Defines the number of synchronization register stages for signals passing from the DW_axi_dmac Core Clock Domain to the Register Bus Interface clock domain. ■ 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:** 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_SLVIF_CLOCK_MODE==1 **Parameter Name:** DMAX_C_2_S_SYNC_DEPTH

Label	Description
DW_axi_dmac Manager i Interface Clocking Mode (for i = 1; i <= DMAX_NUM_MASTER_IF)	Selects the relationship between the Manager i Interface clock (aclk_m(i)) and the Core clock. ■ 0: Manager i interface clock is synchronous to the core clock. ■ 1: Manager i interface clock is asynchronous to the core clock. **Values:** ■ Synchronous (0) ■ Asynchronous (1) **Default Value:** Synchronous **Enabled:** DMAX_LS_PROT_EN==0 **Parameter Name:** DMAX_MSTIF(i)_CLOCK_MODE
Manager 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 Manager i clock domain to the DW_axi_dmac Core Clock domain. ■ 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:** 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_MSTIF(i)_CLOCK_MODE == 1 **Parameter Name:** DMAX_M(i)_2_C_SYNC_DEPTH
Core to Manager 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 Manager i clock domain. ■ 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:** 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_MSTIF(i)_CLOCK_MODE == 1 **Parameter Name:** DMAX_C_2_M(i)_SYNC_DEPTH
Synchronizer FIFO Depths of Manager Interface	
Manager Interface Read/Write Address Channel FIFO Depth	AXI manager 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

Label	Description
Manager Interface Read/Write Data Channel FIFO Depth	AXI manager interface Read/Write Data Channel FIFO depth. Setting appropriate value based on the system requirement allows some logic optimization of the implementation. When channel FIFO selects external scheme, it is suggested to set this parameter to 8. **Values:** 4, 8 **Default Value:** (DMAX_CH_MEM_EXT==1) ? 8 : 4 **Enabled:** DMAX_MSTIF1_CLOCK_MODE == 1 || DMAX_MSTIF2_CLOCK_MODE == 1 || DMAX_CH_MEM_EXT == 1 **Parameter Name:** DMAX_M_DATA_FIFO_DEPTH
Manager Interface Write Response Channel FIFO Depth	AXI manager 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
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

Label	Description
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. ■ 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:** 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. ■ 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:** 2, 3, 4 **Default Value:** 2 **Enabled:** DMAX_ASYNC_HS_EN==1 **Parameter Name:** DMAX_C_2_HS_SYNC_DEPTH
Asynchronous Synchronization Options for DMA Handshake Interface (y) - 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

4.5 Safety Feature Configuration Parameters

Table 4-5 Safety Feature Configuration Parameters

Label	Description
Safety Feature	
Include AXI Safety Features?	This parameter allows user to Enable the Safety features. If enabled, this allows us to enable Safety Features: ■ Register Bus Interface Parity Feature ■ ECC Protection Feature of : ❑ Channel Memory Interface ❑ UID Memory Interface ❑ AXI Manager Interface ■ Lock Step Protection Feature **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** <DWC-AXI-DMAC-SAFETY feature authorize> == 1 **Parameter Name:** DMAX_SAFETY_FEATURE_EN
Interface Parity Protection Configuration	
Register Bus Interface Parity Check Enable?	Specifies whether or not parity check is enabled for data bus of the AHB/APB4 register bus interface. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_SAFETY_FEATURE_EN==1 **Parameter Name:** DMAX_PARITY_PROT_EN
Register Bus Interface Parity Mode?	Default value of the Parity Mode i.e. DMAC_PARITYREG.PARITY_MODE field. **Values:** ■ Even (0x0) ■ Odd (0x1) **Default Value:** Even **Enabled:** DMAX_PARITY_PROT_EN == 1 **Parameter Name:** DMAX_DEF_PARITY_MODE

Label	Description
The Parity Check is done after a Pipeline stage?	The Write Parity check can be performed when hwdata is ready or after a pipelining stage. (For input timing purpose). Due to this the hready_resp will also be delayed by one or two cycle. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** (DMAX_PARITY_PROT_EN == 1) && (DMAX_SLVIF_MODE == 0) **Parameter Name:** DMAX_HWDATA_PIPE_LINE
ECC and Parity Protection Feature Configuration	
Include ECC Protection feature?	This parameter allows user to include ECC/Parity Protection feature. If enabled, the ECC Protection feature can be included for: ■ Channel Memory Interface ■ UID Memory Interface If enabled, ECC/Parity Protection feature can be included for: ■ AXI Manager Interface Setting appropriate value based on the system requirement allows some logic optimization of the implementation. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DMAX_SAFETY_FEATURE_EN==1 **Parameter Name:** DMAX_ECC_PROT_EN
Include ECC Protection for the FIFO Memory Interface?	This parameter allows user to include or exclude ECC Protection feature for the FIFO Memory interface of all the channels. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** (DMAX_ECC_PROT_EN == 1) && (DMAX_CH_MEM_EXT >= 1) **Parameter Name:** DMAX_ECC_MEMIF_EN
Include ECC Protection for the UID Memory Interface?	This parameter allows user to include or exclude ECC Protection feature for the UID Memory interface of all the channels. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** (DMAX_ECC_PROT_EN == 1) && (DMAX_CH_MEM_EXT >= 1) && (DMAX_HAS_RUID_PARAM == 1) **Parameter Name:** DMAX_ECC_UIDMEMIF_EN

Label	Description
Error Injection Capability Enabled?	Enables the error injection capability in FIFO/UID memory interface. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DMAX_ECC_MEMIF_EN==1 || DMAX_ECC_UIDMEMIF_EN==1 **Parameter Name:** DMAX_ECC_ERR_INJ_EN
Include Safety Protection feature on the AXI Interface?	Enables the Safety feature on the AXI Manager Interface. Safety Features include: ■ Parity Protection Feature and/or ■ ECC Protection Feature **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DMAX_ECC_PROT_EN == 1 **Parameter Name:** DMAX_MXIF_ECC_PAR_EN
AXI Interface ECC and Parity Mode	Selects the AXI Interface ECC and Parity Mode. ■ 1: ECC for Address/Data and Control signals ■ 2: ECC for Address/Data and Parity for Control signals **Values:** ■ ECC - Address/Data and Control (1) ■ ECC - Address/Data and Parity - Control (2) **Default Value:** ECC - Address/Data and Control **Enabled:** DMAX_MXIF_ECC_PAR_EN == 1 **Parameter Name:** DMAX_MXIF_ECC_PAR_MODE
AXI Interface Parity Type?	Selects AXI Interface Parity type for data, address, or control signals ■ Even Parity ■ Odd Parity **Values:** ■ Even (0) ■ Odd (1) **Default Value:** Odd **Enabled:** DMAX_MXIF_ECC_PAR_EN == 1 && DMAX_MXIF_ECC_PAR_MODE == 2 **Parameter Name:** DMAX_MXIF_PAR_TYPE

Label	Description
AXI Interface ECC Mode?	Selects the ECC Engine used for ECC Generation and Monitoring for AXI Manager Interface. The following ECC Modes are supported: ■ DesignWare ECC Engine(0) ■ ARC Mode ECC Engine(1) ■ User Defined ECC Engine(2) Setting appropriate value based on the system requirement. **Values:** ■ DesignWare ECC Engine (0) ■ ARC Mode ECC Engine (1) ■ User Defined ECC Engine (2) **Default Value:** DesignWare ECC Engine **Enabled:** DMAX_ECC_PROT_EN==1 && (DMAX_ECC_MXIF_EN==1 || DMAX_MXIF_ECC_PAR_EN==1) **Parameter Name:** DMAX_MXIF_ECC_MODE
AXI Interface ECC Granularity Type?	AXI Interface ECC Resolution Type. This parameter selects the Granularity of ECC Checkbits generation. ■ 0: Full Width ECC Computation ■ 1: 32 bit Granularity ECC Computation ■ 2: 64 bit Granularity ECC Computation **Values:** ■ Full Width (0) ■ 32-Bit ECC (1) ■ 64-Bit ECC (2) **Default Value:** Full Width **Enabled:** DMAX_MXIF_ECC_PAR_EN == 1 **Parameter Name:** DMAX_MXIF_ECC_GRAN_TYPE
Include Safety Protection feature on the AXI Interface VALID and READY Handshake signal?	Enables AXI Interface Ready and Valid Parity feature **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_MXIF_ECC_PAR_EN==1 **Parameter Name:** DMAX_MXIF_READY_VALID_PAR_EN

Label	Description
AXI Interface VALID and READY handshake signal Parity type?	Selects AXI Interface Parity type for valid and ready signals ■ Even Parity ■ Odd Parity **Values:** ■ Even (0) ■ Odd (1) **Default Value:** Odd **Enabled:** DMAX_MXIF_READY_VALID_PAR_EN == 1 **Parameter Name:** DMAX_MXIF_READY_VALID_PAR_TYPE
Include ECC additional diagnostic features?	This parameter allows user to include or exclude ECC additional diagnostic features like access to correctable, and uncorrectable counters through the register DMAC_COMMONREG_ECCCTLSTATUSREG. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** DMAX_ECC_PROT_EN == 1 **Parameter Name:** DMAX_ECC_DIAG_EN
Lock Step Protection Feature Configuration	
Include Lock Step Protection feature?	This parameter allows user to include Lock Step Protection feature. If enabled, this allows us to duplicate the DW_axi_dmac instance to provide delayed Lock Step operation. **Note:** The Lock Step Protection feature duplicates the entire DW_axi_dmac to provide the Lock step operation. Hence this feature must be chosen based system level functional safety requirement. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** (DMAX_CH_MEM_EXT >= 1) && (DMAX_SAFETY_FEATURE_EN == 1) **Parameter Name:** DMAX_LS_PROT_EN
Include Lock Step Protection diagnostic features?	This parameter allows user to include or exclude Lock Step Protection feature diagnostic feature to include hardware error injection logic. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_LS_PROT_EN == 1 **Parameter Name:** DMAX_LS_DIAG_EN

4.6 Low Power Configuration Parameters

Table 4-6 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 Register Bus Interface?	When set to Yes (1), includes the logic that implements Context Sensitive Low Power feature in Register 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 Manager Interface?	Include or exclude logic that implements Context Sensitive Low Power feature in Manager Bus Interface. **Values:** ■ No (0) ■ Yes (1) **Default Value:** No **Enabled:** DMAX_CSLP_EN == 1 **Parameter Name:** DMAX_MXIF_CSLP_EN

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 Manager Interface Low Power Delay Counter	Defines the width of the AXI Manager 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 Manager Interface Low Power Delay Counter	Defines the default load value of the AXI Manager Interface Low Power Delay Counter. **Values:** 4, ..., 255 **Default Value:** 4 **Enabled:** DMAX_MXIF_CSLP_EN == 1 **Parameter Name:** DMAX_MXIF_LPDLY

4.7 Channel x Configuration Parameters

Table 4-7 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. For more information on the FIFO depth calculation, see "Calculating FIFO Depth" section in the user guide. **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

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 manager 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 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

Label	Description
Hardcode the Manager Interface Attached to the Source of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI manager 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:** ■ Manager 1 (0x0) ■ Manager 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 Manager Interface Attached to the Destination of Channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI manager 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:** ■ Manager 1 (0x0) ■ Manager 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 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) ■ Thirty-two Word (1024) **Default Value:** Word **Enabled:** Always **Parameter Name:** DMAX_CH(x)_STW

Label	Description
Hardcode Channel x 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) ■ Thirty-two Word (1024) **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

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
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

Label	Description
Hardcode the Manager Interface Attached to the LLP peripheral of channel x (for x = 1; x <= DMAX_NUM_CHANNELS)	Hardcode the AXI manager 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:** ■ Manager 1 (0x0) ■ Manager 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
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

Label	Description
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
Include AXI Unique ID Feature in AXI Read Channel of Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	This parameter allows user to include AXI Unique ID feature in the AXI Read Channel. If enabled, then AXI Read Address Channel will generate the AXI transfers with the unique AXI ID. This will allow Source memory to provide read data out of order. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** DMAX_CH(x)_RD_UID_EN
Number of AXI Read Channel Unique ID's supported for Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	This parameter allows user to select the number of AXI Read Channel Unique ID's supported. This parameter significantly affects the depth of the re-ordering buffer - hence area, so this feature and depth of the unique ID must be chosen carefully based on the requirement only for the required channel. **Values:** 2, 4, 8 **Default Value:** 2 **Enabled:** DMAX_CH(x)_RD_UID_EN==1 **Parameter Name:** DMAX_CH(x)_RD_UID
Include AXI Unique ID Feature in AXI Write Channel of Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	This parameter allows user to include AXI Unique ID feature in the AXI Write Channel. If enabled, then AXI Write Address and Data Channel will generate the AXI transfers with the unique AXI ID. This will allow Destination memory to provide write response out of order. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** DMAX_CH(x)_WR_UID_EN

Label	Description
Number of AXI Write Channel Unique ID's supported for Channel x? (for x = 1; x <= DMAX_NUM_CHANNELS)	This parameter allows user to select the number of AXI Write Channel Unique ID's supported. **Values:** 2, 4, 8 **Default Value:** 2 **Enabled:** DMAX_CH(x)_WR_UID_EN==1 **Parameter Name:** DMAX_CH(x)_WR_UID

5

Signal Descriptions

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

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

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

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

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

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

- **Active State:** Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).
- **Registered:** Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of *No* does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.
- **Synchronous to:** Indicates which clocks in the IP sample this input (drive for an output). 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. The presence of the postfix SuperList indicates a list of all possible clocks over all possible configs. Consult coreConsultant report for which clock applies to your specific configuration.
- **Exists:** Name of configuration parameter that populates this signal in your configuration.

- **Power Domain:** Name of power/voltage domain that this signal is part of when power/voltage islands are used. The SINGLE_DOMAIN value indicates that there are no islands.

Attributes used with Synchronous To:

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

The I/O signals are grouped as follows:

- "Core Clock and Reset Signals" on page 189
- "Test Interface Signals" on page 191
- "AHB Signals" on page 192
- "APB Signals" on page 196
- "Manager Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals" on page 200
- "Hardware Handshaking Signals" on page 214
- "External FIFO Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS)" on page 217
- "External UID Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS)" on page 219
- "Debug Interface Signals" on page 221
- "Interrupt Interface Signals" on page 230
- "Lock Step Signals" on page 238
- "Low Power Interface Signals" on page 241

5.1 Core Clock and Reset Signals

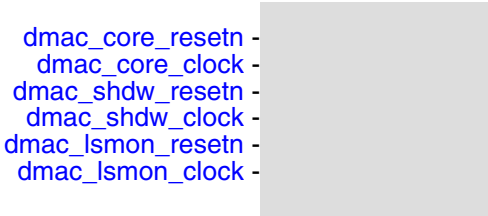


Table 5-1 Core Clock and Reset Signals

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
dmac_shdw_resetn	I	DW_axi_dmac shadow instance reset. Active low input that asynchronously resets the DW_axi_dmac. The reset must be synchronously de-asserted after the rising edge of dmac_shdw_clock. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. This reset is available only in the Lock Step mode of operation and is used for the Shadow DW_axi_dmac instance. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Port Name	I/O	Description
dmac_shdw_clock	I	DW_axi_dmac shadow instance clock. This clock is available only in the Lock Step mode of operation and is used for the Shadow DW_axi_dmac instance. **Note:** This dmac_shdw_clock must be synchronous to the dmac_core_clock. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_lsmon_resetn	I	DW_axi_dmac lock step monitor reset. Active low input that asynchronously resets the DW_axi_dmac. The reset must be synchronously de-asserted after the rising edge of dmac_lsmon_clock. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. This reset is available only in the Lock Step mode of operation and is used for the Shadow DW_axi_dmac instance. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_lsmon_clock	I	DW_axi_dmac lock step monitor clock. This clock is available only in the Lock Step mode of operation and is used for the Shadow DW_axi_dmac instance. **Note:** This dmac_lsmon_clock must be synchronous to the dmac_core_clock. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5.2 Test Interface Signals

scan_mode -

Table 5-2 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.

5.3 AHB Signals

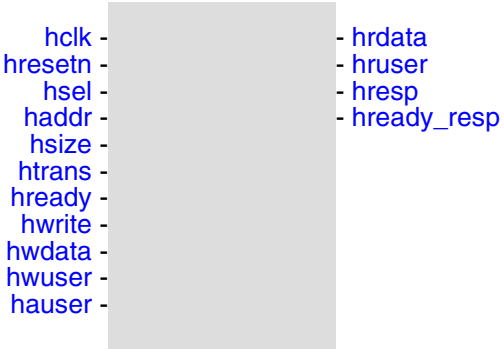


Table 5-3 AHB Signals

Port Name	I/O	Description
hclk	I	AHB register bus interface clock. **Exists:** (DMAX_SLVIF_CLOCK_MODE == 1) && (DMAX_SLVIF_MODE == 0) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hresetn	I	AHB register bus 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) && (DMAX_SLVIF_MODE == 0) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
hsel	I	Subordinate select. Asserted when the current transfer to the AHB bus is intended for the DW_axi_dmac. **Exists:** (DMAX_SLVIF_MODE == 0) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
haddr[31:0]	I	Address. **Exists:** (DMAX_SLVIF_MODE == 0) **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:** (DMAX_SLVIF_MODE == 0) **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:** (DMAX_SLVIF_MODE == 0) **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 completed. **Exists:** (DMAX_SLVIF_MODE == 0) **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:** (DMAX_SLVIF_MODE == 0) **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

Port Name	I/O	Description
hwdata[(DMAX_S_DATA_WIDTH-1):0]	I	Write data to subordinate. **Exists:** (DMAX_SLVIF_MODE == 0) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hwuser[((DMAX_S_DATA_WIDTH/8)-1):0]	I	AHB Register Bus Interface write data parity. This write data parity is driven by the System manager that configures DW_axi_dmac. **Exists:** (DMAX_SLVIF_MODE == 0) && (DMAX_PARITY_PROT_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hauser[((DMAX_S_ADDR_WIDTH/8)-1):0]	I	AHB Register Bus Interface address parity. This address parity is driven by the System manager that configures DW_axi_dmac. **Exists:** (DMAX_SLVIF_MODE == 0) && (DMAX_PARITY_PROT_EN == 1) && (DMAX_ADDR_PARITY_PROT_EN) **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 subordinate. **Exists:** (DMAX_SLVIF_MODE == 0) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
hruser[((DMAX_S_DATA_WIDTH/8)-1):0]	O	AHB Register Bus Interface read data parity. The DW_axi_dmac drives this read data parity along with the read data. **Exists:** (DMAX_SLVIF_MODE == 0) && (DMAX_PARITY_PROT_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
hresp[1:0]	O	Response type from subordinate. 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:** (DMAX_SLVIF_MODE == 0) **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:** (DMAX_SLVIF_MODE == 0) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "hclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

5.4 APB Signals

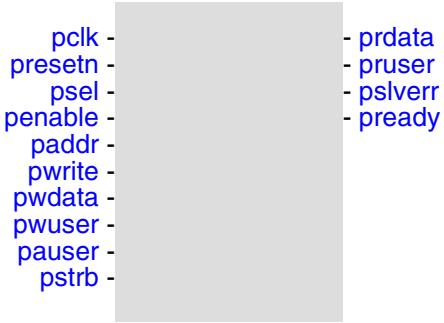


Table 5-4 APB Signals

Port Name	I/O	Description
pclk	I	APB completer interface clock. **Exists:** (DMAX_SLVIF_CLOCK_MODE == 1) && (DMAX_SLVIF_MODE == 2) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A.
presetn	I	APB completer interface reset. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of pclk. DW_axi_dmac does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (DMAX_SLVIF_CLOCK_MODE == 1) && (DMAX_SLVIF_MODE == 2) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low
psel	I	Completer select. Asserted to transfer to SETUP state of the APB bus. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
penable	I	Completer Penable. Asserted to transfer to ACCESS state of the APB bus. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
paddr[31:0]	I	APB Address. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A.
pwrite	I	APB transfer direction. When high, this signal indicates a write transfer. When low, this signal indicates a read transfer. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A.
pwdata[(DMAX_S_DATA_WIDTH-1):0]	I	APB Write data to completer. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** (DMAX_PARITY_PROT_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A.
pwuser[((DMAX_S_DATA_WIDTH/8)-1):0]	I	APB4 Completer Bus Interface write data parity. This write data parity is driven by the System manager that configures DW_axi_dmac. **Exists:** (DMAX_SLVIF_MODE == 2) && (DMAX_PARITY_PROT_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
pauser[((DMAX_S_ADDR_WIDTH/8)-1):0]	I	APB4 Completer Bus Interface address parity. This address parity is driven by the system manager that configures DW_axi_dmac. **Exists:** (DMAX_SLVIF_MODE == 2) && (DMAX_PARITY_PROT_EN == 1) && (DMAX_ADDR_PARITY_PROT_EN) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pstrb[((DMAX_S_DATA_WIDTH/8)-1):0]	I	Write data strobe to completer. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A.
prdata[(DMAX_S_DATA_WIDTH-1):0]	O	APB Read data from completer. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High.
pruser[((DMAX_S_DATA_WIDTH/8)-1):0]	O	APB4 Completer Bus Interface read data parity. The DW_axi_dmac drives this read data parity during read cycles when pwrite is Low. **Exists:** (DMAX_SLVIF_MODE == 2) && (DMAX_PARITY_PROT_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pslverr	O	APB response type from completer. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High.

Port Name	I/O	Description
pready	O	APB Transfer complete. **Exists:** (DMAX_SLVIF_MODE == 2) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? "pclk" : "dmac_core_clock" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High.

5.5 Manager Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals

aclk_mi -
aresetn_mi -
awready_mi -
awready_parity_mi -
arready_mi -
arready_parity_mi -
wready_mi -
wready_parity_mi -
rid_mi -
rdata_mi -
rresp_mi -
rlast_mi -
ruser_mi -
rvalid_mi -
rvalid_parity_mi -
bid_mi -
bresp_mi -
buser_mi -
bvalid_mi -
bvalid_parity_mi -
dmac_endian_format_mstif_mi -
dmac_le_select_lli_mstif_mi -

- awid_mi
- awaddr_mi
- awlen_mi
- awsize_mi
- awburst_mi
- awlock_mi
- awcache_mi
- awprot_mi
- awqos_mi
- awuser_mi
- awvalid_mi
- awvalid_parity_mi
- arid_mi
- araddr_mi
- arlen_mi
- arsize_mi
- arburst_mi
- arlock_mi
- arcache_mi
- arprot_mi
- arqos_mi
- aruser_mi
- arvalid_mi
- arvalid_parity_mi
- wid_mi
- wdata_mi
- wstrb_mi
- wlast_mi
- last_write_mi
- wuser_mi
- wvalid_mi
- wvalid_parity_mi
- rready_mi
- rready_parity_mi
- bready_mi
- bready_parity_mi

Table 5-5 Manager Interface (for i = 1; i <= DMAX_NUM_MASTER_IF) Signals

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

Port Name	I/O	Description
aresetn_mi	I	AXI3/AXI4 Manager i interface reset. Active low input that asynchronously resets the AXI3/AXI4 Manager i interface to its default state. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of aclk_mi/dmac_core_clock. 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 Manager 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 manager 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 Manager 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 Manager 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

Port Name	I/O	Description
awsize_mi[2:0]	O	AXI3/AXI4 Manager 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 Manager 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 Manager 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 Manager 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 Manager 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

Port Name	I/O	Description
awqos_mi[3:0]	O	AXI4 Manager 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 manager 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
awuser_mi[(DMAX_M_AWUSER_WIDTH-1):0]	O	AXI Write Address Channel User sideband signal. This signal used to transport the Write Address channel ECC checkbits. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_PROT_EN == 1) && (DMAX_ECC_MXIF_EN== 1 || DMAX_MXIF_ECC_PAR_EN== 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 Manager 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
awvalid_parity_mi	O	AXI Manager i Interface Write Address valid parity output. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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

Port Name	I/O	Description
awready_mi	I	AXI3/AXI4 Manager i interface write address ready. Indicates the AXI subordinate 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
awready_parity_mi	I	AXI Manager i Interface Write Address ready parity input. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
arid_mi[(DMAX_M_ID_WIDTH-1):0]	O	AXI3/AXI4 Manager 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 manager 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 Manager 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

Port Name	I/O	Description
arlen_mi[(DMAX_M_BURSTLEN_WIDTH-1):0]	O	AXI3/AXI4 Manager 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 Manager 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
arburst_mi[1:0]	O	AXI3/AXI4 Manager 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 Manager 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 Manager 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

Port Name	I/O	Description
arprot_mi[2:0]	O	AXI3/AXI4 Manager 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
arqos_mi[3:0]	O	AXI4 Manager 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 manager 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
aruser_mi[(DMAX_M_ARUSER_WIDTH-1):0]	O	AXI Read Address Channel User sideband signal. This signal used to transport the Read Address channel ECC checkbits. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_PROT_EN == 1) && (DMAX_ECC_MXIF_EN== 1 || DMAX_MXIF_ECC_PAR_EN== 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 Manager 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

Port Name	I/O	Description
arvalid_parity_mi	O	AXI Manager i Interface Read Address valid parity output. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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 Manager i interface read address ready. Indicates the AXI subordinate 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
arready_parity_mi	I	AXI Manager i Interface Read Address ready parity input. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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 Manager 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 manager 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

Port Name	I/O	Description
wdata_mi[(DMAX_M_DATA_WIDTH-1):0]	O	AXI3/AXI4 Manager 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 Manager 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 Manager 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
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
wuser_mi[(DMAX_M_WUSER_WIDTH-1):0]	O	AXI Write Data Channel User sideband signal. This signal used to transport the Write Data channel ECC checkbits. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_PROT_EN == 1) && (DMAX_ECC_MXIF_EN== 1 || DMAX_MXIF_ECC_PAR_EN== 1)) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
wvalid_mi	O	AXI3/AXI4 Manager 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
wvalid_parity_mi	O	AXI Manager i Interface Write Data valid parity output. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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 Manager i interface write data ready. Indicates the AXI subordinate 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
wready_parity_mi	I	AXI Manager i Interface Write Data ready parity input. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rid_mi[(DMAX_M_ID_WIDTH-1):0]	I	AXI3/AXI4 Manager 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:** (DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
rdata_mi[(DMAX_M_DATA_WIDTH-1):0]	I	AXI3/AXI4 Manager i interface read data. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp_mi[1:0]	I	AXI3/AXI4 Manager i Interface read response. AXI subordinate indicates the status of read transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rlast_mi	I	AXI3/AXI4 Manager 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:** (DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** High
ruser_mi[(DMAX_M_RUSER_WIDTH-1):0]	I	AXI Read Data Channel User sideband signal. This signal used to transport the Read Data channel ECC checkbits. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_PROT_EN == 1) && (DMAX_ECC_MXIF_EN== 1 || DMAX_MXIF_ECC_PAR_EN== 1)) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
rvalid_mi	I	AXI3/AXI4 Manager 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
rvalid_parity_mi	I	AXI Manager i Interface Read Data valid parity input. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rready_mi	O	AXI3/AXI4 Manager 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
rready_parity_mi	O	AXI Manager i Interface Read Data ready parity output. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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 Manager 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

Port Name	I/O	Description
bresp_mi[1:0]	I	AXI3/AXI4 Manager i interface write response. AXI subordinate indicates the status of write transaction. **Exists:** Always **Synchronous To:** (DMAX_MSTIF(i)_CLOCK_MODE == 1) ? "aclk_mi" : "dmac_core_clock" **Registered:** (DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
buser_mi[(DMAX_M_BUSER_WIDTH-1):0]	I	AXI Write Response Channel User sideband signal. This signal used to transport the Write Response channel ECC checkbits. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_PROT_EN == 1) && (DMAX_ECC_MXIF_EN== 1 || DMAX_MXIF_ECC_PAR_EN== 1)) **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 Manager i interface write response valid. AXI subordinate 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
bvalid_parity_mi	I	AXI Manager i Interface Write Response valid parity input. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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 Manager 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

Port Name	I/O	Description
bready_parity_mi	O	AXI Manager i Interface Write Response ready parity output. **Exists:** (DMAX_MXIF_READY_VALID_PAR_EN == 1) **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
dmac_endian_format_mstif_mi	I	Manager 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	Manager 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

5.6 Hardware Handshaking Signals

Table 5-6 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

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:** (((DMAX_ASYNC_HS_EN == 1) && (DMAX_HS(y)_ASYNC_CLK == 1)) ? "Yes" : "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:** (((DMAX_ASYNC_HS_EN == 1) && (DMAX_HS(y)_ASYNC_CLK == 1)) ? "Yes" : "No") **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

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

5.7 External FIFO Memory Interface Signals (for X = 1; X <= DMAX-_NUM_CHANNELS)

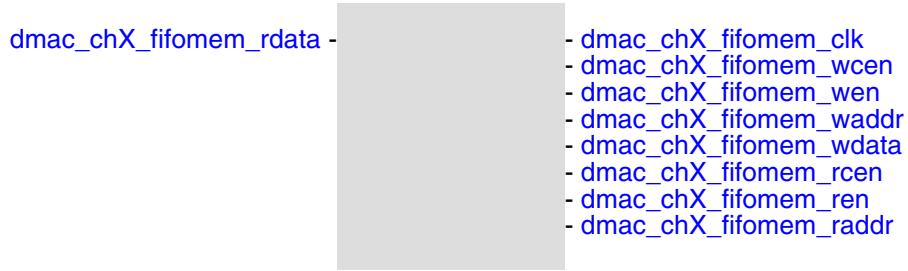


Table 5-7 External FIFO Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS)

Port Name	I/O	Description
dmac_chX_fifomem_clk	O	Channel X FIFO memory clock **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_fifomem_wcen	O	Channel X FIFO memory write port enable **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** (DMAX_CH_MEM_EXT>= 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_fifomem_wen	O	Channel X FIFO memory write enable **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** (DMAX_CH_MEM_EXT>= 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_fifomem_waddr[(LOG2_DMAX_CH1_FIFO_DEPTH-1):0]	O	Channel X FIFO memory write address **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
dmac_chX_fifomem_wdata[(DMAX_CH1_EXTMEM_DATA_WIDTH-1):0]	O	Channel X FIFO memory write data **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_fifomem_rcen	O	Channel X FIFO memory read port enable **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_fifomem_ren	O	Channel X FIFO memory read enable **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_fifomem_raddr[(LOG2_DMAX_CH1_FIFO_DEPTH-1):0]	O	Channel X FIFO memory read address **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_fifomem_rdata[(DMAX_CH1_EXTMEM_DATA_WIDTH-1):0]	I	Channel X FIFO memory read data **Exists:** (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5.8 External UID Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS)

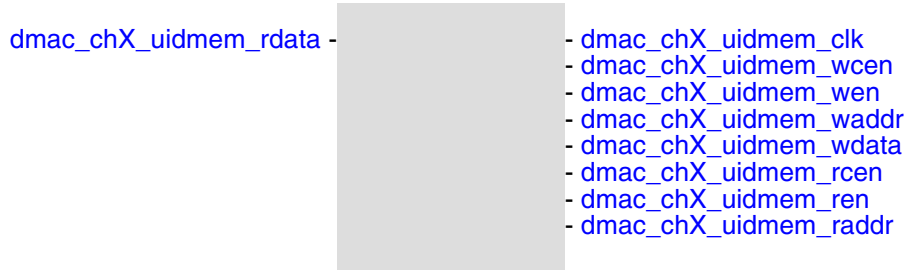


Table 5-8 External UID Memory Interface Signals (for X = 1; X <= DMAX_NUM_CHANNELS)

Port Name	I/O	Description
dmac_chX_uidmem_clk	O	Channel X UID Memory clock **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_uidmem_wcen	O	Channel X UID Memory write port enable **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_uidmem_wen	O	Channel X UID Memory write enable **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_uidmem_waddr[(LOG2_DMAX_CH1_UIDMEM_DEPTH-1):0]	O	Channel X UID Memory write address **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
dmac_chX_uidmem_wdata[(DMAX_CH1_UIDMEM_DATA_WIDTH-1):0]	O	Channel X UID Memory write data **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_uidmem_rcen	O	Channel X UID Memory read port enable **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_uidmem_ren	O	Channel X UID Memory read enable **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** Low
dmac_chX_uidmem_raddr[(LOG2_DMAX_CH1_UIDMEM_DEPTH-1):0]	O	Channel X UID Memory read address **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
dmac_chX_uidmem_rdata[(DMAX_CH1_UIDMEM_DATA_WIDTH-1):0]	I	Channel X UID Memory read data **Exists:** (DMAX_CHX_RD_UID_EN == 1) && (DMAX_NUM_CHANNELS >= X) && (DMAX_CH_MEM_EXT == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5.9 Debug Interface Signals

Table 5-9 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

Port Name	I/O	Description
slvif_busy	O	Register Bus Interface status signal indicating that whether the register bus interface is busy or not. This signal is asserted if there is an active transfer on the register bus interface. **Exists:** (DMAX_SLVIF_STATUS_OP_EN == 1) **Synchronous To:** (DMAX_SLVIF_CLOCK_MODE == 1) ? ((DMAX_SLVIF_MODE == 0) ? "hclk" : "pclk") : "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 Manager 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 manager 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

Port Name	I/O	Description
debug_grant_index_ar_ch_m1[(LOG2_DMAX_ARB_RD_REQ_WIDTH-1):0]	O	Debug port AXI Manager 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:** (DMAX_MULT_ARB_EN == 1) ? "No" : "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 Manager 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:** (DMAX_MULT_ARB_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_wr_arb_req_m1[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Manager 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

Port Name	I/O	Description
debug_ch_rd_arb_req_m1[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Manager 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 Manager 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 Manager 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:** (DMAX_MULT_ARB_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
debug_grant_index_aw_ch_m2[(LOG2_DMAX_ARB_WR_REQ_WIDTH-1):0]	O	Debug port AXI Manager 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:** (DMAX_MULT_ARB_EN == 1) ? "No" : "Yes" **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_wr_arb_req_m2[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Manager Interface 2 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) && (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 Manager Interface 2 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) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
debug_ch_lli_rd_req_m2[(DMAX_NUM_CHANNELS-1):0]	O	Debug port AXI Manager Interface 2 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) && (DMAX_NUM_MASTER_IF > 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
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
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

Port Name	I/O	Description
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
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

Port Name	I/O	Description
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
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

Port Name	I/O	Description
debug_ch_aborted	O	Multiplexed - Channel Terminate status (generated due to Channel Suspend request, Channel disable, User Channel terminate, 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

5.10 Interrupt Interface Signals

- intr
- intr_ch
- intr_cmnreg
- wrparity_prot_err_intr
- wrparity_prot_err_intr_ch
- wrparity_prot_err_intr_cmnreg
- addrparity_prot_err_intr
- addrparity_prot_err_intr_ch
- addrparity_prot_err_intr_cmnreg
- eccprot_chmem_corr_err_intr
- eccprot_chmem_corr_err_intr_ch
- eccprot_chmem_uncorr_err_intr
- eccprot_chmem_uncorr_err_intr_ch
- eccprot_uidmem_corr_err_intr
- eccprot_uidmem_corr_err_intr_ch
- eccprot_uidmem_uncorr_err_intr
- eccprot_uidmem_uncorr_err_intr_ch
- prot_mxif_corr_err_intr
- prot_mxifi_corr_err_intr_rch0 (for i = 1; i <= DMAX_NUM_MASTER_IF)
- prot_mxifi_corr_err_intr_rch1 (for i = 1; i <= DMAX_NUM_MASTER_IF)
- prot_mxifi_corr_err_intr_bch (for i = 1; i <= DMAX_NUM_MASTER_IF)
- prot_mxif_uncorr_err_intr
- prot_mxif_valid_ready_parerr_intr
- prot_mxif_valid_ready_parerr_intr_ch
- prot_mxifi_uncorr_err_intr_rch0 (for i = 1; i <= DMAX_NUM_MASTER_IF)
- prot_mxifi_uncorr_err_intr_rch1 (for i = 1; i <= DMAX_NUM_MASTER_IF)
- prot_mxifi_uncorr_err_intr_bch (for i = 1; i <= DMAX_NUM_MASTER_IF)

Table 5-10 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

Port Name	I/O	Description
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
wrparity_prot_err_intr	O	Logical OR of all individual (combined) Write Parity Protection Error from Common Register Space and Channel Specific Register spaces. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (DMAX_INTR_IO_TYPE != 1) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" :"dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
wrparity_prot_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated write parity error signal from each Channel Specific Register spaces. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (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
wrparity_prot_err_intr_cmnreg	O	Dedicated write parity protection error signal from common register space. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (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

Port Name	I/O	Description
addrparity_prot_err_intr	O	Logical OR of all individual (combined) Address Parity Protection Error from Common Register Space and Channel Specific Register spaces. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (DMAX_INTR_IO_TYPE != 1) && (DMAX_ADDR_PARITY_PROT_EN) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" :"dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
addrparity_prot_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated address parity error signal from each Channel Specific Register spaces. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (DMAX_INTR_IO_TYPE > 0) && (DMAX_ADDR_PARITY_PROT_EN) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" :"dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
addrparity_prot_err_intr_cmnreg	O	Dedicated address parity protection error signal from common register space. **Exists:** (DMAX_PARITY_PROT_EN == 1) && (DMAX_INTR_IO_TYPE > 0) && (DMAX_ADDR_PARITY_PROT_EN) **Synchronous To:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "hclk" :"dmac_core_clock" **Registered:** (DMAX_INTR_SYNC2SLVCLK == 1) ? "Yes" : "No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_chmem_corr_err_intr	O	Logical OR of all individual (combined) ECC Protection Correctable Error from all Channel FIFO Memory interfaces. Please refer the section - "ECC Protection feature FIFO Memory Interface" for more information. **Exists:** ((DMAX_ECC_MEMIF_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
eccprot_chmem_corr_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated ECC Protection Correctable Error from each Channel FIFO Memory interfaces. Please refer the section "ECC Protection feature - FIFO Memory Interface" for more information. **Exists:** ((DMAX_ECC_MEMIF_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_chmem_uncorr_err_intr	O	Logical OR of all individual (combined) ECC Protection Uncorrectable Error from all Channel FIFO Memory interfaces. Please refer the section - "ECC Protection feature FIFO Memory Interface" for more information. **Exists:** ((DMAX_ECC_MEMIF_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_chmem_uncorr_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated ECC Protection Uncorrectable Error from each Channel FIFO Memory interfaces. Please refer the section "ECC Protection feature - FIFO Memory Interface" for more information. **Exists:** ((DMAX_ECC_MEMIF_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_uidmem_corr_err_intr	O	Logical OR of all individual (combined) ECC Protection Correctable Error from all Channel UID Memory interfaces. Please refer the section - "ECC Protection feature - UID Memory Interface" for more information. **Exists:** ((DMAX_ECC_UIDMEMIF_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
eccprot_uidmem_corr_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated ECC Protection Correctable Error from each Channel UID Memory interfaces. Please refer the section "ECC Protection feature - UID Memory Interface" for more information. **Exists:** ((DMAX_ECC_UIDMEMIF_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_uidmem_uncorr_err_intr	O	Logical OR of all individual (combined) ECC Protection Uncorrectable Error from all Channel UID Memory interfaces. Please refer the section - "ECC Protection feature UID Memory Interface" for more information. **Exists:** ((DMAX_ECC_UIDMEMIF_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
eccprot_uidmem_uncorr_err_intr_ch[(DMAX_NUM_CHANNELS-1):0]	O	Dedicated ECC Protection Uncorrectable Error from each Channel UID Memory interfaces. Please refer the section "ECC Protection feature - UID Memory Interface" for more information. **Exists:** ((DMAX_ECC_UIDMEMIF_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxif_corr_err_intr	O	Logical OR of all individual (combined) ECC Protection Correctable Error from all Channel AXI Manager interfaces. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
prot_mxifi_corr_err_intr_rch0 (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated ECC Protection Correctable Error from AXI Manager Interface (i) Read Data Channel (Read data). Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxifi_corr_err_intr_rch1 (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated ECC Protection Correctable Error from AXI Manager Interface (i) Read Data Channel (Other signals). Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxifi_corr_err_intr_bch (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated ECC Protection Correctable Error from AXI Manager Interface (i) Write Response Channel. Please refer the section - "ECC Protection feature AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxif_uncorr_err_intr	O	Logical OR of all individual (combined) Parity/ECC Protection Uncorrectable Error from all Channel AXI Manager interfaces. Please refer the section - "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
prot_mxif_valid_ready_parerr_intr	O	Logical OR of all individual (combined) VALID and READY Parity Protection Error from all AXI Manager interfaces. Please refer the section "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_MXIF_READY_VALID_PAR_EN == 1) && (DMAX_INTR_IO_TYPE != 1)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxif_valid_ready_parerr_intr_ch[(MXIF_VLD_RDY_ERRINTR_WIDTH-1):0]	O	Dedicated VALID and READY signal Parity Error from each AXI Manager Interface AR/AW/W/R/B Channels. Where each bit corresponds to specific channel VALID or READY Parity Error. Please refer the section "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_MXIF_READY_VALID_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxifi_uncorr_err_intr_rch0 (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated Parity/ECC Protection Uncorrectable Error from AXI Manager Interface (i) Read Data Channel (Read Data). Please refer the section - "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prot_mxifi_uncorr_err_intr_rch1 (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated Parity/ECC Protection Uncorrectable Error from AXI Manager Interface (i) Read Data Channel (Other signals). Please refer the section - "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
prot_mxifi_uncorr_err_intr_bch (for i = 1; i <= DMAX_NUM_MASTER_IF)	O	Dedicated Parity/ECC Protection Uncorrectable Error from AXI Manager Interface (i) Write Response Channel. Please refer the section - "ECC and Parity Protection feature for AXI Manager Interface" for more information. **Exists:** ((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_INTR_IO_TYPE > 0)) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

5.11 Lock Step Signals

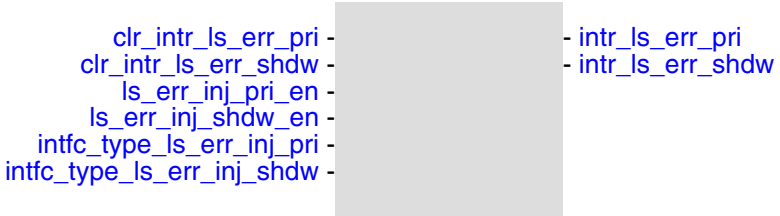


Table 5-11 Lock Step Signals

Port Name	I/O	Description
intr_ls_err_pri	O	Primary Lock Step Safety Monitor - Lock Step Protection error interrupt output. This signal indicates the status of the safety monitor comparison between the Primary and Shadow DMA Controller outputs. If the above comparison fails, then this signal will be asserted and this signal can be cleared only with the help of the clr_intr_ls_err_pri signal. Please refer the section Lock Step Protection "Safety Monitor" for more information. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
intr_ls_err_shdw	O	Shadow Lock Step Safety Monitor - Lock Step Protection error interrupt output. This signal indicates the status of the safety monitor comparison between the Primary and Shadow DMA Controller outputs. If the above comparison fails, then this signal will be asserted and this signal can be cleared only with the help of the clr_intr_ls_err_shdw signal. Please refer the section Lock Step Protection "Safety Monitor" for more information. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
clr_intr_ls_err_pri	I	Primary Lock Step Safety Monitor - Clear control signal for Lock Step Protection error interrupt input. This signal is used to clear the Lock Step Protection error interrupt. Please refer the section Lock Step Protection "Safety Monitor" for more information. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
clr_intr_ls_err_shdw	I	Shadow Lock Step Safety Monitor - Clear control signal for Lock Step Protection error interrupt. This signal is used to clear the Lock Step Protection error interrupt. Please refer the section Lock Step Protection "Safety Monitor" for more information. **Exists:** (DMAX_LS_PROT_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
ls_err_inj_pri_en	I	Primary Lock Step Protection Error Injection enable. This signal is used to artificially inject the error into the intr_ls_err_pri. Please refer the section Lock Step Protection "Error Injection" for more information. **Exists:** (DMAX_LS_DIAG_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
ls_err_inj_shdw_en	I	Shadow Lock Step Protection Error Injection enable. This signal is used to artificially inject the error into the intr_ls_err_shdw. Please refer the section Lock Step Protection "Error Injection" for more information. **Exists:** (DMAX_LS_DIAG_EN == 1) **Synchronous To:** dmac_core_clock **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
intfc_type_ls_err_inj_pri	I	Primary Lock Step Safety Monitor. This signal is used along with the ls_err_inj_pri_en signal to select the primary or the shadow instance of the Primary Lock Step Safety Monitor for error injection. When ls_err_inj_pri_en=1, ■ 0 corresponds to primary instance ■ 1 corresponds to shadow instance For more information, see the section "Lock Step Protection - Error Injection". **Exists:** (SNPS_RSVDPARAM_148 == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Port Name	I/O	Description
intfc_type_ls_err_inj_shdw	I	Shadow Lock Step Safety Monitor. This signal is used along with ls_err_inj_shdw_en to select primary or shadow instance of Shadow Lock Step Safety Monitor for error injection. When ls_err_inj_shdw_en=1, ■ 0 corresponds to primary instance ■ 1 corresponds to shadow instance For more information, see the section "Lock Step Protection - Error Injection". **Exists:** (SNPS_RSVDPARAM_148 == 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

5.12 Low Power Interface Signals

- dmac_lp_req
- ch_lp_req
- sbiu_lp_req
- mxif_arch_lp_req
- mxif_arch_oclk_lp_req
- mxif_awch_lp_req
- mxif_awch_oclk_lp_req
- mxif_wch_lp_req
- mxif_wch_oclk_lp_req
- mxif_bch_lp_req
- mxif_bch_oclk_lp_req
- mxif_rch_lp_req
- mxif_rch_oclk_lp_req
- mxif_r_ch_idle
- mxif_b_ch_idle

Table 5-12 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_CLOCK_MODE == 1) ? ((DMAX_SLVIF_MODE == 0) ? "hclk" : "pclk") : "dmac_core_clock" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

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 Manager 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 Manager Interface Context Sensitive Low Power feature. When AXI Manager interface i clock is synchronous, respective mxif_arch_oclk_lp_req bit is tied to 0. **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 Manager 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 Manager Interface Context Sensitive Low Power feature. When AXI Manager interface i clock is synchronous, respective mxif_awch_oclk_lp_req bit is tied to 0. **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 Manager 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

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 Manager Interface Context Sensitive Low Power feature. When AXI Manager interface i clock is synchronous, respective mxif_wch_oclk_lp_req bit is tied to 0. **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 Manager 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 Manager Interface Context Sensitive Low Power feature. When AXI Manager interface i clock is synchronous, respective mxif_bch_oclk_lp_req bit is tied to 0. **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 Manager 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 Manager Interface Context Sensitive Low Power feature. When AXI Manager interface i clock is synchronous, respective mxif_rch_oclk_lp_req bit is tied to 0. **Exists:** (DMAX_MXIF_CSLP_EN== 1) **Synchronous To:** dmac_core_clock **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Port Name	I/O	Description
mxif_r_ch_idle[(DMAX_NUM_MASTER_IF-1):0]	O	Debug port AXI Manager 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 Manager 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

6

Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 6-1 Possible Read and Write Behaviors

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

Table 6-2 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 6-3 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 controller 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.

Register definitions for each component memory map.

Table 6-4 Registers for the DW_axi_dmac_mem_map Memory Map

Register	Offset	Description
DW_axi_dmac common register address block Common_Registers_Address_Block Exists: Always		
“DMAC_IDREG” on page 252	0x0	DW_axi_dmac Component ID Register
“DMAC_COMPVERREG” on page 253	0x8	DW_axi_dmac Component Version Register
“DMAC_CFGREG” on page 254	0x10	DW_axi_dmac Global Configuration Register
“DMAC_CHENREG” on page 256	0x18	DW_axi_dmac Channel Enable Register
“DMAC_CHENREG2” on page 284	0x18	DW_axi_dmac Channel Enable Register 2
“DMAC_CHSUSPREG” on page 309	0x20	DW_axi_dmac Channel Suspend Register
“DMAC_CHABORTREG” on page 348	0x28	DW_axi_dmac Channel Terminate Register
“DMAC_INTSTATUSREG2” on page 391	0x30	DW_axi_dmac Combined Interrupt Status Register 2

Register	Offset	Description
“DMAC_INTSTATUSREG” on page 399	0x30	DW_axi_dmac Combined Interrupt Status Register
“DMAC_COMMONREG_INTCLEARREG” on page 402	0x38	DW_axi_dmac Common Register Space Interrupt Clear Register
“DMAC_COMMONREG_INTSTATUS_ENABLEREG” on page 413	0x40	DW_axi_dmac Common Register Space Interrupt Enable Register
“DMAC_COMMONREG_INTSIGNAL_ENABLEREG” on page 428	0x48	DW_axi_dmac Common Register Space Interrupt Signal Enable Register
“DMAC_COMMONREG_INTSTATUSREG” on page 444	0x50	DW_axi_dmac Common Register Space Interrupt Status Register
“DMAC_RESETREG” on page 461	0x58	DW_axi_dmac Software Reset Register
“DMAC_LOWPOWER_CFGREG” on page 462	0x60	DW_axi_dmac Low Power Configuration Register
“DMAC_COMMON_PARCTLREG” on page 466	0x70	DW_axi_dmac Parity Control Register
“DMAC_COMMON_ECCCTLSTATUSREG” on page 468	0x78	DW_axi_dmac ECC Control and Status Register
“DMAC_SFTY_ERR_INJ_CTRL” on page 472	0x80	DW_axi_dmac Safety Error Injection Control Register
DW_axi_dmac Channel x register address block Channelx_Register_Address_Block (for x = 1; x <= DMAX_NUM_CHANNELS) Exists: DMAX_NUM_CHANNELS >= 1		
“CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 476	0x100 + (x-1)*0x100	DW_axi_dmac Channel x Source Address Register
“CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 477	0x108 + (x-1)*0x100	DW_axi_dmac Channel x Destination Address Register
“CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 478	0x110 + (x-1)*0x100	DW_axi_dmac Channel x Block Transfer Size Register
“CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 480	0x118 + (x-1)*0x100	DW_axi_dmac Channel x Control Register
“CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 493	0x120 + (x-1)*0x100	DW_axi_dmac Channel x Configuration Register
“CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 505	0x120 + (x-1)*0x100	DW_axi_dmac Channel x Configuration Register 2

Register	Offset	Description
“CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 517	0x128 + (x-1)*0x100	DW_axi_dmac Channel x Linked List Pointer Register
“CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 519	0x130 + (x-1)*0x100	DW_axi_dmac Channel x Status Register
“CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 521	0x138 + (x-1)*0x100	DW_axi_dmac Channel x Software Handshake Source Register
“CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 525	0x140 + (x-1)*0x100	DW_axi_dmac Channel x Software Handshake Destination Register
“CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 529	0x148 + (x-1)*0x100	DW_axi_dmac Channel x Block Transfer Resume Request Register
“CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 531	0x150 + (x-1)*0x100	DW_axi_dmac Channel x AXI ID Register
“CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 533	0x158 + (x-1)*0x100	DW_axi_dmac Channel x AXI QoS Register
“CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 535	0x160 + (x-1)*0x100	DW_axi_dmac Channel x Source Status Register
“CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 537	0x168 + (x-1)*0x100	DW_axi_dmac Channel x Destination Status Register
“CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 539	0x170 + (x-1)*0x100	DW_axi_dmac Channel x Source Status Fetch Register
“CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 540	0x178 + (x-1)*0x100	DW_axi_dmac Channel x Destination Status Fetch Register
“CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 541	0x180 + (x-1)*0x100	DW_axi_dmac Channel x Interrupt Status Enable Register
“CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 555	0x188 + (x-1)*0x100	DW_axi_dmac Channel x Interrupt Status Register

Register	Offset	Description
“CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 573	0x190 + (x-1)*0x100	DW_axi_dmac Channel x Interrupt Signal Enable Register
“CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 587	0x198 + (x-1)*0x100	DW_axi_dmac Channel x Interrupt Status Clear Register
“CHx_CFG_EXTD (for x = 1; x <= DMAX_NUM_CHANNELS)” on page 597	0x1A0 + (x-1)*0x100	DW_axi_dmac Channel x Extended Configuration Register

6.1 Common_Registers_Address_Block Registers

6.1.1 DMAC_IDREG

- **Name:** DW_axi_dmac Component ID Register
- **Description:** DMAC ID Register contains a 32-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 6-5 Fields for Register: DMAC_IDREG

Bits	Name	Memory Access	Description
63:32	RSVD_DMAC_IDREG	R	DMAC_IDREG Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
31:0	DMAC_ID	R	DMAC ID Number. **Value After Reset:** DMAX_ID_NUM **Exists:** Always

6.1.2 DMAC_COMPVERREG

- **Name:** DW_axi_dmac Component Version Register
- **Description:** This register contains a 32-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 6-6 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

6.1.3 DMAC_CFGREG

- **Name:** DW_axi_dmac Global Configuration Register
- **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

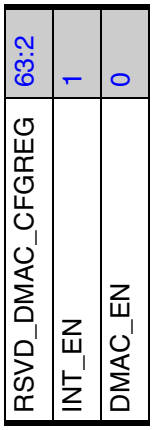


Table 6-7 Fields for Register: DMAC_CFGREG

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

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

6.1.4 DMAC_CHENREG

- **Name:** DW_axi_dmac Channel Enable Register
- **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 register bus 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 register bus 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 terminate 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 register bus 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 6-8 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

Bits	Name	Memory Access	Description
47	CH8_ABORT_WE	* Varies	This bit is used to write enable the Channel-8 Terminate 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 Terminate 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"}
45	CH6_ABORT_WE	* Varies	This bit is used to write enable the Channel-6 Terminate 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"}

Bits	Name	Memory Access	Description
44	CH5_ABORT_WE	* Varies	This bit is used to write enable the Channel-5 Terminate 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 Terminate 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"}
42	CH3_ABORT_WE	* Varies	This bit is used to write enable the Channel-3 Terminate 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"}

Bits	Name	Memory Access	Description
41	CH2_ABORT_WE	* Varies	This bit is used to write enable the Channel-2 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
39	CH8_ABORT	* Varies	Channel-8 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH8_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH8_ABORT): Request for Channel-8 Terminate ■ 0x0 (DISABLE_CH8_ABORT): No Request for Channel-8 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
38	CH7_ABORT	* Varies	Channel-7 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH7_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH7_ABORT): Request for Channel-7 Terminate ■ 0x0 (DISABLE_CH7_ABORT): No Request for Channel-7 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
37	CH6_ABORT	* Varies	Channel-6 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH6_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH6_ABORT): Request for Channel-6 Terminate ■ 0x0 (DISABLE_CH6_ABORT): No Request for Channel-6 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
36	CH5_ABORT	* Varies	Channel-5 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH5_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH5_ABORT): Request for Channel-5 Terminate ■ 0x0 (DISABLE_CH5_ABORT): No Request for Channel-5 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
35	CH4_ABORT	* Varies	Channel-4 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH4_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH4_ABORT): Request for Channel-4 Terminate ■ 0x0 (DISABLE_CH4_ABORT): No Request for Channel-4 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
34	CH3_ABORT	* Varies	Channel-3 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH3_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH3_ABORT): Request for Channel-3 Terminate ■ 0x0 (DISABLE_CH3_ABORT): No Request for Channel-3 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
33	CH2_ABORT	* Varies	Channel-2 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH2_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH2_ABORT): Request for Channel-2 Terminate ■ 0x0 (DISABLE_CH2_ABORT): No Request for Channel-2 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
32	CH1_ABORT	* Varies	Channel-1 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH1_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH1_ABORT): Request for Channel-1 Terminate ■ 0x0 (DISABLE_CH1_ABORT): No Request for Channel-1 Terminate **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

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

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

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

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

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

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

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

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

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

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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

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

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

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

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

6.1.5 DMAC_CHENREG2

- **Name:** DW_axi_dmac Channel Enable Register 2
- **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 register bus 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

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	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
CH32_EN_WE	CH31_EN_WE	CH30_EN_WE	CH29_EN_WE	CH28_EN_WE	CH27_EN_WE	CH26_EN_WE	CH25_EN_WE	CH24_EN_WE	CH23_EN_WE	CH22_EN_WE	CH21_EN_WE	CH20_EN_WE	CH19_EN_WE	CH18_EN_WE	CH17_EN_WE	CH32_EN	CH31_EN	CH30_EN	CH29_EN	CH28_EN	CH27_EN	CH26_EN	CH25_EN	CH24_EN	CH23_EN	CH22_EN	CH21_EN	CH20_EN	CH19_EN	CH18_EN	CH17_EN	CH16_EN_WE	CH15_EN_WE	CH14_EN_WE	CH13_EN_WE	CH12_EN_WE	CH11_EN_WE	CH10_EN_WE	CH9_EN_WE	CH8_EN_WE	CH7_EN_WE	CH6_EN_WE	CH5_EN_WE	CH4_EN_WE	CH3_EN_WE	CH2_EN_WE	CH1_EN_WE	CH16_EN	CH15_EN	CH14_EN	CH13_EN	CH12_EN	CH11_EN	CH10_EN	CH9_EN	CH8_EN	CH7_EN	CH6_EN	CH5_EN	CH4_EN	CH3_EN	CH2_EN	CH1_EN

Table 6-9 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

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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

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

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

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

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

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

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

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

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

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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

6.1.6 DMAC_CHSUSPREG

- **Name:** DW_axi_dmac Channel Suspend Register
- **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 register bus 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

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	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
CH32_SUSP_WE	CH31_SUSP_WE	CH30_SUSP_WE	CH29_SUSP_WE	CH28_SUSP_WE	CH27_SUSP_WE	CH26_SUSP_WE	CH25_SUSP_WE	CH24_SUSP_WE	CH23_SUSP_WE	CH22_SUSP_WE	CH21_SUSP_WE	CH20_SUSP_WE	CH19_SUSP_WE	CH18_SUSP_WE	CH17_SUSP_WE	CH32_SUSP	CH31_SUSP	CH30_SUSP	CH29_SUSP	CH28_SUSP	CH27_SUSP	CH26_SUSP	CH25_SUSP	CH24_SUSP	CH23_SUSP	CH22_SUSP	CH21_SUSP	CH20_SUSP	CH19_SUSP	CH18_SUSP	CH17_SUSP	CH16_SUSP_WE	CH15_SUSP_WE	CH14_SUSP_WE	CH13_SUSP_WE	CH12_SUSP_WE	CH11_SUSP_WE	CH10_SUSP_WE	CH9_SUSP_WE	CH8_SUSP_WE	CH7_SUSP_WE	CH6_SUSP_WE	CH5_SUSP_WE	CH4_SUSP_WE	CH3_SUSP_WE	CH2_SUSP_WE	CH1_SUSP_WE	CH16_SUSP	CH15_SUSP	CH14_SUSP	CH13_SUSP	CH12_SUSP	CH11_SUSP	CH10_SUSP	CH9_SUSP	CH8_SUSP	CH7_SUSP	CH6_SUSP	CH5_SUSP	CH4_SUSP	CH3_SUSP	CH2_SUSP	CH1_SUSP

Table 6-10 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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true
62	CH31_SUSP_WE	W	This bit is used as a write enable to the Channel-31 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH31_SUSP): Enable Write to respective CH31_SUSP bit ■ 0x0 (DISABLE_WR_CH31_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true

Bits	Name	Memory Access	Description
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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true
58	CH27_SUSP_WE	W	This bit is used as a write enable to the Channel-27 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH27_SUSP): Enable Write to respective CH27_SUSP bit ■ 0x0 (DISABLE_WR_CH27_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true

Bits	Name	Memory Access	Description
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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true
54	CH23_SUSP_WE	W	This bit is used as a write enable to the Channel-23 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH23_SUSP): Enable Write to respective CH23_SUSP bit ■ 0x0 (DISABLE_WR_CH23_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true

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

Bits	Name	Memory Access	Description
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
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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true
30	CH15_SUSP_WE	W	This bit is used as a write enable to the Channel-15 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH15_SUSP): Enable Write to respective CH15_SUSP bit ■ 0x0 (DISABLE_WR_CH15_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true

Bits	Name	Memory Access	Description
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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true
26	CH11_SUSP_WE	W	This bit is used as a write enable to the Channel-11 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH11_SUSP): Enable Write to respective CH11_SUSP bit ■ 0x0 (DISABLE_WR_CH11_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true

Bits	Name	Memory Access	Description
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${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true
22	CH7_SUSP_WE	W	This bit is used as a write enable to the Channel-7 Suspend bit. The read back value of this register bit is always 0. **Values:** ■ 0x1 (ENABLE_WR_CH7_SUSP): Enable Write to respective CH7_SUSP bit ■ 0x0 (DISABLE_WR_CH7_SUSP): Disable Write to CH${ch_num}_SUSP bit **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true

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

Bits	Name	Memory Access	Description
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
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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.1.7 DMAC_CHABORTREG

- **Name:** DW_axi_dmac Channel Terminate Register
- **Description:** This is DW_axi_dmac Channel Terminate Register. The channel terminate 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 register bus interface write transfer.
- **Size:** 64 bits
- **Offset:** 0x28
- **Exists:** DMAX_NUM_CHANNELS > 8

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	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
CH32_ABORT_WE	CH31_ABORT_WE	CH30_ABORT_WE	CH29_ABORT_WE	CH28_ABORT_WE	CH27_ABORT_WE	CH26_ABORT_WE	CH25_ABORT_WE	CH24_ABORT_WE	CH23_ABORT_WE	CH22_ABORT_WE	CH21_ABORT_WE	CH20_ABORT_WE	CH19_ABORT_WE	CH18_ABORT_WE	CH17_ABORT_WE	CH32_ABORT	CH31_ABORT	CH30_ABORT	CH29_ABORT	CH28_ABORT	CH27_ABORT	CH26_ABORT	CH25_ABORT	CH24_ABORT	CH23_ABORT	CH22_ABORT	CH21_ABORT	CH20_ABORT	CH19_ABORT	CH18_ABORT	CH17_ABORT	CH16_ABORT_WE	CH15_ABORT_WE	CH14_ABORT_WE	CH13_ABORT_WE	CH12_ABORT_WE	CH11_ABORT_WE	CH10_ABORT_WE	CH9_ABORT_WE	CH8_ABORT_WE	CH7_ABORT_WE	CH6_ABORT_WE	CH5_ABORT_WE	CH4_ABORT_WE	CH3_ABORT_WE	CH2_ABORT_WE	CH1_ABORT_WE	CH16_ABORT	CH15_ABORT	CH14_ABORT	CH13_ABORT	CH12_ABORT	CH11_ABORT	CH10_ABORT	CH9_ABORT	CH8_ABORT	CH7_ABORT	CH6_ABORT	CH5_ABORT	CH4_ABORT	CH3_ABORT	CH2_ABORT	CH1_ABORT

Table 6-11 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 Terminate 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"}

Bits	Name	Memory Access	Description
62	CH31_ABORT_WE	* Varies	This bit is used to write enable the Channel-31 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
59	CH28_ABORT_WE	* Varies	This bit is used to write enable the Channel-28 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
56	CH25_ABORT_WE	* Varies	This bit is used to write enable the Channel-25 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
53	CH22_ABORT_WE	* Varies	This bit is used to write enable the Channel-22 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
50	CH19_ABORT_WE	* Varies	This bit is used to write enable the Channel-19 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
47	CH32_ABORT	* Varies	Channel-32 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH32_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH32_ABORT): Request for Channel-32 Terminate ■ 0x0 (DISABLE_CH32_ABORT): No Request for Channel-32 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 31 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
46	CH31_ABORT	* Varies	Channel-31 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH31_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH31_ABORT): Request for Channel-31 Terminate ■ 0x0 (DISABLE_CH31_ABORT): No Request for Channel-31 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 30 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
45	CH30_ABORT	* Varies	Channel-30 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH30_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH30_ABORT): Request for Channel-30 Terminate ■ 0x0 (DISABLE_CH30_ABORT): No Request for Channel-30 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 29 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
44	CH29_ABORT	* Varies	Channel-29 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH29_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH29_ABORT): Request for Channel-29 Terminate ■ 0x0 (DISABLE_CH29_ABORT): No Request for Channel-29 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 28 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
43	CH28_ABORT	* Varies	Channel-28 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH28_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH28_ABORT): Request for Channel-28 Terminate ■ 0x0 (DISABLE_CH28_ABORT): No Request for Channel-28 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 27 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
42	CH27_ABORT	* Varies	Channel-27 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH27_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH27_ABORT): Request for Channel-27 Terminate ■ 0x0 (DISABLE_CH27_ABORT): No Request for Channel-27 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 26 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
41	CH26_ABORT	* Varies	Channel-26 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH26_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH26_ABORT): Request for Channel-26 Terminate ■ 0x0 (DISABLE_CH26_ABORT): No Request for Channel-26 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 25 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
40	CH25_ABORT	* Varies	Channel-25 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH25_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH25_ABORT): Request for Channel-25 Terminate ■ 0x0 (DISABLE_CH25_ABORT): No Request for Channel-25 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 24 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
39	CH24_ABORT	* Varies	Channel-24 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH24_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH24_ABORT): Request for Channel-24 Terminate ■ 0x0 (DISABLE_CH24_ABORT): No Request for Channel-24 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 23 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
38	CH23_ABORT	* Varies	Channel-23 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH23_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH23_ABORT): Request for Channel-23 Terminate ■ 0x0 (DISABLE_CH23_ABORT): No Request for Channel-23 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 22 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
37	CH22_ABORT	* Varies	Channel-22 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH22_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH22_ABORT): Request for Channel-22 Terminate ■ 0x0 (DISABLE_CH22_ABORT): No Request for Channel-22 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 21 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
36	CH21_ABORT	* Varies	Channel-21 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH21_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH21_ABORT): Request for Channel-21 Terminate ■ 0x0 (DISABLE_CH21_ABORT): No Request for Channel-21 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 20 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
35	CH20_ABORT	* Varies	Channel-20 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH20_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH20_ABORT): Request for Channel-20 Terminate ■ 0x0 (DISABLE_CH20_ABORT): No Request for Channel-20 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 19 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
34	CH19_ABORT	* Varies	Channel-19 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH19_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH19_ABORT): Request for Channel-19 Terminate ■ 0x0 (DISABLE_CH19_ABORT): No Request for Channel-19 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 18 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
33	CH18_ABORT	* Varies	Channel-18 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH18_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH18_ABORT): Request for Channel-18 Terminate ■ 0x0 (DISABLE_CH18_ABORT): No Request for Channel-18 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 17 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
32	CH17_ABORT	* Varies	Channel-17 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH17_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH17_ABORT): Request for Channel-17 Terminate ■ 0x0 (DISABLE_CH17_ABORT): No Request for Channel-17 Terminate **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 Terminate 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"}

Bits	Name	Memory Access	Description
30	CH15_ABORT_WE	* Varies	This bit is used to write enable the Channel-15 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
27	CH12_ABORT_WE	* Varies	This bit is used to write enable the Channel-12 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
24	CH9_ABORT_WE	* Varies	This bit is used to write enable the Channel-9 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
21	CH6_ABORT_WE	* Varies	This bit is used to write enable the Channel-6 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
18	CH3_ABORT_WE	* Varies	This bit is used to write enable the Channel-3 Terminate 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 Terminate 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 Terminate 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"}

Bits	Name	Memory Access	Description
15	CH16_ABORT	* Varies	Channel-16 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH16_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH16_ABORT): Request for Channel-16 Terminate ■ 0x0 (DISABLE_CH16_ABORT): No Request for Channel-16 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 15 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
14	CH15_ABORT	* Varies	Channel-15 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH15_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH15_ABORT): Request for Channel-15 Terminate ■ 0x0 (DISABLE_CH15_ABORT): No Request for Channel-15 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 14 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
13	CH14_ABORT	* Varies	Channel-14 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH14_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH14_ABORT): Request for Channel-14 Terminate ■ 0x0 (DISABLE_CH14_ABORT): No Request for Channel-14 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 13 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
12	CH13_ABORT	* Varies	Channel-13 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH13_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH13_ABORT): Request for Channel-13 Terminate ■ 0x0 (DISABLE_CH13_ABORT): No Request for Channel-13 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 12 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
11	CH12_ABORT	* Varies	Channel-12 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH12_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH12_ABORT): Request for Channel-12 Terminate ■ 0x0 (DISABLE_CH12_ABORT): No Request for Channel-12 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 11 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
10	CH11_ABORT	* Varies	Channel-11 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH11_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH11_ABORT): Request for Channel-11 Terminate ■ 0x0 (DISABLE_CH11_ABORT): No Request for Channel-11 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 10 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
9	CH10_ABORT	* Varies	Channel-10 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH10_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH10_ABORT): Request for Channel-10 Terminate ■ 0x0 (DISABLE_CH10_ABORT): No Request for Channel-10 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 9 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
8	CH9_ABORT	* Varies	Channel-9 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH9_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH9_ABORT): Request for Channel-9 Terminate ■ 0x0 (DISABLE_CH9_ABORT): No Request for Channel-9 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 8 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
7	CH8_ABORT	* Varies	Channel-8 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH8_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH8_ABORT): Request for Channel-8 Terminate ■ 0x0 (DISABLE_CH8_ABORT): No Request for Channel-8 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 7 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
6	CH7_ABORT	* Varies	Channel-7 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH7_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH7_ABORT): Request for Channel-7 Terminate ■ 0x0 (DISABLE_CH7_ABORT): No Request for Channel-7 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 6 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
5	CH6_ABORT	* Varies	Channel-6 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH6_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH6_ABORT): Request for Channel-6 Terminate ■ 0x0 (DISABLE_CH6_ABORT): No Request for Channel-6 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 5 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
4	CH5_ABORT	* Varies	Channel-5 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH5_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH5_ABORT): Request for Channel-5 Terminate ■ 0x0 (DISABLE_CH5_ABORT): No Request for Channel-5 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 4 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
3	CH4_ABORT	* Varies	Channel-4 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH4_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH4_ABORT): Request for Channel-4 Terminate ■ 0x0 (DISABLE_CH4_ABORT): No Request for Channel-4 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 3 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
2	CH3_ABORT	* Varies	Channel-3 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH3_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH3_ABORT): Request for Channel-3 Terminate ■ 0x0 (DISABLE_CH3_ABORT): No Request for Channel-3 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 2 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
1	CH2_ABORT	* Varies	Channel-2 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH2_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH2_ABORT): Request for Channel-2 Terminate ■ 0x0 (DISABLE_CH2_ABORT): No Request for Channel-2 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 1 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
0	CH1_ABORT	* Varies	Channel-1 Terminate Request. Software sets this bit to 1 to request channel terminate. If this bit is set to 1, DW_axi_dmac disables the channel immediately. Terminating the channel might result in AXI Protocol violation as DW_axi_dmac does not make sure that all AXI transfers initiated on the manager interface are completed.Terminating 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 manager 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 terminating. ■ 0: No Channel Terminate Request. ■ 1: Request for Channel Terminate. DW_axi_dmac clears this bit to 0 once the channel is terminated (when it sets CH1_Status.CH_ABORTED bit to 1). **Values:** ■ 0x1 (ENABLE_CH1_ABORT): Request for Channel-1 Terminate ■ 0x0 (DISABLE_CH1_ABORT): No Request for Channel-1 Terminate **Value After Reset:** 0x0 **Exists:** DMAX_NUM_CHANNELS > 0 **Volatile:** true **Memory Access:** {(DMAX_CH_ABORT_EN == 1) ? "read-write" : "read-only"}

6.1.8 DMAC_INTSTATUSREG2

- **Name:** DW_axi_dmac Combined Interrupt Status Register 2
- **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

63:33	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_INTSTATUSREG_63to33	CommonReg_IntStat	CH32_IntStat	CH31_IntStat	CH30_IntStat	CH29_IntStat	CH28_IntStat	CH27_IntStat	CH26_IntStat	CH25_IntStat	CH24_IntStat	CH23_IntStat	CH22_IntStat	CH21_IntStat	CH20_IntStat	CH19_IntStat	CH18_IntStat	CH17_IntStat	CH16_IntStat	CH15_IntStat	CH14_IntStat	CH13_IntStat	CH12_IntStat	CH11_IntStat	CH10_IntStat	CH9_IntStat	CH8_IntStat	CH7_IntStat	CH6_IntStat	CH5_IntStat	CH4_IntStat	CH3_IntStat	CH2_IntStat	CH1_IntStat

Table 6-12 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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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
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

Bits	Name	Memory Access	Description
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

6.1.9 DMAC_INTSTATUSREG

- **Name:** DW_axi_dmac Combined Interrupt Status Register
- **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

63:17	16	15:8	7	6	5	4	3	2	1	0
RSVD_DMAC_INTSTATUSREG_63to17	CommonReg_IntStat	RSVD_DMAC_INTSTATUSREG	CH8_IntStat	CH7_IntStat	CH6_IntStat	CH5_IntStat	CH4_IntStat	CH3_IntStat	CH2_IntStat	CH1_IntStat

Table 6-13 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
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

Bits	Name	Memory Access	Description
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
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

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

6.1.10 DMAC_COMMONREG_INTCLEARREG

- **Name:** DW_axi_dmac Common Register Space Interrupt Clear Register
- **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

63:31	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_COMMONREG_INTCLEARREG_63to31	Clear_MXIF2_BCH_PROT_BVALID_ParErr_IntStat	Clear_MXIF2_RCH_PROT_RVALID_ParErr_IntStat	Clear_MXIF2_WCH_PROT_WREADY_ParErr_IntStat	Clear_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	Clear_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	Clear_MXIF1_BCH_PROT_BVALID_ParErr_IntStat	Clear_MXIF1_RCH_PROT_RVALID_ParErr_IntStat	Clear_MXIF1_WCH_PROT_WREADY_ParErr_IntStat	Clear_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	Clear_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	Clear_MXIF2_BCH_PROT_UnCorrERR_IntStat	Clear_MXIF2_BCH_PROT_CorrERR_IntStat	Clear_MXIF2_RCH1_PROT_UnCorrERR_IntStat	Clear_MXIF2_RCH1_PROT_CorrERR_IntStat	Clear_MXIF2_RCH0_PROT_UnCorrERR_IntStat	Clear_MXIF2_RCH0_PROT_CorrERR_IntStat	Clear_MXIF1_BCH_PROT_UnCorrERR_IntStat	Clear_MXIF1_BCH_PROT_CorrERR_IntStat	Clear_MXIF1_RCH1_PROT_UnCorrERR_IntStat	Clear_MXIF1_RCH1_PROT_CorrERR_IntStat	Clear_MXIF1_RCH0_PROT_UnCorrERR_IntStat	Clear_MXIF1_RCH0_PROT_CorrERR_IntStat	Clear_SLVIF_UndefinedReg_DEC_ERR_IntStat	Clear_SLVIF_CommonReg_WRPARITY_ERR_IntStat	Clear_SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	RSVD_DMAC_COMMONREG_INTCLEARREG_5to4	Clear_SLVIF_CommonReg_WrOnHold_ERR_IntStat	Clear_SLVIF_CommonReg_RD2WO_ERR_IntStat	Clear_SLVIF_CommonReg_WR2RO_ERR_IntStat	Clear_SLVIF_CommonReg_DEC_ERR_IntStat

Table 6-14 Fields for Register: DMAC_COMMONREG_INTCLEARREG

Bits	Name	Memory Access	Description
63:31	RSVD_DMAC_COMMONREG_INTCLEARREG_63to31	W	DMAC Common Register Interrupt Clear Register (bits 63to31) Reserved bits - Write Only **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
30	Clear_MXIF2_BCH_PROT_BVALID_ParErr_IntStat	W	AXI Manager Interface 2 Write Response Channel BVALID Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_BCH_PROT_BVALID_ParErr): Clear the MXIF2_BCH_PROT_BVALID_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
29	Clear_MXIF2_RCH_PROT_RVALID_ParErr_IntStat	W	AXI Manager Interface 2 Read Data Channel RVALID Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_RCH_PROT_RVALID_ParErr): Clear the MXIF2_RCH_PROT_RVALID_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
28	Clear_MXIF2_WCH_PROT_WREADY_ParErr_IntStat	W	AXI Manager Interface 2 Write Data Channel WREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_WCH_PROT_WREADY_ParErr): Clear the MXIF2_WCH_PROT_WREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
27	Clear_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	W	AXI Manager Interface 2 Write Address Channel AWREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_AWCH_PROT_AWREADY_ParErr): Clear the MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
26	Clear_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	W	AXI Manager Interface 2 Read Address Channel ARREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_ARCH_PROT_ARREADY_ParErr): Clear the MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
25	Clear_MXIF1_BCH_PROT_BVALID_ParErr_IntStat	W	AXI Manager Interface 1 Write Response Channel BVALID Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_BCH_PROT_BVALID_ParErr): Clear the MXIF1_BCH_PROT_BVALID_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
24	Clear_MXIF1_RCH_PROT_RVALID_ParErr_IntStat	W	AXI Manager Interface 1 Read Data Channel RVALID Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_RCH_PROT_RVALID_ParErr): Clear the MXIF1_RCH_PROT_RVALID_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
23	Clear_MXIF1_WCH_PROT_WREADY_ParErr_IntStat	W	AXI Manager Interface 1 Write Data Channel WREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_WCH_PROT_WREADY_ParErr): Clear the MXIF1_WCH_PROT_WREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
22	Clear_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	W	AXI Manager Interface 1 Write Address Channel AWREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_AWCH_PROT_AWREADY_ParErr): Clear the MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
21	Clear_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	W	AXI Manager Interface 1 Read Address Channel ARREADY Parity Protection related Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_ARCH_PROT_ARREADY_ParErr): Clear the MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
20	Clear_MXIF2_BCH_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 2 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_BCH_PROT_UnCorrERR): Clear the MXIF2_BCH_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
19	Clear_MXIF2_BCH_PROT_CorrERR_IntStat	W	AXI Manager Interface 2 Write Response Channel ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_BCH_PROT_CorrERR): Clear the MXIF2_BCH_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
18	Clear_MXIF2_RCH1_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 2 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_RCH1_PROT_UnCorrERR): Clear the MXIF2_RCH1_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
17	Clear_MXIF2_RCH1_PROT_CorrERR_IntStat	W	AXI Manager Interface 2 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_RCH1_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_RCH1_PROT_CorrERR): Clear the MXIF2_RCH1_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
16	Clear_MXIF2_RCH0_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 2 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_RCH0_PROT_UnCorrERR): Clear the MXIF2_RCH0_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
15	Clear_MXIF2_RCH0_PROT_CorrERR_IntStat	W	AXI Manager Interface 2 Read Channel (Data) ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF2_RCH0_PROT_CorrERR): Clear the MXIF2_RCH0_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
14	Clear_MXIF1_BCH_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 1 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_BCH_PROT_UnCorrERR): Clear the MXIF1_BCH_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
13	Clear_MXIF1_BCH_PROT_CorrERR_IntStat	W	AXI Manager Interface 1 Write Response Channel ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_BCH_PROT_CorrERR): Clear the MXIF1_BCH_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
12	Clear_MXIF1_RCH1_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 1 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_RCH1_PROT_UnCorrERR): Clear the MXIF1_RCH1_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
11	Clear_MXIF1_RCH1_PROT_CorrERR_IntStat	W	AXI Manager Interface 1 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_RCH1_PROT_CorrERR): Clear the MXIF1_RCH1_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
10	Clear_MXIF1_RCH0_PROT_UnCorrERR_IntStat	W	AXI Manager Interface 1 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_RCH0_PROT_UnCorrERR): Clear the MXIF1_RCH0_PROT_UnCorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
9	Clear_MXIF1_RCH0_PROT_CorrERR_IntStat	W	AXI Manager Interface 1 Read Channel (Data) ECC Protection related Correctable Error Interrupt Clear bit. This bit is used to clear the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (CLEAR_MXIF1_RCH0_PROT_CorrERR): Clear the MXIF1_RCH0_PROT_CorrERR_IntStat interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
8	Clear_SLVIF_UndefinedReg_DEC_ERR_IntStat	W	Register Bus 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	Clear_SLVIF_CommonReg_WRPARITY_ERR_IntStat	W	Register Bus Interface Common Register Write Parity Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit (SLVIF_CommonReg_WRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_WRPARITY_ERR): Clear the SLVIF_CommonReg_WRPARITY_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
6	Clear_SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	W	Register Bus Interface Common Register Address Parity Error Interrupt clear Bit. This bit is used to clear the corresponding channel interrupt status bit (SLVIF_CommonReg_ADDRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (CLEAR_SLVIF_CommonReg_ADDRPARITY_ERR): Clear the SLVIF_CommonReg_ADDRPARITY_ERR interrupt in the interrupt register DMAC_CommonReg_IntStatusReg ■ 0x0 (No_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
5:4	RSVD_DMAC_COMMONREG_INTCLEARREG_5to4	W	DMAC Common Register Interrupt Clear Register (bits 5to4). Reserved bits - Write only **Value After Reset:** 0x0 **Exists:** Always
3	Clear_SLVIF_CommonReg_WrOnHold_ERR_IntStat	W	Register Bus 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	Register Bus 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

Bits	Name	Memory Access	Description
1	Clear_SLVIF_CommonReg_WR2RO_ERR_IntStat	W	Register Bus 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	Register Bus 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

6.1.11 DMAC_COMMONREG_INTSTATUS_ENABLEREG

- **Name:** DW_axi_dmac Common Register Space Interrupt Enable Register
- **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

63:31	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_COMMONREG_INTSTATUS_ENABLEREG_63to31	Enable_MXIF2_BCH_PROT_BVALID_ParErr_IntStat	Enable_MXIF2_RCH_PROT_RVALID_ParErr_IntStat	Enable_MXIF2_WCH_PROT_WREADY_ParErr_IntStat	Enable_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	Enable_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	Enable_MXIF1_BCH_PROT_BVALID_ParErr_IntStat	Enable_MXIF1_RCH_PROT_RVALID_ParErr_IntStat	Enable_MXIF1_WCH_PROT_WREADY_ParErr_IntStat	Enable_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	Enable_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	Enable_MXIF2_BCH_PROT_UnCorrERR_IntStat	Enable_MXIF2_BCH_PROT_CorrERR_IntStat	Enable_MXIF2_RCH1_PROT_UnCorrERR_IntStat	Enable_MXIF2_RCH1_PROT_CorrERR_IntStat	Enable_MXIF2_RCH0_PROT_UnCorrERR_IntStat	Enable_MXIF2_RCH0_PROT_CorrERR_IntStat	Enable_MXIF1_BCH_PROT_UnCorrERR_IntStat	Enable_MXIF1_BCH_PROT_CorrERR_IntStat	Enable_MXIF1_RCH1_PROT_UnCorrERR_IntStat	Enable_MXIF1_RCH1_PROT_CorrERR_IntStat	Enable_MXIF1_RCH0_PROT_UnCorrERR_IntStat	Enable_MXIF1_RCH0_PROT_CorrERR_IntStat	Enable_SLVIF_UndefinedReg_DEC_ERR_IntStat	Enable_SLVIF_CommonReg_WRPARITY_ERR_IntStat	Enable_SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	RSVD_DMAC_COMMONREG_INTSTATUS_ENABLEREG_5to4	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntStat	Enable_SLVIF_CommonReg_RD2WO_ERR_IntStat	Enable_SLVIF_CommonReg_WR2RO_ERR_IntStat	Enable_SLVIF_CommonReg_DEC_ERR_IntStat

Table 6-15 Fields for Register: DMAC_COMMONREG_INTSTATUS_ENABLEREG

Bits	Name	Memory Access	Description
63:31	RSVD_DMAC_COMMONREG_INTSTATUS_ENABLEREG_63to31	R	DMAC Common Register Interrupt Status Enable Register (bits 63to31). Reserved bits - Read Only **Value After Reset:** 0x1fffffffff **Exists:** Always

Bits	Name	Memory Access	Description
30	Enable_MXIF2_BCH_PROT_BVALID_ParErr_IntStat	* Varies	AXI Manager Interface 2 Write Response Channel BVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_BVALID_ParErr_IntStat): MXIF2_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_BCH_PROT_BVALID_ParErr_IntStat): MXIF2_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
29	Enable_MXIF2_RCH_PROT_RVALID_ParErr_IntStat	* Varies	AXI Manager Interface 2 Read Data Channel RVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_RCH_PROT_RVALID_ParErr_IntStat): MXIF2_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_RCH_PROT_RVALID_ParErr_IntStat): MXIF2_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
28	Enable_MXIF2_WCH_PROT_WREADY_ParErr_IntStat	* Varies	AXI Manager Interface 2 Write Data Channel WREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_WCH_PROT_WREADY_ParErr_IntStat): MXIF2_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_WCH_PROT_WREADY_ParErr_IntStat): MXIF2_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
27	Enable_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	* Varies	AXI Manager Interface 2 Write Address Channel AWREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
26	Enable_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	* Varies	AXI Manager Interface 2 Read Address Channel ARREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
25	Enable_MXIF1_BCH_PROT_BVALID_ParErr_IntStat	* Varies	AXI Manager Interface 1 Write Response Channel BVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_BVALID_ParErr_IntStat): MXIF1_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_BCH_PROT_BVALID_ParErr_IntStat): MXIF1_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
24	Enable_MXIF1_RCH_PROT_RVALID_ParErr_IntStat	* Varies	AXI Manager Interface 1 Read Data Channel RVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_RCH_PROT_RVALID_ParErr_IntStat): MXIF1_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_RCH_PROT_RVALID_ParErr_IntStat): MXIF1_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}
23	Enable_MXIF1_WCH_PROT_WREADY_ParErr_IntStat	* Varies	AXI Manager Interface 1 Write Data Channel WREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_WCH_PROT_WREADY_ParErr_IntStat): MXIF1_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_WCH_PROT_WREADY_ParErr_IntStat): MXIF1_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
22	Enable_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	* Varies	AXI Manager Interface 1 Write Address Channel AWREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}
21	Enable_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	* Varies	AXI Manager Interface 1 Read Address Channel ARREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
20	Enable_MXIF2_BCH_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 2 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_UnCorrERR_IntStat): MXIF2_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_BCH_PROT_UnCorrERR_IntStat): MXIF2_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
19	Enable_MXIF2_BCH_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 2 Write Response Channel ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_CorrERR_IntStat): MXIF2_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_BCH_PROT_CorrERR_IntStat): MXIF2_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
18	Enable_MXIF2_RCH1_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 2 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_RCH1_PROT_UnCorrERR_IntStat): MXIF2_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_RCH1_PROT_UnCorrERR_IntStat): MXIF2_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
17	Enable_MXIF2_RCH1_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 2 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH1_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_RCH1_PROT_CorrERR_IntStat): MXIF2_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_RCH1_PROT_CorrERR_IntStat): MXIF2_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
16	Enable_MXIF2_RCH0_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 2 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_RCH0_PROT_UnCorrERR_IntStat): MXIF2_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_RCH0_PROT_UnCorrERR_IntStat): MXIF2_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
15	Enable_MXIF2_RCH0_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 2 Read Channel (Data) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF2_RCH0_PROT_CorrERR_IntStat): MXIF2_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF2_RCH0_PROT_CorrERR_IntStat): MXIF2_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
14	Enable_MXIF1_BCH_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 1 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_UnCorrERR_IntStat): MXIF1_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_BCH_PROT_UnCorrERR_IntStat): MXIF1_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
13	Enable_MXIF1_BCH_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 1 Write Response Channel ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_CorrERR_IntStat): MXIF1_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_BCH_PROT_CorrERR_IntStat): MXIF1_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
12	Enable_MXIF1_RCH1_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 1 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_RCH1_PROT_UnCorrERR_IntStat): MXIF1_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_RCH1_PROT_UnCorrERR_IntStat): MXIF1_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
11	Enable_MXIF1_RCH1_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 1 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_RCH1_PROT_CorrERR_IntStat): MXIF1_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_RCH1_PROT_CorrERR_IntStat): MXIF1_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
10	Enable_MXIF1_RCH0_PROT_UnCorrERR_IntStat	* Varies	AXI Manager Interface 1 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_RCH0_PROT_UnCorrERR_IntStat): MXIF1_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_RCH0_PROT_UnCorrERR_IntStat): MXIF1_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
9	Enable_MXIF1_RCH0_PROT_CorrERR_IntStat	* Varies	AXI Manager Interface 1 Read Channel (Data) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg). **Values:** ■ 0x1 (ENABLE_MXIF1_RCH0_PROT_CorrERR_IntStat): MXIF1_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_MXIF1_RCH0_PROT_CorrERR_IntStat): MXIF1_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
8	Enable_SLVIF_UndefinedReg_DEC_ERR_IntStat	R/W	Register Bus 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	Enable_SLVIF_CommonReg_WRPARITY_ERR_IntStat	* Varies	Register Bus Interface Common Register Write Parity Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_WRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WRPARITY_ERR): SLVIF_CommonReg_WRPARITY_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_WRPARITY_ERR): SLVIF_CommonReg_WRPARITY_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
6	Enable_SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	* Varies	Register Bus Interface Common Register Address Parity Error Interrupt Status Enable Bit. This bit is used to enable the corresponding channel interrupt status bit (SLVIF_CommonReg_ADDRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_ADDRPARITY_ERR): SLVIF_CommonReg_ADDRPARITY_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled ■ 0x0 (DISABLE_SLVIF_CommonReg_ADDRPARITY_ERR): SLVIF_CommonReg_ADDRPARITY_ERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Disabled **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ADDR_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
5:4	RSVD_DMAC_COMMONREG_INTSTATUS_ENABLEREG_5to4	R	DMAC Common Register Interrupt Status Enable Register (bits 5to4). Reserved bits - Read Only **Value After Reset:** 0x3 **Exists:** Always
3	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntStat	R/W	Register Bus 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
2	Enable_SLVIF_CommonReg_RD2WO_ERR_IntStat	R/W	Register Bus 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	Register Bus 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

Bits	Name	Memory Access	Description
0	Enable_SLVIF_CommonReg_DEC_ERR_IntStat	R/W	Register Bus 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

6.1.12 DMAC_COMMONREG_INTSIGNAL_ENABLEREG

- **Name:** DW_axi_dmac Common Register Space Interrupt Signal Enable Register
- **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

63:31	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_COMMONREG_INTSIGNAL_ENABLEREG_63to31	Enable_MXIF2_BCH_PROT_BVALID_ParErr_IntSignal	Enable_MXIF2_RCH_PROT_RVALID_ParErr_IntSignal	Enable_MXIF2_WCH_PROT_WREADY_ParErr_IntSignal	Enable_MXIF2_AWCH_PROT_AWREADY_ParErr_IntSignal	Enable_MXIF2_ARCH_PROT_ARREADY_ParErr_IntSignal	Enable_MXIF1_BCH_PROT_BVALID_ParErr_IntSignal	Enable_MXIF1_RCH_PROT_RVALID_ParErr_IntSignal	Enable_MXIF1_WCH_PROT_WREADY_ParErr_IntSignal	Enable_MXIF1_AWCH_PROT_AWREADY_ParErr_IntSignal	Enable_MXIF1_ARCH_PROT_ARREADY_ParErr_IntSignal	Enable_MXIF2_BCH_PROT_UnCorrERR_IntSignal	Enable_MXIF2_BCH_PROT_CorrERR_IntSignal	Enable_MXIF2_RCH1_PROT_UnCorrERR_IntSignal	Enable_MXIF2_RCH1_PROT_CorrERR_IntSignal	Enable_MXIF2_RCH0_PROT_UnCorrERR_IntSignal	Enable_MXIF2_RCH0_PROT_CorrERR_IntSignal	Enable_MXIF1_BCH_PROT_UnCorrERR_IntSignal	Enable_MXIF1_BCH_PROT_CorrERR_IntSignal	Enable_MXIF1_RCH1_PROT_UnCorrERR_IntSignal	Enable_MXIF1_RCH1_PROT_CorrERR_IntSignal	Enable_MXIF1_RCH0_PROT_UnCorrERR_IntSignal	Enable_MXIF1_RCH0_PROT_CorrERR_IntSignal	Enable_SLVIF_UndefinedReg_DEC_ERR_IntSignal	Enable_SLVIF_CommonReg_WRPARITY_ERR_IntSignal	Enable_SLVIF_CommonReg_ADDRPARITY_ERR_IntSignal	RSVD_DMAC_COMMONREG_INTSIGNAL_ENABLEREG_5to4	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntSignal	Enable_SLVIF_CommonReg_RD2WO_ERR_IntSignal	Enable_SLVIF_CommonReg_WR2RO_ERR_IntSignal	Enable_SLVIF_CommonReg_DEC_ERR_IntSignal

Table 6-16 Fields for Register: DMAC_COMMONREG_INTSIGNAL_ENABLEREG

Bits	Name	Memory Access	Description
63:31	RSVD_DMAC_COMMONREG_INTSIGNAL_ENABLEREG_63to31	R	DMAC Common Register Interrupt Signal Enable Register (bits 63to31) Reserved bits - Read Only **Value After Reset:** 0x1fffffffff **Exists:** Always

Bits	Name	Memory Access	Description
30	Enable_MXIF2_BCH_PROT_BVALID_ParErr_IntSignal	* Varies	AXI Manager Interface 2 Write Response Channel BVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_BVALID_ParErr_IntStat): MXIF2_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_BCH_PROT_BVALID_ParErr_IntStat): MXIF2_BCH_PROT_BVALID_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
29	Enable_MXIF2_RCH_PROT_RVALID_ParErr_IntSignal	* Varies	AXI Manager Interface 2 Read Data Channel RVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_RCH_PROT_RVALID_ParErr_IntStat): MXIF2_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_RCH_PROT_RVALID_ParErr_IntStat): MXIF2_RCH_PROT_RVALID_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
28	Enable_MXIF2_WCH_PROT_WREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 2 Write Data Channel WREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_WCH_PROT_WREADY_ParErr_IntStat): MXIF2_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_WCH_PROT_WREADY_ParErr_IntStat): MXIF2_WCH_PROT_WREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
27	Enable_MXIF2_AWCH_PROT_AWREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 2 Write Address Channel AWREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
26	Enable_MXIF2_ARCH_PROT_ARREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 2 Read Address Channel ARREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1 && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
25	Enable_MXIF1_BCH_PROT_BVALID_ParErr_IntSignal	* Varies	AXI Manager Interface 1 Write Response Channel BVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_BVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_BVALID_ParErr_IntStat): MXIF1_BCH_PROT_BVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_BCH_PROT_BVALID_ParErr_IntStat): MXIF1_BCH_PROT_BVALID_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
24	Enable_MXIF1_RCH_PROT_RVALID_ParErr_IntSignal	* Varies	AXI Manager Interface 1 Read Data Channel RVALID Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH_PROT_RVALID_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_RCH_PROT_RVALID_ParErr_IntStat): MXIF1_RCH_PROT_RVALID_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_RCH_PROT_RVALID_ParErr_IntStat): MXIF1_RCH_PROT_RVALID_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}
23	Enable_MXIF1_WCH_PROT_WREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 1 Write Data Channel WREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_WCH_PROT_WREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_WCH_PROT_WREADY_ParErr_IntStat): MXIF1_WCH_PROT_WREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_WCH_PROT_WREADY_ParErr_IntStat): MXIF1_WCH_PROT_WREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
22	Enable_MXIF1_AWCH_PROT_AWREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 1 Write Address Channel AWREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat): MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}
21	Enable_MXIF1_ARCH_PROT_ARREADY_ParErr_IntSignal	* Varies	AXI Manager Interface 1 Read Address Channel ARREADY Parity Protection related Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat): MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_MXIF_READY_VALID_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
20	Enable_MXIF2_BCH_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 2 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_UnCorrERR_IntStat): MXIF2_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_BCH_PROT_UnCorrERR_IntStat): MXIF2_BCH_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
19	Enable_MXIF2_BCH_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 2 Write Response Channel ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_BCH_PROT_CorrERR_IntStat): MXIF2_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_BCH_PROT_CorrERR_IntStat): MXIF2_BCH_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
18	Enable_MXIF2_RCH1_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 2 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_RCH1_PROT_UnCorrERR_IntStat): MXIF2_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_RCH1_PROT_UnCorrERR_IntStat): MXIF2_RCH1_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
17	Enable_MXIF2_RCH1_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 2 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCHPROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_RCH1_PROT_CorrERR_IntStat): MXIF2_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_RCH1_PROT_CorrERR_IntStat): MXIF2_RCH1_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
16	Enable_MXIF2_RCH0_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 2 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_RCH0_PROT_UnCorrERR_IntStat): MXIF2_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_RCH0_PROT_UnCorrERR_IntStat): MXIF2_RCH0_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}
15	Enable_MXIF2_RCH0_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 2 Read Channel (Data) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF2_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF2_RCH0_PROT_CorrERR_IntStat): MXIF2_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF2_RCH0_PROT_CorrERR_IntStat): MXIF2_RCH0_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {((DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) && (DMAX_NUM_MASTER_IF > 1)) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
14	Enable_MXIF1_BCH_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 1 Write Response Channel Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_UnCorrERR_IntStat): MXIF1_BCH_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_BCH_PROT_UnCorrERR_IntStat): MXIF1_BCH_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
13	Enable_MXIF1_BCH_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 1 Write Response Channel ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_BCH_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_BCH_PROT_CorrERR_IntStat): MXIF1_BCH_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_BCH_PROT_CorrERR_IntStat): MXIF1_BCH_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
12	Enable_MXIF1_RCH1_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 1 Read Channel (Other Control signals) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_RCH1_PROT_UnCorrERR_IntStat): MXIF1_RCH1_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_RCH1_PROT_UnCorrERR_IntStat): MXIF1_RCH1_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
11	Enable_MXIF1_RCH1_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 1 Read Channel (Other Control signals) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH1_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_RCH1_PROT_CorrERR_IntStat): MXIF1_RCH1_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_RCH1_PROT_CorrERR_IntStat): MXIF1_RCH1_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
10	Enable_MXIF1_RCH0_PROT_UnCorrERR_IntSignal	* Varies	AXI Manager Interface 1 Read Channel (Data) Parity/ECC Protection related Uncorrectable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_UnCorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_RCH0_PROT_UnCorrERR_IntStat): MXIF1_RCH0_PROT_UnCorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_RCH0_PROT_UnCorrERR_IntStat): MXIF1_RCH0_PROT_UnCorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}
9	Enable_MXIF1_RCH0_PROT_CorrERR_IntSignal	* Varies	AXI Manager Interface 1 Read Channel (Data) ECC Protection related Correctable Error Interrupt Status bit. This bit is used to enable the corresponding channel interrupt status bit (MXIF1_RCH0_PROT_CorrERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_MXIF1_RCH0_PROT_CorrERR_IntStat): MXIF1_RCH0_PROT_CorrERR_IntStat bit in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_MXIF1_RCH0_PROT_CorrERR_IntStat): MXIF1_RCH0_PROT_CorrERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MXIF_EN == 1 || DMAX_MXIF_ECC_PAR_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
8	Enable_SLVIF_UndefinedReg_DEC_ERR_IntSignal	R/W	Register Bus 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	Enable_SLVIF_CommonReg_WRPARITY_ERR_IntSignal	* Varies	Register Bus Interface Write Parity Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit (SLVIF_CommonReg_WRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_WRPARITY_ERR_IntSignal): SLVIF_CommonReg_WRPARITY_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_WRPARITY_ERR_IntSignal): SLVIF_CommonReg_WRPARITY_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
6	Enable_SLVIF_CommonReg_ADDRPARITY_ERR_IntSignal	* Varies	Register Bus Interface Address Parity Error Interrupt Signal Enable Bit. This bit is used to enable the propagation of corresponding channel interrupt status bit (SLVIF_CommonReg_ADDRPARITY_ERR_IntStat in DMAC_CommonReg_IntStatusReg) to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_SLVIF_CommonReg_ADDRPARITY_ERR_IntSignal): SLVIF_CommonReg_ADDRPARITY_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Enabled at port level ■ 0x0 (DISABLE_SLVIF_CommonReg_ADDRPARITY_ERR_IntSignal): SLVIF_CommonReg_ADDRPARITY_ERR_IntStat signal in DMAC_CommonReg_IntStatusReg is Disabled at port level **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ADDR_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
5:4	RSVD_DMAC_COMMONREG_INTSIGNAL_ENABLEREG_5to4	R	DMAC Common Register Interrupt Signal Enable Register (bits 5to4). Reserved bits - Read Only **Value After Reset:** 0x3 **Exists:** Always
3	Enable_SLVIF_CommonReg_WrOnHold_ERR_IntSignal	R/W	Register Bus 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

Bits	Name	Memory Access	Description
2	Enable_SLVIF_CommonReg_RD2WO_ERR_IntSignal	R/W	Register Bus 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
1	Enable_SLVIF_CommonReg_WR2RO_ERR_IntSignal	R/W	Register Bus 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

Bits	Name	Memory Access	Description
0	Enable_SLVIF_CommonReg_DEC_ERR_IntSignal	R/W	Register Bus 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

6.1.13 DMAC_COMMONREG_INTSTATUSREG

- **Name:** DW_axi_dmac Common Register Space Interrupt Status Register
- **Description:** This Register captures Register bus interface access errors.
  - Decode Error.
  - Write to read only register.
  - Read to write only register.
  - write on hold.
  - undefined address.
  - Common Register Space - Write Parity Error
- **Size:** 64 bits
- **Offset:** 0x50
- **Exists:** Always

63:31	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_COMMONREG_INTSTATUSREG_63to31	MXIF2_BCH_PROT_BVALID_ParErr_IntStat	MXIF2_RCH_PROT_RVALID_ParErr_IntStat	MXIF2_WCH_PROT_WREADY_ParErr_IntStat	MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	MXIF1_BCH_PROT_BVALID_ParErr_IntStat	MXIF1_RCH_PROT_RVALID_ParErr_IntStat	MXIF1_WCH_PROT_WREADY_ParErr_IntStat	MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	MXIF2_BCH_PROT_UnCorrERR_IntStat	MXIF2_BCH_PROT_CorrERR_IntStat	MXIF2_RCH1_PROT_UnCorrERR_IntStat	MXIF2_RCH1_PROT_CorrERR_IntStat	MXIF2_RCH0_PROT_UnCorrERR_IntStat	MXIF2_RCH0_PROT_CorrERR_IntStat	MXIF1_BCH_PROT_UnCorrERR_IntStat	MXIF1_BCH_PROT_CorrERR_IntStat	MXIF1_RCH1_PROT_UnCorrERR_IntStat	MXIF1_RCH1_PROT_CorrERR_IntStat	MXIF1_RCH0_PROT_UnCorrERR_IntStat	MXIF1_RCH0_PROT_CorrERR_IntStat	SLVIF_UndefinedReg_DEC_ERR_IntStat	SLVIF_CommonReg_WRPARITY_ERR_IntStat	SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	RSVD_DMAC_COMMONREG_INTSTATUSREG_5to4	SLVIF_CommonReg_WrOnHold_ERR_IntStat	SLVIF_CommonReg_RD2WO_ERR_IntStat	SLVIF_CommonReg_WR2RO_ERR_IntStat	SLVIF_CommonReg_DEC_ERR_IntStat

Table 6-17 Fields for Register: DMAC_COMMONREG_INTSTATUSREG

Bits	Name	Memory Access	Description
63:31	RSVD_DMAC_COMMONREG_INTSTATUSREG_63to31	R	DMAC Common Register Interrupt Signal Enable Register (bits 63to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
30	MXIF2_BCH_PROT_BVALID_ParErr_IntStat	R	AXI Manager Interface 2 Write Response Channel BVALID Parity Protection Error Interrupt Status bit. This error occurs if BVALID Parity error is detected on the Write Response AXI Channel. ■ 0: No AXI Manager 2 Write Response Channel BVALID Parity Error. ■ 1: AXI Manager 2 Write Response Channel BVALID Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_BVALID_ParErr): AXI Manager 2 Write Response Channel BVALID Parity Error detected ■ 0x0 (Inactive_PROT_BVALID_ParErr): AXI Manager 2 Write Response Channel BVALID Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
29	MXIF2_RCH_PROT_RVALID_ParErr_IntStat	R	AXI Manager Interface 2 Read Data Channel RVALID Parity Protection Error Interrupt Status bit. This error occurs if RVALID Parity error is detected on the Read Data AXI Channel. ■ 0: No AXI Manager 2 Read Data Channel RVALID Parity Error. ■ 1: AXI Manager 2 Read Data Channel RVALID Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_RVALID_ParErr): AXI Manager 2 Read Data Channel RVALID Parity Error detected ■ 0x0 (Inactive_PROT_RVALID_ParErr): AXI Manager 2 Read Data Channel RVALID Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
28	MXIF2_WCH_PROT_WREADY_ParErr_IntStat	R	AXI Manager Interface 2 Write Data Channel WREADY Parity Protection Error Interrupt Status bit. This error occurs if WREADY Parity error is detected on the Write Data AXI Channel. ■ 0: No AXI Manager 2 Write Data Channel WREADY Parity Error. ■ 1: AXI Manager 2 Write Data Channel WREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_WREADY_ParErr): AXI Manager 2 Write Data Channel WREADY Parity Error detected ■ 0x0 (Inactive_PROT_WREADY_ParErr): AXI Manager 2 Write Data Channel WREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
27	MXIF2_AWCH_PROT_AWREADY_ParErr_IntStat	R	AXI Manager Interface 2 Write Address Channel AWREADY Parity Protection Error Interrupt Status bit. This error occurs if AWREADY Parity error is detected on the Write Address AXI Channel. ■ 0: No AXI Manager 2 Write Address Channel AWREADY Parity Error. ■ 1: AXI Manager 2 Write Address Channel AWREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_AWREADY_ParErr): AXI Manager 2 Write Address Channel AWREADY Parity Error detected ■ 0x0 (Inactive_PROT_AWREADY_ParErr): AXI Manager 2 Write Address Channel AWREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
26	MXIF2_ARCH_PROT_ARREADY_ParErr_IntStat	R	AXI Manager Interface 2 Read Address Channel ARREADY Parity Protection Error Interrupt Status bit. This error occurs if ARREADY Parity error is detected on the Read Address AXI Channel. ■ 0: No AXI Manager 2 Read Address Channel ARREADY Parity Error. ■ 1: AXI Manager 2 Read Address Channel ARREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_ARREADY_ParErr): AXI Manager 2 Read Address Channel ARREADY Parity Error detected ■ 0x0 (Inactive_PROT_ARREADY_ParErr): AXI Manager 2 Read Address Channel ARREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
25	MXIF1_BCH_PROT_BVALID_ParErr_IntStat	R	AXI Manager Interface 1 Write Response Channel BVALID Parity Protection Error Interrupt Status bit. This error occurs if BVALID Parity error is detected on the Write Response AXI Channel. ■ 0: No AXI Manager 1 Write Response Channel BVALID Parity Error. ■ 1: AXI Manager 1 Write Response Channel BVALID Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_BVALID_ParErr): AXI Manager 1 Write Response Channel BVALID Parity Error detected ■ 0x0 (Inactive_PROT_BVALID_ParErr): AXI Manager 1 Write Response Channel BVALID Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
24	MXIF1_RCH_PROT_RVALID_ParErr_IntStat	R	AXI Manager Interface 1 Read Data Channel RVALID Parity Protection Error Interrupt Status bit. This error occurs if RVALID Parity error is detected on the Read Data AXI Channel. ■ 0: No AXI Manager 1 Read Data Channel RVALID Parity Error. ■ 1: AXI Manager 1 Read Data Channel RVALID Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_RVALID_ParErr): AXI Manager 1 Read Data Channel RVALID Parity Error detected ■ 0x0 (Inactive_PROT_RVALID_ParErr): AXI Manager 1 Read Data Channel RVALID Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
23	MXIF1_WCH_PROT_WREADY_ParErr_IntStat	R	AXI Manager Interface 1 Write Data Channel WREADY Parity Protection Error Interrupt Status bit. This error occurs if WREADY Parity error is detected on the Write Data AXI Channel. ■ 0: No AXI Manager 1 Write Data Channel WREADY Parity Error. ■ 1: AXI Manager 1 Write Data Channel WREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_WREADY_ParErr): AXI Manager 1 Write Data Channel WREADY Parity Error detected ■ 0x0 (Inactive_PROT_WREADY_ParErr): AXI Manager 1 Write Data Channel WREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
22	MXIF1_AWCH_PROT_AWREADY_ParErr_IntStat	R	AXI Manager Interface 1 Write Address Channel AWREADY Parity Protection Error Interrupt Status bit. This error occurs if AWREADY Parity error is detected on the Write Address AXI Channel. ■ 0: No AXI Manager 1 Write Address Channel AWREADY Parity Error. ■ 1: AXI Manager 1 Write Address Channel AWREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_AWREADY_ParErr): AXI Manager 1 Write Address Channel AWREADY Parity Error detected ■ 0x0 (Inactive_PROT_AWREADY_ParErr): AXI Manager 1 Write Address Channel AWREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
21	MXIF1_ARCH_PROT_ARREADY_ParErr_IntStat	R	AXI Manager Interface 1 Read Address Channel ARREADY Parity Protection Error Interrupt Status bit. This error occurs if ARREADY Parity error is detected on the Read Address AXI Channel. ■ 0: No AXI Manager 1 Read Address Channel ARREADY Parity Error. ■ 1: AXI Manager 1 Read Address Channel ARREADY Parity Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_ARREADY_ParErr): AXI Manager 1 Read Address Channel ARREADY Parity Error detected ■ 0x0 (Inactive_PROT_ARREADY_ParErr): AXI Manager 1 Read Address Channel ARREADY Parity Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
20	MXIF2_BCH_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 2 Write Response Channel Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Write Response payload. ■ 0: No AXI Manager 2 Write Response Channel Uncorrectable Error. ■ 1: AXI Manager 2 Write Response Channel Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 2 Write Response Channel Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 2 Write Response Channel Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
19	MXIF2_BCH_PROT_CorrERR_IntStat	R	AXI Manager Interface 2 Write Response Channel ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Write Response payload. ■ 0: No AXI Manager 2 Write Response Channel Correctable Error. ■ 1: AXI Manager 2 Write Response Channel Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 2 Write Response Channel Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 2 Write Response Channel Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
18	MXIF2_RCH1_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 1 Read Channel (Other Control signals) Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Read Control payload. ■ 0: No AXI Manager 2 Read Channel Control signals related Uncorrectable Error. ■ 1: AXI Manager 2 Read Channel Control signals related Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 2 Read Channel (Other Control signals) Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 2 Read Channel (Other Control signals) Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
17	MXIF2_RCH1_PROT_CorrERR_IntStat	R	AXI Manager Interface 2 Read Channel (Other Control signals) ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Read Control payload. ■ 0: No AXI Manager 2 Read Channel Control signals related Correctable Error. ■ 1: AXI Manager 2 Read Channel Control signals related Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 2 Read Channel (Other Control signals) Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 2 Read Channel (Other Control signals) Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
16	MXIF2_RCH0_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 2 Read Channel (Data) Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Read Data payload. ■ 0: No AXI Manager 2 Read Channel Data related Uncorrectable Error. ■ 1: AXI Manager 2 Read Channel Data related Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 2 Read Channel (Data) Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 2 Read Channel (Data) Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
15	MXIF2_RCH0_PROT_CorrERR_IntStat	R	AXI Manager Interface 2 Read Channel (Data) ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Read Data payload. ■ 0: No AXI Manager 2 Read Channel Data related Correctable Error. ■ 1: AXI Manager 2 Read Channel Data related Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 2 Read Channel (Data) Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 2 Read Channel (Data) Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
14	MXIF1_BCH_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 1 Write Response Channel Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Write Response payload. ■ 0: No AXI Manager 1 Write Response Channel Uncorrectable Error. ■ 1: AXI Manager 1 Write Response Channel Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 1 Write Response Channel Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 1 Write Response Channel Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
13	MXIF1_BCH_PROT_CorrERR_IntStat	R	AXI Manager Interface 1 Write Response Channel ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Write Response payload. ■ 0: No AXI Manager 1 Write Response Channel Correctable Error. ■ 1: AXI Manager 1 Write Response Channel Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 1 Write Response Channel Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 1 Write Response Channel Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
12	MXIF1_RCH1_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 1 Read Channel (Other Control signals) Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Read Control payload. ■ 0: No AXI Manager 1 Read Channel Control signals related Uncorrectable Error. ■ 1: AXI Manager 1 Read Channel Control signals related Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 1 Read Channel (Other Control signals) Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 1 Read Channel (Other Control signals) Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
11	MXIF1_RCH1_PROT_CorrERR_IntStat	R	AXI Manager Interface 1 Read Channel (Other Control signals) ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Read Control payload. ■ 0: No AXI Manager 1 Read Channel Control signals related Correctable Error. ■ 1: AXI Manager 1 Read Channel Control signals related Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 1 Read Channel (Other Control signals) Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 1 Read Channel (Other Control signals) Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
10	MXIF1_RCH0_PROT_UnCorrERR_IntStat	R	AXI Manager Interface 1 Read Channel (Data) Parity/ECC Protection Uncorrectable Error Interrupt Status bit. This error occurs if Parity/ECC Uncorrectable error is detected on the AXI Channel Read Data payload. ■ 0: No AXI Manager 1 Read Channel Data related Uncorrectable Error. ■ 1: AXI Manager 1 Read Channel Data related Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_UnCorrERR): AXI Manager 1 Read Channel (Data) Uncorrectable Error detected ■ 0x0 (Inactive_PROT_UnCorrERR): AXI Manager 1 Read Channel (Data) Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
9	MXIF1_RCH0_PROT_CorrERR_IntStat	R	AXI Manager Interface 1 Read Channel (Data) ECC Protection Correctable Error Interrupt Status bit. This error occurs if ECC Correctable error is detected on the AXI Channel Read Data payload. ■ 0: No AXI Manager 1 Read Channel Data related Correctable Error. ■ 1: AXI Manager 1 Read Channel Data related Correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in DMAC_COMMONREG_INTSTATUS_ENABLEREG register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in DMAC_COMMONREG_INTCLEARREG. **Values:** ■ 0x1 (Active_PROT_CorrERR): AXI Manager 1 Read Channel (Data) Correctable Error detected ■ 0x0 (Inactive_PROT_CorrERR): AXI Manager 1 Read Channel (Data) Correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
8	SLVIF_UndefinedReg_DEC_ERR_IntStat	R	Register Bus 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 register bus interface. ■ 0: No Register Bus Interface Decode Errors. ■ 1: Register Bus 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): Register Bus Interface Decode Error detected ■ 0x0 (Inactive_UndefinedReg_DEC_ERR): No Register Bus Interface Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
7	SLVIF_CommonReg_WRPARITY_ERR_IntStat	R	Register Bus Interface Common Register Write Parity Error Interrupt Status Bit. This error occurs if write operation with data and parity is performed to a register in the common register space, but the locally computed parity bits not matching the received parity bits. ■ 0: No Common Register Space Write Parity Error. ■ 1: Common Register Space Write Parity 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 interrupt clear bit in DMAC_COMMONREG_INTCLEARREG (required when the interrupt is not enabled. **Values:** ■ 0x1 (Active_CommonReg_WRPARITY_ERR): Register Bus Interface Common Register Write Parity Error detected ■ 0x0 (Inactive_CommonReg_WRPARITY_ERR): No Register Bus Interface Common Register Write Parity Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
6	SLVIF_CommonReg_ADDRPARITY_ERR_IntStat	R	Register Bus Interface Common Register Address Parity Error Interrupt Status Bit. This error occurs if address is used to do write/read operation to a register in the common register space, but the locally computed parity bits not matching the received parity bits. ■ 0: No Common Register Space Address Parity Error. ■ 1: Common Register Space Address Parity 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 interrupt clear bit in DMAC_COMMONREG_INTCLEARREG (required when the interrupt is not enabled. **Values:** ■ 0x1 (Active_CommonReg_ADDRPARITY_ERR): Register Bus Interface Common Register Address Parity Error detected ■ 0x0 (Inactive_CommonReg_ADDRPARITY_ERR): No Register Bus Interface Common Register Address Parity Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
5:4	RSVD_DMAC_COMMONREG_INTSTATUSREG_5to4	R	DMAC Common Register Interrupt Status Register (bits 5to4) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
3	SLVIF_CommonReg_WrOnHold_ERR_IntStat	R	Register Bus 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 Register Bus Interface Common Register Write On Hold Errors. ■ 1: Register Bus 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): Register Bus Interface Common Register Write On Hold Error detected ■ 0x0 (Inactive_CommonReg_WrOnHold_ERR): No Register Bus Interface Common Register Write On Hold Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
2	SLVIF_CommonReg_RD2WO_ERR_IntStat	R	Register Bus 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 Register Bus Interface Read to Write Only Errors. ■ 1: Register Bus 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): Register Bus Interface Read to Write Only Error detected ■ 0x0 (Inactive_CommonReg_RD2WO_ERR): No Register Bus Interface Read to Write Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
1	SLVIF_CommonReg_WR2RO_ERR_IntStat	R	Register Bus 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 Register Bus Interface Write to Read Only Errors. ■ 1: Register Bus 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 Register Bus Interface Write to Read Only Errors ■ 0x0 (Inactive_CommonReg_WR2RO_ERR): Register Bus Interface Write to Read Only Error detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
0	SLVIF_CommonReg_DEC_ERR_IntStat	R	Register Bus 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 register bus interface. ■ 0: No Register Bus Interface Decode Errors. ■ 1: Register Bus 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): Register Bus Interface Decode Error detected ■ 0x0 (Inactive_CommonReg_DEC_ERR): No Register Bus Interface Decode Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

6.1.14 DMAC_RESETREG

- **Name:** DW_axi_dmac Software Reset Register
- **Description:** This register is used to initiate the Software Reset to DW_axi_dmac.
- **Size:** 64 bits
- **Offset:** 0x58
- **Exists:** Always

Table 6-18 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 register 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

6.1.15 DMAC_LOWPOWER_CFGREG

- **Name:** DW_axi_dmac Low Power Configuration Register
- **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:** DMAX_CSLP_EN == 1

63:56	55:48	47:40	39:32	31:4	3	2	1	0
RSVD_DMAC_LOWPOWER_CFGREG_63to56	MXIF_LPDLY	SBIU_LPDLY	GLCH_LPDLY	RSVD_DMAC_LOWPOWER_CFGREG_31to4	MXIF_CSLP_EN	SBIU_CSLP_EN	CHNL_CSLP_EN	GBL_CSLP_EN

Table 6-19 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

Bits	Name	Memory Access	Description
55:48	MXIF_LPDLY	R/W	Defines the load value to be programmed into the AXI Manager 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_CFG REG_31to4	R	DMAC_LOWPOWER_CFGREG (bits 4to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
3	MXIF_CSLP_EN	* Varies	AXI Manager Interface Context Sensitive Low Power feature enable. **Values:** ■ 0x0 (MXIF_CSLP_DISABLE): AXI Manager Interface Context Sensitive Low Power feature is disabled ■ 0x1 (MXIF_CSLP_ENABLE): AXI Manager Interface Context Sensitive Low Power feature is enabled **Value After Reset:** "(DMAX_MXIF_CSLP_EN==1) ? 0x1 : 0x0" **Exists:** Always **Volatile:** true **Memory Access:** {(DMAX_MXIF_CSLP_EN==1) ? "read-write" : "read-only"}
2	SBIU_CSLP_EN	* Varies	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 **Memory Access:** {(DMAX_SBIU_CSLP_EN==1) ? "read-write" : "read-only"}
1	CHNL_CSLP_EN	* Varies	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 **Memory Access:** {(DMAX_CHNL_CSLP_EN==1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
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

6.1.16 DMAC_COMMON_PARCTLREG

- **Name:** DW_axi_dmac Parity Control Register
- **Description:** This register contains the fields that enables/disables the Parity Protection Feature and a field that selects even or odd parity mode.
- **Size:** 64 bits
- **Offset:** 0x70
- **Exists:** DMAX_PARITY_PROT_EN == 1

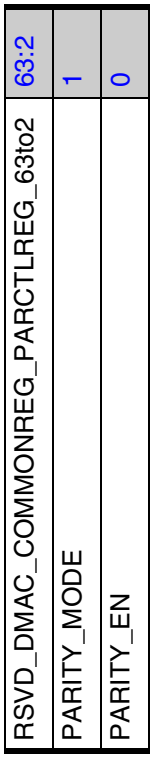


Table 6-20 Fields for Register: DMAC_COMMON_PARCTLREG

Bits	Name	Memory Access	Description
63:2	RSVD_DMAC_COMMONREG_PARCTLREG_63to2	R	DMAC_COMMONREG_PARCTLREG (bits 2to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
1	PARITY_MODE	R/W	Parity Mode, selects the even or odd parity scheme for the Parity Protection feature. **Values:** ■ 0x0 (Even_PARITY_MODE): The Even Parity check is used for registers in the Common/Channel Specific space ■ 0x1 (Odd_PARITY_MODE): The Odd Parity check is used for registers in the Common/Channel Specific space **Value After Reset:** DMAX_DEF_PARITY_MODE **Exists:** Always
0	PARITY_EN	R/W	This field is used to enable/disable the Parity Protection feature. **Values:** ■ 0x0 (Disable_PARITY): The Parity check for registers in the common/channel register space is Disabled ■ 0x1 (Enable_PARITY): The Parity check for registers in the common/channel register space is Enabled **Value After Reset:** DMAX_DEF_PARITY_EN **Exists:** Always

6.1.17 DMAC_COMMON_ECCCTLSTATUSREG

- **Name:** DW_axi_dmac ECC Control and Status Register
- **Description:** This register contains the fields that
  - Reset the ECC Protection Correctable and Uncorrectable Counters.
  - Field that will help to multiplex the above counter values of different DMA Channel FIFO Memory Interfaces, different DMA Channel UID Memory Interfaces, different AXI Manager Interface's Channels.
  - ECC Protection Correctable and Uncorrectable Counter status.
- **Size:** 64 bits
- **Offset:** 0x78
- **Exists:** DMAX_ECC_DIAG_EN == 1

63:42	41:32	31:26	25:16	15:12	11:7	6:5	4	3:0
RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_42to63	ECC_UNCORR_ERR_COUNT	RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_26to31	ECC_CORR_ERR_COUNT	RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_12to15	ECC_INTFC_NO	ECC_INTFC_TYPE	ECC_ERR_COUNT_RST	RSVD_DMAC_ECCCTLSTATUSREG_0to3

Table 6-21 Fields for Register: DMAC_COMMON_ECCCTLSTATUSREG

Bits	Name	Memory Access	Description
63:42	RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_42to63	R	DMAC_COMMONREG_ECCCTLSTATUSREG (bits 42to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
41:32	ECC_UNCORR_ERR_COUNT	R	ECC Uncorrectable Error count This field provides the number of uncorrectable error detected since the ECC Error count reset or soft reset or dmac_resetn. **Value After Reset:** 0x0 **Exists:** Always
31:26	RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_26to31	R	DMAC_COMMONREG_ECCCTLSTATUSREG (bits 26to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
25:16	ECC_CORR_ERR_COUNT	R	ECC Correctable Error count This field provides the number of correctable error detected since the ECC Error count reset or soft reset or dmac_resetn. **Value After Reset:** 0x0 **Exists:** Always
15:12	RSVD_DMAC_COMMONREG_ECCCTLSTATUSREG_12to15	R	DMAC_COMMONREG_ECCCTLSTATUSREG (bits 12to15) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
11:7	ECC_INTFC_NO	R/W	ECC Protected Interface Number This is used to provide the interface number of specific interface type to multiplex the correctable and uncorrectable error counters. The encoding of this field is as follows: ■ When ECC_INTFC_TYPE = 0x0 or 0x1 This field value can vary from 1 to DMAX_NUM_CHANNELS. If illegal value is programmed to this field i.e. value beyond the configured number of channels, then counter will return all zeroes. ■ When ECC_INTFC_TYPE = 0x2 Then this field values can change as mentioned below: ❑ ECC_INTFC_NO[11:10] = 2'b00 ❑ ECC_INTFC_NO[9] = DMAX_NUM_MASTER_IF-1. AXI Manager Interface Number can vary from 0 to DMAX_NUM_MASTER_IF-1. If illegal value is programmed to this field i.e. value beyond the configured number AXI Manager interfaces, then counter will return all zeroes. ❑ ECC_INTFC_NO[8:7] = This field is used to select the specific channel of a AXI Manager Interface. 2'b00: AXI Read Data Channel - RCH0 (ECC counter of Read data), 2'b01: AXI Read Data Channel - RCH1 (ECC counter of Read Data Channel other signals), 2'b10: AXI Write Response Channel - BCH If illegal value is programmed to this field, then counter will return all zeroes. **Value After Reset:** 0x0 **Exists:** Always
6:5	ECC_INTFC_TYPE	R/W	ECC Protected Interace Type This is used to select the required interface that is protected through the ECC. This field is used to multiplex the correctable, uncorrectable error counters for the interfaces types like FIFO Memory Interface, AXI Memory Interface. If the reserved value is programmed, then counter will return all zeroes. **Values:** ■ 0x0 (ECC_CH_MEMIF): Channel FIFO Memory Interface ■ 0x1 (ECC_UID_MEMIF): Channel UID Memory Interface ■ 0x2 (ECC_AXI_MXIF): AXI Manager Interface ■ 0x3 (RSV_ECC_INTFC): Reserved **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
4	ECC_ERR_COUNT_RST	R/W	ECC Error Count Reset This bit is used to reset the ECC Protection Correctable and Uncorrectable Counters. This bit is automatically cleared after all internal ECC Error counters are cleared. **Values:** ■ 0x0 (ECC_COUNT_NORST): Do not Reset the ECC error Counters ■ 0x1 (ECC_COUNT_RST): Reset the ECC error Counters **Value After Reset:** 0x0 **Exists:** Always
3:0	RSVD_DMAC_ECCCTLSTATUSREG_0to3	R	DMAC_COMMONREG_ECCCTLSTATUSREG (bits 0to3) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always

6.1.18 DMAC_SFTY_ERR_INJ_CTRL

- **Name:** DW_axi_dmac Safety Error Injection Control Register
- **Description:** This Register contains the control bits which is used to specify ECC error injection methods supported in DW_axi_dmac.
- **Size:** 64 bits
- **Offset:** 0x80
- **Exists:** DMAX_ECC_ERR_INJ_EN == 1

63:23	22:14	13:11	10	9:y	x:5	4:1	0
RSVD_63_23_SFTY	SFTY_BIT_SEL	ECC_EI_POINTS	ECC_EI_TYPE	RSVD_9_5_SFTY	SFTY_EI_CH	SFTY_EI_FEATURE	SFTY_EI_EN

Table 6-22 Fields for Register: DMAC_SFTY_ERR_INJ_CTRL

Bits	Name	Memory Access	Description
63:23	RSVD_63_23_SFTY	R	DW_axi_dmac Safety Error Injection Control Register (bits 63to23). Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
22:14	SFTY_BIT_SEL	R/W	These bits are programmed to select the bit position to introduce a data path ECC error. **Note**: All the bit locations may not be used for data and ECC Checkbits error injection. The valid values for this field are: ■ Address: 0 to (LOG2_DMAX_CHx_y_DEPTH-1) ■ Data: 0 to (DMAX_CHx_FIFO_WIDTH-1) ■ ECC Checkbits: 0 to (DMAX_CHx_y_ECC_WIDTH-1) Where, x and y correspond to channel number and memory interface (FIFO or UIDMEM), respectively. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
13:11	ECC_EI_POINTS	R/W	This bit field selects the address, data, or ECC checkbits for ECC error injection. **Values:** ■ 0x0 (EI_ADDRESS): Address Error Injection. ■ 0x1 (EI_DATA): Data Error Injection. ■ 0x2 (EI_ECC_CHECKBITS): ECC Checkbits Error Injection. **Value After Reset:** 0x0 **Exists:** Always
10	ECC_EI_TYPE	R/W	Single or double-bit error This bit selects the single or double-bit error injection. This field is active when ECC_EI_EN is 1. The bit position for error insertion is selected using ECC_BIT_SEL field in the same register. The error injection point is specified by ECC_EI_POINTS field. **Note**: For double bit error, bit 0 is enabled by default and ECC_BIT_SEL field is used to specify the second bit position. **Values:** ■ 0x1 (EI_DOUBLE_BIT): Insert double bit error. ■ 0x0 (EI_SINGLE_BIT): Insert single bit error. **Value After Reset:** 0x0 **Exists:** Always
9:y	RSVD_9_5_SFTY	R	DW_axi_dmac Safety Error Injection Control Register (bits 9to5). Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Range Variable[y]:** LOG2_DMAX_NUM_CHANNELS + 5
x:5	SFTY_EI_CH	R/W	These bits select the channel number to inject the ECC error. Where channel number varies from 0 to 31 for FIFO/UID memory interfaces. **Value After Reset:** 0x0 **Exists:** Always **Range Variable[x]:** LOG2_DMAX_NUM_CHANNELS + 4
4:1	SFTY_EI_FEATURE	R/W	These bits select the target memory interface to introduce the ECC error. **Values:** ■ 0x1 (EI_UID_MEM): Select UID memory interface for error Injection. ■ 0x0 (EI_FIFO_MEM): Select FIFO memory interface for error Injection. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
0	SFTY_EI_EN	R/W	This bit enables error injection on FIFO or UID memory interface. The error injection type is specified using the field ECC_EI_TYPE. **Values:** ■ 0x1 (EI_ENABLE): Enable data path Error Injection. ■ 0x0 (EI_DISABLE): Disable data path Error Injection. **Value After Reset:** 0x0 **Exists:** Always

6.2 Channelx_Register_Address_Block Registers

6.2.1 CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Source Address Register
- **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

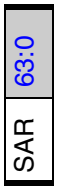


Table 6-23 Fields for Register: CHx_SAR (for x = 1; x <= DMAX_NUM_CHANNELS)

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

6.2.2 CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Destination Address Register
- **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

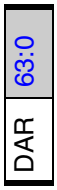


Table 6-24 Fields for Register: CHx_DAR (for x = 1; x <= DMAX_NUM_CHANNELS)

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

6.2.3 CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Block Transfer Size Register
- **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

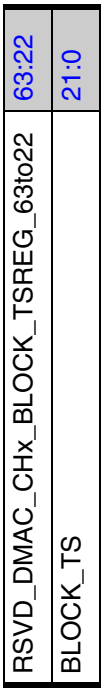


Table 6-25 Fields for Register: CHx_BLOCK_TS (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:22	RSVD_DMAC_CHx_BLOCK_TSREG_63to22	R	DMAC Channelx Block Transfer Size Register (bits 63to22) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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 **Note**: The value programmed to the BLOCK_TS field must be greater than or equal to the source or destination transfer width and the read back value of BLOCK_TS field is always the value programmed into it. This register field is not applicable when source or destination is flow controller and it is applicable when DMA is flow controller. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

6.2.4 CHx_CTL (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Control Register
- **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 register bus 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 6-26 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 subordinate 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.

Bits	Name	Memory Access	Description
63...(cont.)	SHADOWREG_OR_LLI_VALID.	R/W	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

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

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 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

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 peripherals), and during 4K boundary crossings. **Value After Reset:** 0x0 **Exists:** Always
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

Bits	Name	Memory Access	Description
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
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

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

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 write to Destination in the burst transaction ■ 0x1 (DATA_ITEMS_4): 4 Data Item write to Destination in the burst transaction ■ 0x2 (DATA_ITEMS_8): 8 Data Item write to Destination in the burst transaction ■ 0x3 (DATA_ITEMS_16): 16 Data Item write to Destination in the burst transaction ■ 0x4 (DATA_ITEMS_32): 32 Data Item write to Destination in the burst transaction ■ 0x5 (DATA_ITEMS_64): 64 Data Item write to Destination in the burst transaction ■ 0x6 (DATA_ITEMS_128): 128 Data Item write to Destination in the burst transaction ■ 0x7 (DATA_ITEMS_256): 256 Data Item write to Destination in the burst transaction ■ 0x8 (DATA_ITEMS_512): 512 Data Item write to Destination in the burst transaction ■ 0x9 (DATA_ITEMS_1024): 1024 Data Item write to Destination in the burst transaction **Value After Reset:** 0x0 **Exists:** Always

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
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 ■ 0x7 (BITS_1024): Destination Transfer Width is 1024 bits **Value After Reset:** {(DMAX_CH(x)_DTW_ENC < 8) ? DMAX_CH(x)_DTW_ENC : 2} **Exists:** Always

Bits	Name	Memory Access	Description
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 ■ 0x7 (BITS_1024): Source Transfer Width is 1024 bits **Value After Reset:** {(DMAX_CH(x)_STW_ENC < 8) ? 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
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

Bits	Name	Memory Access	Description
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
2	DMS	{(DMAX_CH(x)_DMS == 2) ? "read-write" : "read-only"}	Destination Manager Select. Identifies the Manager Interface layer from which the destination device (peripheral or memory) is accessed. ■ 0: AXI manager 1 ■ 1: AXI Manager 2 **Values:** ■ 0x1 (MASTER2_INTF): Destination device on Manager-2 interface layer ■ 0x0 (MASTER1_INTF): Destination device on Manager-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

Bits	Name	Memory Access	Description
0	SMS	{(DMAX_CH(x)_SMS == 2) ? "read-write" : "read-only"}	Source Manager Select. Identifies the Manager Interface layer from which the source device (peripheral or memory) is accessed. ■ 0: AXI manager 1 ■ 1: AXI Manager 2 **Values:** ■ 0x1 (MASTER2_INTF): Source device on Manager-2 interface layer ■ 0x0 (MASTER1_INTF): Source device on Manager-1 interface layer **Value After Reset:** {(DMAX_CH(x)_SMS < 2) ? DMAX_CH(x)_SMS : 0} **Exists:** Always

6.2.5 CHx_CFG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Configuration Register
- **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	51:49	48	47:y	x:44	43	42:y	x:39	38	37	36	35	34:32	31:29	28:25	24:22	21:18	17:4	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_29to31	WR_UID	RSVD_DMAC_CHx_CFG_22to24	RD_UID	RSVD_DMAC_CHx_CFG_4to17	DST_MULTBLK_TYPE	SRC_MULTBLK_TYPE

Table 6-27 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 **Volatile:** true

Bits	Name	Memory Access	Description
62:59	DST_OSR_LMT	R/W	Destination Outstanding Request Limit ■ Maximum outstanding request supported is 16. ■ Destination Outstanding Request Limit = DST_OSR_LMT + 1 **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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 **Volatile:** true
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 **Volatile:** true

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 manager bus interface and if the CHx_CFG.LOCK_CH bit is asserted, then no other channels are granted control of the manager bus interface for the duration specified in CHx_CFG.LOCK_CH_L. Indicates to the manager bus interface arbiter that this channel wants exclusive access to the manager 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 manager 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 Manager Bus Interface **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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 **Volatile:** true
48	RSVD_DMAC_CHx_CFG_48	R	DMAC Channelx Transfer Configuration Register (48bit) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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 **Volatile:** true **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 **Volatile:** true **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 **Volatile:** true
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 **Volatile:** true **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 39

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 **Volatile:** true **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 **Note**: This field is reserved for future use and currently this is always active high. **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 **Volatile:** true

Bits	Name	Memory Access	Description
37	SRC_HWHS_POL	R	Source Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Note**: This field is reserved for future use and currently this is always active high. **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 **Volatile:** true
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 **Volatile:** true

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 **Volatile:** true
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 the 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 **Volatile:** true

Bits	Name	Memory Access	Description
31:29	RSVD_DMAC_CHx_CFG_29to31	R	DMAC Channelx Transfer Configuration Register (bits 29to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
28:25	WR_UID	{(DMAX_CH(x)_WR_UID_EN == 1) ? "read-write" : "read-only"}	Defines the number of AXI Unique ID's supported for the AXI Write Channel. The value programmed must be less than or equal to DMAX_CH(x)_WR_UID. Otherwise, it is limited by the value DMAX_CH(x)_WR_UID. **Value After Reset:** DMAX_CH(x)_RST_WR_UID **Exists:** Always **Volatile:** true
24:22	RSVD_DMAC_CHx_CFG_22to24	R	DMAC Channelx Transfer Configuration Register (bits 22to24) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
21:18	RD_UID	{(DMAX_CH(x)_RD_UID_EN == 1) ? "read-write" : "read-only"}	Defines the number of AXI Unique ID's supported for the AXI Read Channel. The value programmed must be less than or equal to DMAX_CH(x)_RD_UID. Otherwise, it is limited by the value DMAX_CH(x)_RD_UID. **Value After Reset:** DMAX_CH(x)_RST_RD_UID **Exists:** Always **Volatile:** true
17:4	RSVD_DMAC_CHx_CFG_4to17	R	DMAC Channelx Transfer Configuration Register (bits 4to17) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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.

Bits	Name	Memory Access	Description
3:2...(cont.)	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"}	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 **Volatile:** true

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.

Bits	Name	Memory Access	Description
1:0...(cont.)	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"}	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 **Volatile:** true

6.2.6 CHx_CFG2 (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Configuration Register 2
- **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	31:29	28:25	24:22	21:18	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_29to31	WR_UID	RSVD_DMAC_CHx_CFG_22to24	RD_UID	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 6-28 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 **Volatile:** true

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 **Volatile:** true
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 **Volatile:** true
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 **Volatile:** true

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 manager bus interface and if the CHx_CFG.LOCK_CH bit is asserted, then no other channels are granted control of the manager bus interface for the duration specified in CHx_CFG.LOCK_CH_L. Indicates to the manager bus interface arbiter that this channel wants exclusive access to the manager 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 manager 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 Manager Bus Interface **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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 **Volatile:** true
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 **Volatile:** true

Bits	Name	Memory Access	Description
38	DST_HWHS_POL	R	Destination Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Note**: This field is reserved for future use and currently this is always active high. **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 **Volatile:** true
37	SRC_HWHS_POL	R	Source Hardware Handshaking Interface Polarity. ■ 0: ACTIVE HIGH ■ 1: ACTIVE LOW **Note**: This field is reserved for future use and currently this is always active high. **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 **Volatile:** true
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 **Volatile:** true

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 **Volatile:** true
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 the 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 **Volatile:** true

Bits	Name	Memory Access	Description
31:29	RSVD_DMAC_CHx_CFG_29to31	R	DMAC Channelx Transfer Configuration Register (bits 29to31) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
28:25	WR_UID	{(DMAX_CH(x)_WR_UID_EN == 1) ? "read-write" : "read-only"}	Defines the number of AXI Unique ID's supported for the AXI Write Channel. The value programmed must be less than or equal to DMAX_CH(x)_WR_UID. Otherwise, it is limited by the value DMAX_CH(x)_WR_UID. **Value After Reset:** DMAX_CH(x)_RST_WR_UID **Exists:** Always **Volatile:** true
24:22	RSVD_DMAC_CHx_CFG_22to24	R	DMAC Channelx Transfer Configuration Register (bits 22to24) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
21:18	RD_UID	{(DMAX_CH(x)_RD_UID_EN == 1) ? "read-write" : "read-only"}	Defines the number of AXI Unique ID's supported for the AXI Read Channel. The value programmed must be less than or equal to DMAX_CH(x)_RD_UID. Otherwise, it is limited by the value DMAX_CH(x)_RD_UID. **Value After Reset:** DMAX_CH(x)_RST_RD_UID **Exists:** Always **Volatile:** true
17	RSVD_DMAC_CHx_CFG_17	R	DMAC Channelx Transfer Configuration Register (bit 17) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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 **Volatile:** true **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 11

Bits	Name	Memory Access	Description
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 **Volatile:** true **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 **Volatile:** true
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 **Volatile:** true **Range Variable[y]:** LOG2_DMAX_NUM_HS_IF + 4

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 **Volatile:** true **Range Variable[x]:** LOG2_DMAX_NUM_HS_IF + 3 **Memory Access:** {(DMAX_NUM_HS_IF == 0) ? "read-only" : "read-write"}

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.

Bits	Name	Memory Access	Description
3:2...(cont.)	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"}	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 **Volatile:** true

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.

Bits	Name	Memory Access	Description
1:0...(cont.)	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"}	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 **Volatile:** true

6.2.7 CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Linked List Pointer Register
- **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

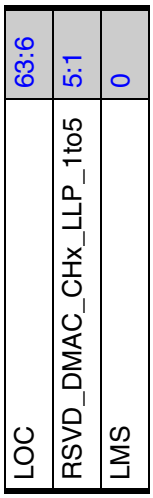


Table 6-29 Fields for Register: CHx_LLP (for x = 1; x <= DMAX_NUM_CHANNELS)

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

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 manager Select This bit identifies the AXI layer/interface where the memory device that stores the next linked list item resides. ■ 0: AXI Manager 1 ■ 1: AXI Manager 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 (MANAGER1_INTF): next Linked List item resides on AXI Manager1 interface ■ 0x1 (MANAGER2_INTF): next Linked List item resides on AXI Manager2 interface **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

6.2.8 CHx_STATUSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Status Register
- **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

63:47	46:32	31:22	21:0
RSVD_DMAC_CHx_STATUSREG_47to63	DATA_LEFT_IN_FIFO	RSVD_DMAC_CHx_STATUSREG_22to31	CMPLTD_BLK_TFR_SIZE

Table 6-30 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

Bits	Name	Memory Access	Description
46:32	DATA_LEFT_IN_FIFO	R	Data Left in FIFO. The data left in the FIFO status indicates the total number of data left in the channel FIFO after completing the current DMA block transfer or DMA block abnormal termination due to the following reasons: ■ AXI Transfer Error response (SLVERR or DECERR) reception ■ Channel Terminate The width of the data in channel FIFO is defined in terms of CHx_CTL.SRC_TR_WIDTH. For more information, see "DMA Status Register - CHx_STATUSREG" section of the databook. **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. The Completed Block Transfer Size status indicates the total number of data items transferred from the Source memory/peripheral to the Destination memory/peripheral at any point in the DMA block transfer. The completed block transfer size or data items is defined as follows: ■ When DMA or Source is a Flow controller, this field is defined in terms of the CHx_CTL.SRC_TR_WIDTH. ■ When Destination is a Flow Controller, this field is defined in terms of the CHx_CTL.DST_TR_WIDTH. For more information, see "DMA Status Register - CHx_STATUSREG" section of the databook. **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

6.2.9 CHx_SWHSSRCREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Software Handshake Source Register
- **Description:** Channelx Software handshake Source Register.
- **Size:** 64 bits
- **Offset:** 0x138 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:6	5	4	3	2	1	0
RSVD_DMAC_CHx_SWHSSRCREG_6to63	SWHS_LST_SRC_WE	SWHS_LST_SRC	SWHS_SGLREQ_SRC_WE	SWHS_SGLREQ_SRC	SWHS_REQ_SRC_WE	SWHS_REQ_SRC

Table 6-31 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
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

Bits	Name	Memory Access	Description
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. 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 current transfer is not the last transfer **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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

Bits	Name	Memory Access	Description
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. 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
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

Bits	Name	Memory Access	Description
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. 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

6.2.10 CHx_SWHSDSTREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Software Handshake Destination Register
- **Description:** Channelx Software handshake Destination Register.
- **Size:** 64 bits
- **Offset:** 0x140 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:6	5	4	3	2	1	0
RSVD_DMAC_CHx_SWHSDSTREG_6to63	SWHS_LST_DST_WE	SWHS_LST_DST	SWHS_SGLREQ_DST_WE	SWHS_SGLREQ_DST	SWHS_REQ_DST_WE	SWHS_REQ_DST

Table 6-32 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

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
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. 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

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. 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
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

Bits	Name	Memory Access	Description
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. 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

6.2.11 CHx_BLK_TFR_RESUMEREQREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Block Transfer Resume Request Register
- **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

RSVD_DMAC_CHx_BLK_TFR_RESUMEREQREG_1to63	63:1
BLK_TFR_RESUMEREQ	0

Table 6-33 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

6.2.12 CHx_AXI_IDREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x AXI ID Register
- **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 >= 1) && (DMAX_AXI_ID_SUFFIX_WIDTH != 0)

63:32	31:y	x:16	15:y	x:0
RSVD_DMAC_CHx_AXI_IDREG_32to63	RSVD_DMAC_CHx_AXI_IDREG_IDW_L2NCm32to63	AXI_WRITE_ID_SUFFIX	RSVD_DMAC_CHx_AXI_IDREG_IDW_L2NCm1to31	AXI_READ_ID_SUFFIX

Table 6-34 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

Bits	Name	Memory Access	Description
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 manager 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 manager 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
x:0	AXI_READ_ID_SUFFIX	R/W	AXI Read ID Suffix These bits form part of the ARID output of AXI3/AXI4 manager 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 manager 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

6.2.13 CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x AXI QoS Register
- **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

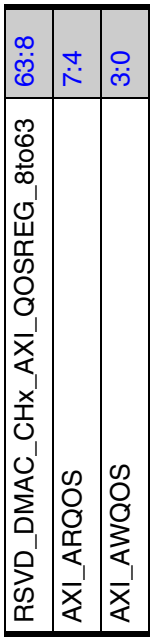


Table 6-35 Fields for Register: CHx_AXI_QOSREG (for x = 1; x <= DMAX_NUM_CHANNELS)

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 manager interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(DMAX_HAS_QOS == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
3:0	AXI_AWQOS	* Varies	AXI AWQOS. These bits form the awqos output of AXI4 manager interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(DMAX_HAS_QOS == 1) ? "read-write" : "read-only"}

6.2.14 CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Source Status Register
- **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 >= 1) && (DMAX_CH1_SRC_STAT_EN == 1)

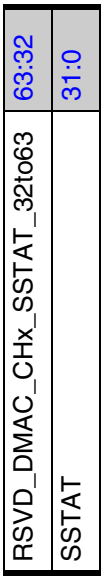


Table 6-36 Fields for Register: CHx_SSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

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

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

6.2.15 CHx_DSTAT (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Destination Status Register
- **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 >= 1) && (DMAX_CH1_DST_STAT_EN == 1)

Table 6-37 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

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

6.2.16 CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Source Status Fetch Register
- **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 >= 1) && (DMAX_CH1_SRC_STAT_EN == 1)

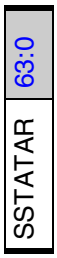


Table 6-38 Fields for Register: CHx_SSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

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

6.2.17 CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Destination Status Fetch Register
- **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 >= 1) && (DMAX_CH1_DST_STAT_EN == 1)

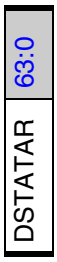


Table 6-39 Fields for Register: CHx_DSTATAR (for x = 1; x <= DMAX_NUM_CHANNELS)

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

6.2.18 CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Interrupt Status Enable Register
- **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

63:36	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_CHx_INTSTATUS_ENABLEREG_32to63	Enable_ECC_PROT_UIDMem_UnCorrERR_IntStat	Enable_ECC_PROT_UIDMem_CorrERR_IntStat	Enable_ECC_PROT_CHMem_UnCorrERR_IntStat	Enable_ECC_PROT_CHMem_CorrERR_IntStat	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_26	Enable_SLVIF_WRPARITY_ERR_IntStat	Enable_SLVIF_ADDRPARITY_ERR_IntStat	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to23	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 6-40 Fields for Register: CHx_INTSTATUS_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:36	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

Bits	Name	Memory Access	Description
35	Enable_ECC_PROT_UIDMem_UnCorrERR_IntStat	* Varies	Channel x Unique ID Memory Interface ECC Protection Uncorrectable Error Interrupt enable. ■ 0: Disable the generation of Channel x UID Memory Interface Uncorrectable Error Interrupt in CHx_INTSTATUSREG. ■ 1: Enable the generation Channel x UID Memory Interface Uncorrectable Error Interrupt in CHx_INTSTATUSREG. **Values:** ■ 0x1 (ENABLE_ECC_UIDMEM_UNCORRERR): Enable the propagation of UID Memory ECC Uncorrectable error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_ECC_UIDMEM_UNCORRERR): Disable the propagation of UID Memory ECC Uncorrectable error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_UIDMEMIF_EN == 1) ? "read-write" : "read-only"}
34	Enable_ECC_PROT_UIDMem_CorrERR_IntStat	* Varies	Channel x Unique ID Memory Interface ECC Protection Correctable Error Interrupt enable. ■ 0: Disable the generation of Channel x UID Memory Interface Correctable Error Interrupt in CHx_INTSTATUSREG. ■ 1: Enable the generation Channel x UID Memory Interface Correctable Error Interrupt in CHx_INTSTATUSREG. **Values:** ■ 0x1 (ENABLE_ECC_UIDMEM_CORRERR): Enable the propagation of UID Memory ECC Correctable error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_ECC_UIDMEM_CORRERR): Disable the propagation of UID Memory ECC Correctable error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_UIDMEMIF_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
33	Enable_ECC_PROT_CHMem_UnCorrERR_IntStat	* Varies	Channel x Channel Memory Interface ECC Protection Uncorrectable Error Interrupt enable. ■ 0: Disable the generation of Channel x Channel Memory Interface Uncorrectable Error Interrupt in CHx_INTSTATUSREG. ■ 1: Enable the generation Channel x Channel Memory Interface Uncorrectable Error Interrupt in CHx_INTSTATUSREG. **Values:** ■ 0x1 (ENABLE_ECC_CHMEM_UNCORRERR): Enable the propagation of Channel Memory ECC Uncorrectable error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_ECC_CHMEM_UNCORRERR): Disable the propagation of Channel Memory ECC Uncorrectable error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MEMIF_EN == 1) ? "read-write" : "read-only"}
32	Enable_ECC_PROT_CHMem_CorrERR_IntStat	* Varies	Channel x Channel Memory Interface ECC Protection Correctable Error Interrupt enable. ■ 0: Disable the generation of Channel x Channel Memory Interface Correctable Error Interrupt in CHx_INTSTATUSREG. ■ 1: Enable the generation of Channel x Channel Memory Interface Correctable Error Interrupt in CHx_INTSTATUSREG. **Values:** ■ 0x1 (ENABLE_ECC_CHMEM_CORRERR): Enable the propagation of Channel Memory ECC Correctable error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_ECC_CHMEM_CORRERR): Disable the propagation of Channel Memory ECC Correctable error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MEMIF_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
31	Enable_CH_ABORTED_IntStat	R/W	Channel Terminated Status Enable. ■ 0: Disable the generation of Channel Terminated Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Channel Terminated Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_CH_ABORTED): Enable the generation of Channel Terminated Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_CH_ABORTED): Disable the generation of Channel Terminated 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
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

Bits	Name	Memory Access	Description
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
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	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_26	R	DMAC Channelx Interrupt Status Register (bit 26) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
25	Enable_SLVIF_WRPARITY_ERR_IntStat	* Varies	Register Bus Interface Write Parity Error Enable. ■ 0: Disable the generation of Register Bus Interface Write Parity Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Write Parity Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (Enable_SLVIF_WRPARITY_ERR): Enable the generation of Register Bus Interface Write Parity Error in CH1_INTSTATUSREG ■ 0x0 (Disable_SLVIF_WRPARITY_ERR): Disable the generation of Register Bus Interface Write Parity Error in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
24	Enable_SLVIF_ADDRPARITY_ERR_IntStat	* Varies	Register Bus Interface Address Parity Error Enable. ■ 0: Disable the generation of Register Bus Interface Address Parity Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Address Parity Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (Enable_SLVIF_ADDRPARITY_ERR): Enable the generation of Register Bus Interface Address Parity Error in CH1_INTSTATUSREG ■ 0x0 (Disable_SLVIF_ADDRPARITY_ERR): Disable the generation of Register Bus Interface Address Parity Error in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ADDR_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
23:22	RSVD_DMAC_CHx_INTSTATUS_ENABLEREG_22to23	R	DMAC Channelx Interrupt Status Register (bits 22to23) Reserved bits - Read Only **Value After Reset:** 0x3 **Exists:** Always

Bits	Name	Memory Access	Description
21	Enable_SLVIF_WRONHOLD_ERR_IntStat	R/W	Register Bus Interface Write On Hold Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Write On Hold Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Write On Hold Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WRONHOLD_ERR): Enable the generation of Register Bus Interface Write On Hold Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WRONHOLD_ERR): Disable the generation of Register Bus Interface Write On Hold Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
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_SHADOW-REG_WRON_VALID_ERR): Enable the generation of Shadow register Write On Valid Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_SHADOW-REG_WRON_VALID_ERR): Disable the generation of Shadow Register Write On Valid Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
19	Enable_SLVIF_WRONCHEN_ERR_IntStat	R/W	Register Bus Interface Write On Channel Enabled Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Write On Channel enabled Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Write On Channel enabled Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WRONCHEN_ERR): Enable the generation of Register Bus Interface Write On Channel enabled Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WRONCHEN_ERR): Disable the generation of Register Bus Interface Write On Channel enabled Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
18	Enable_SLVIF_RD2RWO_ERR_IntStat	R/W	Register Bus Interface Read to write Only Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Read to Write only Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Read to Write Only Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_RD2RWO_ERR): Enable the generation of Register Bus Interface Read to Write Only Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_RD2RWO_ERR): Disable the generation of Register Bus Interface Read to Write only Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
17	Enable_SLVIF_WR2RO_ERR_IntStat	R/W	Register Bus Interface Write to Read Only Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Write to Read only Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Write to Read Only Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_WR2RO_ERR): Enable the generation of Register Bus Interface Write to Read Only Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_WR2RO_ERR): Disable the generation of Register Bus Interface Write to Read only Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
16	Enable_SLVIF_DEC_ERR_IntStat	R/W	Register Bus Interface Decode Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Decode Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Decode Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_DEC_ERR): Enable the generation of Register Bus Interface Decode Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_DEC_ERR): Disable the generation of Register Bus 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

Bits	Name	Memory Access	Description
14	Enable_SLVIF_MULTIBLKTYPE_ERR_IntStat	R/W	Register Bus Interface Multi Block type Error Status Enable. ■ 0: Disable the generation of Register Bus Interface Multi Block type Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Register Bus Interface Multi Block type Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SLVIF_MULTIBLKTYPE_ERR): Enable the generation of Register Bus Interface Multi Block type Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SLVIF_MULTIBLKTYPE_ERR): Disable the generation of Register Bus Interface Multi Block type Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
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 Subordinate Error Status Enable. ■ 0: Disable the generation of LLI WRITE Subordinate Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI WRITE Subordinate Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_WR_SLV_ERR): Enable the generation of LLI WRITE Subordinate Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_WR_SLV_ERR): Disable the generation of LLI WRITE Subordinate Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
11	Enable_LLI_RD_SLV_ERR_IntStat	R/W	LLI Read Subordinate Error Status Enable. ■ 0: Disable the generation of LLI Read Subordinate Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of LLI Read Subordinate Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_LLI_RD_SLV_ERR): Enable the generation of LLI Read Subordinate Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_LLI_RD_SLV_ERR): Disable the generation of LLI Read Subordinate 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
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

Bits	Name	Memory Access	Description
8	Enable_DST_SLV_ERR_IntStat	R/W	Destination Subordinate Error Status Enable. ■ 0: Disable the generation of Destination Subordinate Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Destination Subordinate Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_DST_SLV_ERR): Enable the generation of Destination Subordinate Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_DST_SLV_ERR): Disable the generation of Destination Subordinate Error Interrupt in CH1_INTSTATUSREG **Value After Reset:** 0x1 **Exists:** Always
7	Enable_SRC_SLV_ERR_IntStat	R/W	Source Subordinate Error Status Enable. ■ 0: Disable the generation of Source Subordinate Error Interrupt in CHx_INTSTATUSREG ■ 1: Enable the generation of Source Subordinate Error Interrupt in CHx_INTSTATUSREG **Values:** ■ 0x1 (ENABLE_SRC_SLV_ERR): Enable the generation of Source Subordinate Error Interrupt in CH1_INTSTATUSREG ■ 0x0 (DISABLE_SRC_SLV_ERR): Disable the generation of Source Subordinate 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

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
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

Bits	Name	Memory Access	Description
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
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

6.2.19 CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Interrupt Status Register
- **Description:** Channelx Interrupt Status Register captures the Channelx specific interrupts
- **Size:** 64 bits
- **Offset:** 0x188 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= x

63:36	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_CHx_INTSTATUSREG_36to63	ECC_PROT_UIDMem_UnCorrERR_IntStat	ECC_PROT_UIDMem_CorrERR_IntStat	ECC_PROT_CHMem_UnCorrERR_IntStat	ECC_PROT_CHMem_CorrERR_IntStat	CH_ABORTED_IntStat	CH_DISABLED_IntStat	CH_SUSPENDED_IntStat	CH_SRC_SUSPENDED_IntStat	CH_LOCK_CLEARED_IntStat	RSVD_DMAC_CHx_INTSTATUSREG_26	SLVIF_WRPARITY_ERR_IntStat	SLVIF_ADDRPARITY_ERR_IntStat	RSVD_DMAC_CHx_INTSTATUSREG_22to23	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 6-41 Fields for Register: CHx_INTSTATUS (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:36	RSVD_DMAC_CHx_INTSTATUSREG_36to63	R	DMAC Channelx Specific Interrupt Register (bits 36to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
35	ECC_PROT_UIDMem_UnCorrERR_IntStat	R	Channel x UID Memory Interface ECC Protection related Uncorrectable Error Interrupt Status bit. This error occurs if ECC Uncorrectable error is detected on the UID Memory Interface data. ■ 0: No Channel x UID Memory Interface Uncorrectable Error. ■ 1: Channel x UID Memory Interface Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in CHx_INTSTATUS_ENABLEReg register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in CHx_IntClearReg. **Values:** ■ 0x1 (ACTIVE_EccPROT_UIDMem_UnCorrERR): Channel x UID Memory Interface Uncorrectable Error detected ■ 0x0 (INACTIVE_EccPROT_UIDMem_UnCorrERR): Channel x UID Memory Interface Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
34	ECC_PROT_UIDMem_CorrERR_IntStat	R	Channel x UID Memory Interface ECC Protection related Correctable Error Interrupt Status bit. This error occurs if ECC correctable error is detected on the UID Memory Interface data. ■ 0: No Channel x UID Memory Interface correctable Error. ■ 1: Channel x UID Memory Interface correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in CHx_INTSTATUS_ENABLEReg register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in CHx_IntClearReg. **Values:** ■ 0x1 (ACTIVE_EccPROT_UIDMem_CorrERR): Channel x UID Memory Interface correctable Error detected ■ 0x0 (INACTIVE_EccPROT_UIDMem_CorrERR): Channel x UID Memory Interface correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
33	ECC_PROT_CHMem_UnCorrERR_IntStat	R	Channel x FIFO Memory Interface ECC Protection related Uncorrectable Error Interrupt Status bit. This error occurs if ECC Uncorrectable error is detected on the FIFO Memory Interface data. ■ 0: No Channel x FIFO Memory Interface Uncorrectable Error. ■ 1: Channel x FIFO Memory Interface Uncorrectable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in CHx_INTSTATUS_ENABLEReg register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in CHx_IntClearReg. **Values:** ■ 0x1 (ACTIVE_EccPROT_CHMem_UnCorrERR): Channel x FIFO Memory Interface Uncorrectable Error detected ■ 0x0 (INACTIVE_EccPROT_CHMem_UnCorrERR): Channel x FIFO Memory Interface Uncorrectable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
32	ECC_PROT_CHMem_CorrERR_IntStat	R	Channel x FIFO Memory Interface ECC Protection related Correctable Error Interrupt Status bit. This error occurs if ECC correctable error is detected on the FIFO Memory Interface data. ■ 0: No Channel x FIFO Memory Interface correctable Error. ■ 1: Channel x FIFO Memory Interface correctable Error detected. Error Interrupt status is generated if the corresponding Status Enable bit in CHx_INTSTATUS_ENABLEReg register is set to 1. This bit is cleared to 0 on writing 1 to the corresponding interrupt clear bit in CHx_IntClearReg. **Values:** ■ 0x1 (ACTIVE_EccPROT_CHMem_CorrERR): Channel x FIFO Memory Interface correctable Error detected ■ 0x0 (INACTIVE_EccPROT_CHMem_CorrERR): Channel x FIFO Memory Interface correctable Error not detected **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
31	CH_ABORTED_IntStat	R	Channel Terminated. This indicates to the software that the corresponding channel in DW_axi_dmac is terminated. ■ 0: Channel is not terminated ■ 1: Channel is terminated 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 terminated ■ 0x0 (INACTIVE_CH_ABORTED): Channel is not terminated **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

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

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 manager 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	RSVD_DMAC_CHx_INTSTATUSREG_26	R	DMAC Channelx Specific Interrupt Register (bit 26) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
25	SLVIF_WRPARITY_ERR_IntStat	R	Register Bus Interface Write Parity Error. This error occurs if a Write operation is performed on a channel register; But the Write data fails the even/odd parity check. ■ 0: No Register Bus Interface Write Parity Errors. ■ 1: Register Bus Interface Write Parity 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_WRPARITY_ERR): Register Bus Interface Write Parity Error detected ■ 0x0 (INACTIVE_SLVIF_WRPARITY_ERR): No Register Bus Interface Write Parity Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
24	SLVIF_ADDRPARITY_ERR_IntStat	R	Register Bus Interface Address Parity Error. This error occurs if address is used to do write/read operation on a channel register; But the Address fails the even/odd parity check. ■ 0: No Register Bus Interface Address Parity Errors. ■ 1: Register Bus Interface Address Parity 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_ADDRPARITY_ERR): Register Bus Interface Address Parity Error detected ■ 0x0 (INACTIVE_SLVIF_ADDRPARITY_ERR): No Register Bus Interface Address Parity Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
23:22	RSVD_DMAC_CHx_INTSTATUSREG_22to23	R	DMAC Channelx Specific Interrupt Register (bits 22to23) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
21	SLVIF_WRONHOLD_ERR_IntStat	R	Register Bus 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 Register Bus Interface Write On Hold Errors. ■ 1: Register Bus 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): Register Bus Interface Write On Hold Error detected ■ 0x0 (INACTIVE_SLVIF_WRONHOLD_ERR): No Register Bus Interface Write On Hold Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
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 Register Bus Interface Shadow Register Write On Valid Errors. ■ 1: Register Bus 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_SHADOW-REG_WRON_VALID_ERR): Register Bus Interface Shadow Register Write On Valid Error detected ■ 0x0 (INACTIVE_SLVIF_SHADOW-REG_WRON_VALID_ERR): No Register Bus Interface Shadow Register Write On Valid Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
19	SLVIF_WRONCHEN_ERR_IntStat	R	Register Bus 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 Register Bus Interface Write On Channel Enabled Errors. ■ 1: Register Bus 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): Register Bus Interface Write On Channel Enabled Error detected ■ 0x0 (INACTIVE_SLVIF_WRONCHEN_ERR): No Register Bus Interface Write On Channel Enabled Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
18	SLVIF_RD2RWO_ERR_IntStat	R	Register Bus Interface Read to write Only Error. This error occurs if read operation is performed to a Write Only register. ■ 0: No Register Bus Interface Read to Write Only Errors. ■ 1: Register Bus 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): Register Bus Interface Read to Write Only Error detected ■ 0x0 (INACTIVE_SLVIF_RD2RWO_ERR): No Register Bus Interface Read to Write Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
17	SLVIF_WR2RO_ERR_IntStat	R	Register Bus Interface Write to Read Only Error. This error occurs if write operation is performed to a Read Only register. ■ 0: No Register Bus Interface Write to Read Only Errors. ■ 1: Register Bus 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): Register Bus Interface Write to Read Only Error detected ■ 0x0 (INACTIVE_SLVIF_WR2RO_ERR): No Register Bus Interface Write to Read Only Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
16	SLVIF_DEC_ERR_IntStat	R	Register Bus 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 register bus interface. ■ 0: No Register Bus Interface Decode errors. ■ 1: Register Bus 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): Register Bus Interface Decode Error detected ■ 0x0 (INACTIVE_SLVIF_DEC_ERR): No Register Bus Interface Decode errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
15	RSVD_DMAC_CHx_INTSTATUSREG_15	R	DMAC Channelx Specific Interrupt Register (bit 15) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
14	SLVIF_MULTIBLKTYPE_ERR_IntStat	R	Register Bus 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

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

Bits	Name	Memory Access	Description
12	LLI_WR_SLV_ERR_IntStat	R	LLI WRITE Subordinate Error. Subordinate Error detected by Manager Interface during LLI write-back operation. This error occurs if the register bus interface on which LLI resides issues a Subordinate 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 Subordinate Errors. ■ 1: LLI Write SUBORDINATE 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 SUBORDINATE Error detected ■ 0x0 (INACTIVE_LLI_WR_SLV): No LLI write Subordinate Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
11	LLI_RD_SLV_ERR_IntStat	R	LLI Read Subordinate Error. Subordinate Error detected by Manager Interface during LLI read operation. This error occurs if the register bus interface on which LLI resides issues a Subordinate 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 Subordinate Errors. ■ 1: LLI read Subordinate 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 Subordinate Error detected ■ 0x0 (INACTIVE_LLI_RD_SLV_ERR): No LLI Read Subordinate Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
10	LLI_WR_DEC_ERR_IntStat	R	LLI WRITE Decode Error. Decode Error detected by Manager Interface during LLI write-back operation. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/subordinate. 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
9	LLI_RD_DEC_ERR_IntStat	R	LLI Read Decode Error. Decode Error detected by Manager Interface during LLI read operation. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/subordinate. 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 (ACTIVE_LLI_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

Bits	Name	Memory Access	Description
8	DST_SLV_ERR_IntStat	R	Destination Subordinate Error. Subordinate Error detected by Manager Interface during destination data transfer. This error occurs if the register bus interface to which the data is written issues a Subordinate 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 Subordinate Errors ■ 1: Destination Subordinate 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 Subordinate Errors Detected ■ 0x0 (INACTIVE_DST_SLV_ERR): No Destination Subordinate Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
7	SRC_SLV_ERR_IntStat	R	Source Subordinate Error. Subordinate Error detected by Manager Interface during source data transfer. This error occurs if the register bus interface from which the data is read issues a Subordinate 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 Subordinate Errors ■ 1: Source Subordinate 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 Subordinate Error Detected ■ 0x0 (INACTIVE_SRC_SLV_ERR): No Source Subordinate Errors **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

Bits	Name	Memory Access	Description
6	DST_DEC_ERR_IntStat	R	Destination Decode Error. Decode Error detected by Manager Interface during destination data transfer. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/subordinate. 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 Manager Interface during source data transfer. This error occurs if the access is to invalid address and a Decode Error is returned from interconnect/subordinate. 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

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 transaction is not complete **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true
2	RSVD_DMAC_CHx_INTSTATUSREG_2	R	DMAC Channelx Specific Interrupt Register (bit 2) Reserved bit - Read Only **Value After Reset:** 0x0 **Exists:** Always **Volatile:** true

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

6.2.20 CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Interrupt Signal Enable Register
- **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

63:36	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_CHx_INTSIGNAL_ENABLEREG_36to63	Enable_ECC_PROT_UIDMem_UnCorrERR_IntSignal	Enable_ECC_PROT_UIDMem_CorrERR_IntSignal	Enable_ECC_PROT_CHMem_UnCorrERR_IntSignal	Enable_ECC_PROT_CHMem_CorrERR_IntSignal	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_INTSIGNAL_ENABLEREG_26	Enable_SLVIF_WRPARITY_ERR_IntSignal	Enable_SLVIF_ADDRPARITY_ERR_IntSignal	RSVD_DMAC_CHx_INTSIGNAL_ENABLEREG_22to23	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 6-42 Fields for Register: CHx_INTSIGNAL_ENABLEREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:36	RSVD_DMAC_CHx_INTSIGNAL_ENABLEREG_36to63	R	DMAC Channelx Interrupt Signal Enable Register (bits 36to63) Reserved bits - Read Only **Value After Reset:** 0xffffffff **Exists:** Always

Bits	Name	Memory Access	Description
35	Enable_ECC_PROT_UIDMem_UnCorrERR_IntSignal	* Varies	Channel x Unique ID Memory Interface ECC Protection Uncorrectable Error Interrupt Signal enable. ■ 0: Disable the propagation of Channel x UID Memory Interface Uncorrectable Error to generate a port level interrupt. ■ 1: Enable the propagation of Channel x UID Memory Interface Uncorrectable Error to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_ECC_UIDMEM_UNCORRERR): Enable the propagation of UID Memory ECC Uncorrectable error Interrupt to port level interrupt ■ 0x0 (DISABLE_ECC_UIDMEM_UNCORRERR): Disable the propagation of UID Memory ECC Uncorrectable error Interrupt to port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_UIDMEMIF_EN == 1) ? "read-write" : "read-only"}
34	Enable_ECC_PROT_UIDMem_CorrERR_IntSignal	* Varies	Channel x Unique ID Memory Interface ECC Protection Correctable Error Interrupt Signal enable. ■ 0: Disable the propagation of Channel x UID Memory Interface Correctable Error to generate a port level interrupt. ■ 1: Enable the propagation of Channel x UID Memory Interface Correctable Error to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_ECC_UIDMEM_CORRERR): Enable the propagation of UID Memory ECC Correctable error Interrupt to port level interrupt ■ 0x0 (DISABLE_ECC_UIDMEM_CORRERR): Disable the propagation of UID Memory ECC Correctable error Interrupt to port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_UIDMEMIF_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
33	Enable_ECC_PROT_CHMem_UnCorrERR_IntSignal	* Varies	Channel x Channel Memory Interface ECC Protection Uncorrectable Error Interrupt Signal enable. ■ 0: Disable the propagation of Channel x Channel Memory Interface Uncorrectable Error to generate a port level interrupt. ■ 1: Enable the propagation of Channel x Channel Memory Interface Uncorrectable Error to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_ECC_CHMEM_UNCORRERR): Enable the propagation of Channel Memory ECC Uncorrectable error Interrupt to port level interrupt ■ 0x0 (DISABLE_ECC_CHMEM_UNCORRERR): Disable the propagation of Channel Memory ECC Uncorrectable error Interrupt to port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MEMIF_EN == 1) ? "read-write" : "read-only"}
32	Enable_ECC_PROT_CHMem_CorrERR_IntSignal	* Varies	Channel x Channel Memory Interface ECC Protection Correctable Error Interrupt Signal enable. ■ 0: Disable the propagation of Channel x Channel Memory Interface Correctable Error to generate a port level interrupt. ■ 1: Enable the propagation of Channel x Channel Memory Interface Correctable Error to generate a port level interrupt. **Values:** ■ 0x1 (ENABLE_ECC_CHMEM_CORRERR): Enable the propagation of Channel Memory ECC Correctable error Interrupt to port level interrupt ■ 0x0 (DISABLE_ECC_CHMEM_CORRERR): Disable the propagation of Channel Memory ECC Correctable error Interrupt to port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ECC_MEMIF_EN == 1) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
31	Enable_CH_ABORTED_IntSignal	R/W	Channel Terminated Signal Enable. ■ 0: Disable the propagation of Channel Terminated Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Channel Terminated Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_CH_ABORTED_IntSignal): Enable the propagation of Channel Terminated Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_CH_ABORTED_IntSignal): Disable the propagation of Channel Terminated 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
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

Bits	Name	Memory Access	Description
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
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	RSVD_DMAC_CHx_INTSIGNAL_ENABLEREG_26	R	DMAC Channelx Interrupt Signal Enable Register (bit 26) Reserved bit - Read Only **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
25	Enable_SLVIF_WRPARITY_ERR_IntSignal	* Varies	Register Bus Interface Write Parity Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Write Parity Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Write Parity Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (Enable_SLVIF_WRPARITY_ERR_IntSignal): Enable the propagation of Register Bus Interface Write Parity Error Interrupt to generate a port level interrupt ■ 0x0 (Disable_SLVIF_WRPARITY_ERR_IntSignal): Disable the propagation of Register Bus Interface Write Parity Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
24	Enable_SLVIF_ADDRPARITY_ERR_IntSignal	* Varies	Register Bus Interface Address Parity Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Address Parity Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Address Parity Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (Enable_SLVIF_ADDRPARITY_ERR_IntSignal): Enable the propagation of Register Bus Interface Address Parity Error Interrupt to generate a port level interrupt ■ 0x0 (Disable_SLVIF_ADDRPARITY_ERR_IntSignal): Disable the propagation of Register Bus Interface Address Parity Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always **Memory Access:** {(DMAX_ADDR_PARITY_PROT_EN == 1) ? "read-write" : "read-only"}
23:22	RSVD_DMAC_CHx_INTSIGNAL_ENABLEREG_22to23	R	DMAC Channelx Interrupt Signal Enable Register (bits 22to23) Reserved bits - Read Only **Value After Reset:** 0x3 **Exists:** Always

Bits	Name	Memory Access	Description
21	Enable_SLVIF_WRONHOLD_ERR_IntSignal	R/W	Register Bus Interface Write On Hold Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Write On Hold Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Write On Hold Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WRONHOLD_ERR_IntSignal): Enable the propagation of Register Bus Interface Write On Hold Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WRONHOLD_ERR_IntSignal): Disable the propagation of Register Bus Interface Write On Hold Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
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_SHADOW-REG_WRON_VALID_ERR_IntSignal): Enable the propagation of Shadow register Write On Valid Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_SHADOW-REG_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

Bits	Name	Memory Access	Description
19	Enable_SLVIF_WRONCHEN_ERR_IntSignal	R/W	Register Bus Interface Write On Channel Enabled Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Write On Channel enabled Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Write On Channel enabled Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WRONCHEN_ERR_IntSignal): Enable the propagation of Register Bus Interface Write On Channel enabled Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WRONCHEN_ERR_IntSignal): Disable the propagation of Register Bus Interface Write On Channel enabled Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
18	Enable_SLVIF_RD2RWO_ERR_IntSignal	R/W	Register Bus Interface Read to write Only Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Read to Write only Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Read to Write Only Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_RD2RWO_ERR_IntSignal): Enable the propagation of Register Bus Interface Read to Write Only Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_RD2RWO_ERR_IntSignal): Disable the propagation of Register Bus Interface Read to Write only Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
17	Enable_SLVIF_WR2RO_ERR_IntSignal	R/W	Register Bus Interface Write to Read Only Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Write to Read only Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Write to Read Only Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_WR2RO_ERR_IntSignal): Enable the propagation of Register Bus Interface Write to Read Only Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_WR2RO_ERR_IntSignal): Disable the propagation of Register Bus Interface Write to Read only Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
16	Enable_SLVIF_DEC_ERR_IntSignal	R/W	Register Bus Interface Decode Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Decode Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Decode Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_DEC_ERR_IntSignal): Enable the propagation of Register Bus Interface Decode Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_DEC_ERR_IntSignal): Disable the propagation of Register Bus 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

Bits	Name	Memory Access	Description
14	Enable_SLVIF_MULTIBLKTYPE_ERR_IntSignal	R/W	Register Bus Interface Multi Block type Error Signal Enable. ■ 0: Disable the propagation of Register Bus Interface Multi Block type Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Register Bus Interface Multi Block type Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SLVIF_MULTIBLKTYPE_ERR_IntSignal): Enable the propagation of Register Bus Interface Multi Block type Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SLVIF_MULTIBLKTYPE_ERR_IntSignal): Disable the propagation of Register Bus Interface Multi Block type Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
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 Subordinate Error Signal Enable. ■ 0: Disable the propagation of LLI WRITE Subordinate Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI WRITE Subordinate Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_WR_SLV_ERR_IntSignal): Enable the propagation of LLI WRITE Subordinate Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_WR_SLV_ERR_IntSignal): Disable the propagation of LLI WRITE Subordinate Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always

Bits	Name	Memory Access	Description
11	Enable_LLI_RD_SLV_ERR_IntSignal	R/W	LLI Read Subordinate Error Signal Enable. ■ 0: Disable the propagation of LLI Read Subordinate Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of LLI Read Subordinate Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_LLI_RD_SLV_ERR_IntSignal): Enable the propagation of LLI Read Subordinate Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_LLI_RD_SLV_ERR_IntSignal): Disable the propagation of LLI Read Subordinate 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
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

Bits	Name	Memory Access	Description
8	Enable_DST_SLV_ERR_IntSignal	R/W	Destination Subordinate Error Signal Enable. ■ 0: Disable the propagation of Destination Subordinate Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Destination Subordinate Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_DST_SLV_ERR_IntSignal): Enable the propagation of Destination Subordinate Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_DST_SLV_ERR_IntSignal): Disable the propagation of Destination Subordinate Error Interrupt to generate a port level interrupt **Value After Reset:** 0x1 **Exists:** Always
7	Enable_SRC_SLV_ERR_IntSignal	R/W	Source Subordinate Error Signal Enable. ■ 0: Disable the propagation of Source Subordinate Error Interrupt to generate a port level interrupt ■ 1: Enable the propagation of Source Subordinate Error Interrupt to generate a port level interrupt **Values:** ■ 0x1 (ENABLE_SRC_SLV_ERR_IntSignal): Enable the propagation of Source Subordinate Error Interrupt to generate a port level interrupt ■ 0x0 (DISABLE_SRC_SLV_ERR_IntSignal): Disable the propagation of Source Subordinate 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

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
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

Bits	Name	Memory Access	Description
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
0	Enable_BLOCK_TFR_DONE_IntSignal	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

6.2.21 CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Interrupt Status Clear Register
- **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

63:36	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_CHx_INTCLEARREG_36to63	Clear_ECC_PROT_UIDMem_UnCorrERR_IntStat	Clear_ECC_PROT_UIDMem_CorrERR_IntStat	Clear_ECC_PROT_CHMem_UnCorrERR_IntStat	Clear_ECC_PROT_CHMem_CorrERR_IntStat	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_26	Clear_SLVIF_WRPARITY_ERR_IntStat	Clear_SLVIF_ADDRPARITY_ERR_IntStat	RSVD_DMAC_CHx_INTCLEARREG_22to23	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 6-43 Fields for Register: CHx_INTCLEARREG (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:36	RSVD_DMAC_CHx_INTCLEARREG_36to63	W	DMAC Channelx Interrupt Clear Register (bits 36to63) Reserved bit - Write Only **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
35	Clear_ECC_PROT_UIDMem_UnCorrERR_IntStat	W	ECC Protection Uncorrectable UID Memory Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (Clear_ECC_UIDMEM_CORRERR_IntStat): Clear the ECC_PROT_UIDMem_UnCorrERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
34	Clear_ECC_PROT_UIDMem_CorrERR_IntStat	W	ECC Protection Correctable UID Memory Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_ECC_UIDMEM_CORRERR_IntStat): Clear the ECC_PROT_UIDMem_CorrERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
33	Clear_ECC_PROT_CHMem_UnCorrERR_IntStat	W	ECC Protection Uncorrectable Channel Memory Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_ECC_CHMEM_CORRERR_IntStat): Clear the ECC_PROT_CHMem_UnCorrERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
32	Clear_ECC_PROT_CHMem_CorrERR_IntStat	W	ECC Protection Correctable Channel Memory Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_ECC_CHMEM_CORRERR_IntStat): Clear the ECC_PROT_CHMem_CorrERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
31	Clear_CH_ABORTED_IntStat	W	Channel Terminated 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

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	RSVD_DMAC_CHx_INTCLEARREG_26	W	DMAC Channelx Interrupt Clear Register (bit 26) Reserved bit - Write Only **Value After Reset:** 0x0 **Exists:** Always
25	Clear_SLVIF_WRPARITY_ERR_IntStat	W	Register Bus Interface Write Parity Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_WRPARITY_ERR): Clear the SLVIF_WRPARITY_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
24	Clear_SLVIF_ADDRPARITY_ERR_IntStat	W	Register Bus Interface Address Parity Error Interrupt Clear Bit. This bit is used to clear the corresponding channel interrupt status bit in CHx_INTSTATUSREG. **Values:** ■ 0x1 (CLEAR_SLVIF_ADDRPARITY_ERR): Clear the SLVIF_ADDRPARITY_ERR interrupt in the Interrupt Status Register(CH1_IntStatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
23:22	RSVD_DMAC_CHx_INTCLEARREG_22to23	W	DMAC Channelx Interrupt Clear Register (bits 22to23) Reserved bit - Write Only **Value After Reset:** 0x0 **Exists:** Always

Bits	Name	Memory Access	Description
21	Clear_SLVIF_WRONHOLD_ERR_IntStat	W	Register Bus 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
19	Clear_SLVIF_WRONCHEN_ERR_IntStat	W	Register Bus 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	Register Bus 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

Bits	Name	Memory Access	Description
17	Clear_SLVIF_WR2RO_ERR_IntStat	W	Register Bus 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
16	Clear_SLVIF_DEC_ERR_IntStat	W	Register Bus 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 - Write Only **Value After Reset:** 0x0 **Exists:** Always
14	Clear_SLVIF_MULTIBLKTYPE_ERR_IntStat	W	Register Bus 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

Bits	Name	Memory Access	Description
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
12	Clear_LLI_WR_SLV_ERR_IntStat	W	LLI WRITE Subordinate 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 Subordinate 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

Bits	Name	Memory Access	Description
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
8	Clear_DST_SLV_ERR_IntStat	W	Destination Subordinate 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_IntSta-tusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always
7	Clear_SRC_SLV_ERR_IntStat	W	Source Subordinate 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_IntSta-tusReg). ■ 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

Bits	Name	Memory Access	Description
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
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 - Write 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_Int-StatusReg). ■ 0x0 (NO_ACTION): Inactive signal. No action taken. **Value After Reset:** 0x0 **Exists:** Always

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

6.2.22 CHx_CFG_EXTD (for x = 1; x <= DMAX_NUM_CHANNELS)

- **Name:** DW_axi_dmac Channel x Extended Configuration Register
- **Description:** Channel x Extended Configuration 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. This feature adds register fields - CHx_CFG_EXTD.LLI_AXPROT and CHx_CFG_EXTD.LLI_AXCACHE. For the next LLI Fetch and/or LLI Write Back, this programmed AXI Protection/Cache value is used as AxPROT and AxCACHE.
- **Size:** 64 bits
- **Offset:** 0x1A0 + (x-1)*0x100
- **Exists:** DMAX_NUM_CHANNELS >= 1

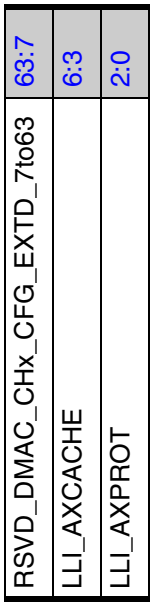


Table 6-44 Fields for Register: CHx_CFG_EXTD (for x = 1; x <= DMAX_NUM_CHANNELS)

Bits	Name	Memory Access	Description
63:7	RSVD_DMAC_CHx_CFG_EXTD_7to63	R	DMAC Channel x Extended Configuration Register (bits 7to63) Reserved bits - Read Only **Value After Reset:** 0x0 **Exists:** Always
6:3	LLI_AXCACHE	* Varies	LLI Fetch and LLI Writeback AXI AxCACHE. These bits form the arcache/awcache output of AXI manager interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(SNPS_RSVDPARAM_120 == 0) ? "read-write" : "read-only"}

Bits	Name	Memory Access	Description
2:0	LLI_AXPROT	* Varies	LLI Fetch and LLI Writeback AXI AxPROT. These bits form the arprot/awprot output of AXI manager interface. **Value After Reset:** 0x0 **Exists:** Always **Memory Access:** {(SNPS_RSVDPARAM_120 == 0) ? "read-write" : "read-only"}

A

Area, Speed, Power, and Performance

This section discusses the area, power, performance, and the hardware configuration parameters that affect the performance of the DW_axi_dmac.

It covers the following topic:

- "Area, Frequency, and Power Consumption" on page 600

A.1 Area, Frequency, and Power Consumption

Table A-1 provides information about the synthesis results (power consumption, frequency, and area) and DFT coverage of the DW_axi_dmac using the industry standard 7nm technology library.

Table A-1 Synthesis Results for DW_axi_dmac

Configuration	Operating Frequency	Gate Count (gates)	Power Consumption		TetraMax (%)		SpyGlass StuckAtCov (%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	dmac_core_clock = 400 MHz	25885	70 nW	0.503 mW	99.94	95.19	99.7
Typical Configuration 1[a]	dmac_core_clock = 400 MHz hclk = 200 MHz	426669	1000 nW	9.649 mW	99.9	95.07	99.8
Typical Configuration - 2[b]	dmac_core_clock = 400 MHz	705272	2000 nW	16.898 mW	99.9	95.05	99.8

a. For the list of configuration parameters and their values for the Typical Configuration 1, see Table A-2.
b. For the list of configuration parameters and their values for the Typical Configuration 2, see Table A-3 on page 605.

Table A-2 Typical Configuration 1

Configuration Parameter	Value
DMAX_ID_NUM	0x756a1
DMAX_NUM_CHANNELS	8
DMAX_NUM_HS_IF	16
DMAX_INTR_IO_TYPE	2
DMAX_SLVIF_STATUS_OP_EN	1
DMAX_CORE_STATUS_OP_EN	1
DMAX_HOLD_IO_EN	1
DMAX_UNALIGED_XFER_EN	1
DMAX_DEBUG_PORTS_EN	1
DMAX_M_ADDER_WIDTH	36
DMAX_ENABLE_LAST_WRITE	1
DMAX_MSTIF1_OSR_LMT	128
DMAX_SLVIF_CLOCK_MODE	1
DMAX_HS_SAME_ASYNC_CLK	1
DMAX_HS1_ASYNC_CLK	1

Configuration Parameter	Value
DMAX_HS2_ASYNC_CLK	1
DMAX_HS3_ASYNC_CLK	1
DMAX_HS4_ASYNC_CLK	1
DMAX_HS5_ASYNC_CLK	0
DMAX_HS6_ASYNC_CLK	0
DMAX_HS7_ASYNC_CLK	0
DMAX_HS8_ASYNC_CLK	0
DMAX_HS9_ASYNC_CLK	1
DMAX_HS10_ASYNC_CLK	1
DMAX_HS11_ASYNC_CLK	1
DMAX_HS12_ASYNC_CLK	1
DMAX_HS13_ASYNC_CLK	1
DMAX_HS14_ASYNC_CLK	6
DMAX_HS15_ASYNC_CLK	0
DMAX_HS16_ASYNC_CLK	1
DMAX_BCM_CKG_CELL_LT	1
DMAX_CSLP_EN	1
DMAX_CHNL_CSLP_EN	1
DMAX_SBIU_CSLP_EN	1
DMAX_MXIF_CSLP_EN	1
DMAX_CH1_FIFO_DEPTH	64
DMAX_CH1_MAX_MSIZE	256
‘DMAX_CH1_MAX_BLOCK_TS	524287
DMAX_CH1_SMS	0X0
DMAX_CH1_DMS	0X0
DMAX_CH1_STW	0
DMAX_CH1_DTW	0
DMAX_CH1_SHADOW_REG_EN	1
DMAX_CH1_MULTI_BLK_EN	1

Configuration Parameter	Value
DMAX_CH1_LLI_PREFETCH_EN	1
DMAX_CH1_LMS	0X0
DMAX_CH1_SRC_STAT_EN	1
DMAX_CH1_DST_STAT_EN	1
DMAX_CH1_LLI_WB_EN	1
DMAX_CH2_FIFO_DEPTH	64
DMAX_CH2_MAX_MSIZE	256
DMAX_CH2_MAX_BLOCK_TS	524287
DMAX_CH2_SMS	0X0
DMAX_CH2_DMS	0X0
DMAX_CH2_STW	0
DMAX_CH2_DTW	0
DMAX_CH2_SHADOW_REG_EN	1
DMAX_CH2_MULTI_BLK_EN	1
DMAX_CH2_LLI_PREFETCH_EN	1
DMAX_CH2_LMS	0X0
DMAX_CH2_SRC_STAT_EN	1
DMAX_CH2_DST_STAT_EN	1
DMAX_CH2_LLI_WB_EN	1
DMAX_CH3_FIFO_DEPTH	64
DMAX_CH3_MAX_MSIZE	256
‘DMAX_CH3_MAX_BLOCK_TS	524287
DMAX_CH3_SMS	0X0
DMAX_CH3_DMS	0X0
DMAX_CH3_STW	0
DMAX_CH3_DTW	0
DMAX_CH3_SHADOW_REG_EN	1
DMAX_CH3_MULTI_BLK_EN	1
DMAX_CH3_LLI_PREFETCH_EN	1

Configuration Parameter	Value
DMAX_CH3_LMS	0X0
DMAX_CH3_SRC_STAT_EN	1
DMAX_CH3_DST_STAT_EN	1
DMAX_CH3_LLI_WB_EN	13
DMAX_CH4_FIFO_DEPTH	32
DMAX_CH4_MAX_MSIZE	256
DMAX_CH4_MAX_BLOCK_TS	524287
DMAX_CH4_SMS	0X0
DMAX_CH4_DMS	0X0
DMAX_CH4_STW	0
DMAX_CH4_DTW	0
DMAX_CH4_SHADOW_REG_EN	1
DMAX_CH4_MULTI_BLK_EN	1
DMAX_CH4_LLI_PREFETCH_EN	1
DMAX_CH4_LMS	0X0
DMAX_CH4_SRC_STAT_EN	1
DMAX_CH4_DST_STAT_EN	1
DMAX_CH4_LLI_WB_EN	1
DMAX_CH5_FIFO_DEPTH	32
DMAX_CH5_MAX_MSIZE	256
DMAX_CH5_MAX_BLOCK_TS	524287
DMAX_CH5_SMS	0X0
DMAX_CH5_DMS	0X0
DMAX_CH5_STW	0
DMAX_CH5_DTW	0
DMAX_CH5_SHADOW_REG_EN	1
DMAX_CH5_MULTI_BLK_EN	1
DMAX_CH5_LLI_PREFETCH_EN	1
DMAX_CH5_LMS	0X0

Configuration Parameter	Value
DMAX_CH5_SRC_STAT_EN	1
DMAX_CH5_DST_STAT_EN	1
DMAX_CH5_LLI_WB_EN	1
DMAX_CH6_FIFO_DEPTH	32
DMAX_CH6_MAX_MSIZE	256
DMAX_CH6_MAX_BLOCK_TS	524287
DMAX_CH6_SMS	0X0
DMAX_CH6_DMS	0X0
DMAX_CH6_STW	0
DMAX_CH6_DTW	0
DMAX_CH6_SHADOW_REG_EN	1
DMAX_CH6_MULTI_BLK_EN	1
DMAX_CH6_LLI_PREFETCH_EN	1
DMAX_CH6_LMS	0X0
DMAX_CH6_SRC_STAT_EN	1
DMAX_CH6_DST_STAT_EN	1
DMAX_CH6_LLI_WB_EN	1
DMAX_CH7_FIFO_DEPTH	32
DMAX_CH7_MAX_MSIZE	256
DMAX_CH7_MAX_BLOCK_TS	524287
DMAX_CH7_SMS	0X0
DMAX_CH7_DMS	0X0
DMAX_CH7_STW	0
DMAX_CH7_DTW	0
DMAX_CH7_SHADOW_REG_EN	1
DMAX_CH7_MULTI_BLK_EN	1
DMAX_CH7_LLI_PREFETCH_EN	1
DMAX_CH7_LMS	0X0
DMAX_CH7_SRC_STAT_EN	1

Configuration Parameter	Value
DMAX_CH7_DST_STAT_EN	1
DMAX_CH7_LLI_WB_EN	1
DMAX_CH8_FIFO_DEPTH	64
DMAX_CH8_MAX_MSIZE	256
DMAX_CH8_MAX_BLOCK_TS	524287
DMAX_CH8_SMS	0X0
DMAX_CH8_DMS	0X0
DMAX_CH8_STW	0
DMAX_CH8_DTW	0
DMAX_CH8_SHADOW_REG_EN	1
DMAX_CH8_MULTI_BLK_EN	1
DMAX_CH8_LLI_PREFETCH_EN	1
DMAX_CH8_LMS	0X0
DMAX_CH8_SRC_STAT_EN	1
DMAX_CH8_DST_STAT_EN	1
DMAX_CH8_LLI_WB_EN	1

Table A-3 Typical Configuration 2

Configuration Parameter	Value
DMAX_ID_NUM	0x0
DMAX_ARB_TYPE	1
DMAX_NUM_CHANNELS	8
DMAX_MSTIF_MODE	1
DMAX_HAS_QOS	1
DMAX_NUM_MASTER_IF	1
DMAX_SLVIF_MODE	2
DMAX_SLVIF_CLOCK_MODE	0
DMAX_S_2_C_SYNC_DEPTH	2
DMAX_C_2_S_SYNC_DEPTH	2
DMAX_MSTIF1_CLOCK_MODE	0

Configuration Parameter	Value
DMAX_M1_2_C_SYNC_DEPTH	2
DMAX_C_2_M1_SYNC_DEPTH	2
DMAX_MSTIF2_CLOCK_MODE	0
DMAX_M2_2_C_SYNC_DEPTH	2
DMAX_C_2_M2_SYNC_DEPTH	2
DMAX_UNALIGNED_XFER_EN	0
DMAX_MULT_ARB_EN	0
DMAX_MSTIF1_OSR_LMT	16
DMAX_MSTIF2_OSR_LMT	16
DMAX_NUM_HS_IF	0
DMAX_ASYNC_HS_EN	0
DMAX_HS_SAME_ASYNC_CLK	1
DMAX_HS_2_C_SYNC_DEPTH	2
DMAX_C_2_HS_SYNC_DEPTH	2
DMAX_HS1_ASYNC_CLK	1
DMAX_HS2_ASYNC_CLK	1
DMAX_HS3_ASYNC_CLK	1
DMAX_HS4_ASYNC_CLK	1
DMAX_HS5_ASYNC_CLK	1
DMAX_HS6_ASYNC_CLK	1
DMAX_HS7_ASYNC_CLK	1
DMAX_HS8_ASYNC_CLK	1
DMAX_HS9_ASYNC_CLK	1
DMAX_HS10_ASYNC_CLK	1
DMAX_HS11_ASYNC_CLK	1
DMAX_HS12_ASYNC_CLK	1
DMAX_HS13_ASYNC_CLK	1
DMAX_HS14_ASYNC_CLK	1
DMAX_HS15_ASYNC_CLK	1

Configuration Parameter	Value
DMAX_HS16_ASYNC_CLK	1
DMAX_HS17_ASYNC_CLK	1
DMAX_HS18_ASYNC_CLK	1
DMAX_HS19_ASYNC_CLK	1
DMAX_HS20_ASYNC_CLK	1
DMAX_HS21_ASYNC_CLK	1
DMAX_HS22_ASYNC_CLK	1
DMAX_HS23_ASYNC_CLK	1
DMAX_HS24_ASYNC_CLK	1
DMAX_HS25_ASYNC_CLK	1
DMAX_HS26_ASYNC_CLK	1
DMAX_HS27_ASYNC_CLK	1
DMAX_HS28_ASYNC_CLK	1
DMAX_HS29_ASYNC_CLK	1
DMAX_HS30_ASYNC_CLK	1
DMAX_HS31_ASYNC_CLK	1
DMAX_HS32_ASYNC_CLK	1
DMAX_HS33_ASYNC_CLK	1
DMAX_HS34_ASYNC_CLK	1
DMAX_HS35_ASYNC_CLK	1
DMAX_HS36_ASYNC_CLK	1
DMAX_HS37_ASYNC_CLK	1
DMAX_HS38_ASYNC_CLK	1
DMAX_HS39_ASYNC_CLK	1
DMAX_HS40_ASYNC_CLK	1
DMAX_HS41_ASYNC_CLK	1
DMAX_HS42_ASYNC_CLK	1
DMAX_HS43_ASYNC_CLK	1
DMAX_HS44_ASYNC_CLK	1

Configuration Parameter	Value
DMAX_HS45_ASYNC_CLK	1
DMAX_HS46_ASYNC_CLK	1
DMAX_HS47_ASYNC_CLK	1
DMAX_HS48_ASYNC_CLK	1
DMAX_HS49_ASYNC_CLK	1
DMAX_HS50_ASYNC_CLK	1
DMAX_HS51_ASYNC_CLK	1
DMAX_HS52_ASYNC_CLK	1
DMAX_HS53_ASYNC_CLK	1
DMAX_HS54_ASYNC_CLK	1
DMAX_HS55_ASYNC_CLK	1
DMAX_HS56_ASYNC_CLK	1
DMAX_HS57_ASYNC_CLK	1
DMAX_HS58_ASYNC_CLK	1
DMAX_HS59_ASYNC_CLK	1
DMAX_HS60_ASYNC_CLK	1
DMAX_HS61_ASYNC_CLK	1
DMAX_HS62_ASYNC_CLK	1
DMAX_HS63_ASYNC_CLK	1
DMAX_HS64_ASYNC_CLK	1
DMAX_INTR_IO_TYPE	2
DMAX_INTR_SYNC2SLVCLK	0
DMAX_SLVIF_STATUS_OP_EN	1
DMAX_CORE_STATUS_OP_EN	1
DMAX_HOLD_IO_EN	1
DMAX_CH_ABORT_EN	1
DMAX_STATIC_ENDIAN_SELECT_MSTIF	1
DMAX_ENDIAN_FORMAT_MSTIF	0
DMAX_LLI_ENDIAN_SELECTION_PIN_EN	0

Configuration Parameter	Value
DMAX_CH_MEM_EXT	1
DMAX_S_DATA_WIDTH	32
DMAX_SAFETY_FEATURE_EN	1
DMAX_PARITY_PROT_EN	1
DMAX_DEF_PARITY_MODE	1
DMAX_HWDATA_PIPE_LINE	0
DMAX_ECC_PROT_EN	1
DMAX_ECC_MEMIF_EN	1
DMAX_ECC_UIDMEMIF_EN	1
DMAX_ECC_ERR_INJ_EN	1
DMAX_MXIF_ECC_PAR_EN	1
DMAX_MXIF_READY_VALID_PAR_EN	1
DMAX_ECC_DIAG_EN	1
DMAX_CH_MEM_REGOUT	1
DMAX_LS_PROT_EN	1
DMAX_LS_DIAG_EN	1
DMAX_M_ADDR_WIDTH	40
DMAX_M_DATA_WIDTH	64
DMAX_M_ID_WIDTH	6
DMAX_M_BURSTLEN_WIDTH	8
DMAX_M_ADDR_FIFO_DEPTH	4
DMAX_M_DATA_FIFO_DEPTH	8
DMAX_M_BRESP_FIFO_DEPTH	4
DMAX_ENABLE_LAST_WRITE	1
DMAX_CSLP_EN	1
DMAX_DEBUG_PORTS_EN	1
DMAX_CHNL_CSLP_EN	1
DMAX_SBIU_CSLP_EN	1
DMAX_MXIF_CSLP_EN	1

Configuration Parameter	Value
DMAX_GLCH_LPDLY_WIDTH	4
DMAX_SBIU_LPDLY_WIDTH	4
DMAX_MXIF_LPDLY_WIDTH	4
DMAX_GLCH_LPDLY	4
DMAX_SBIU_LPDLY	4
DMAX_MXIF_LPDLY	4
DMAX_CH1_FIFO_DEPTH	64
DMAX_CH1_MAX_MSIZE	128
DMAX_CH1_MAX_BLOCK_TS	524287
DMAX_CH1_MAX_AMBA_BURST_LENGTH	32
DMAX_CH1_LOCK_EN	0
DMAX_CH1_TT_FC	0x8
DMAX_CH1_FC	0
DMAX_CH1_TT	0
DMAX_CH1_SMS	0x0
DMAX_CH1_DMS	0x0
DMAX_CH1_STW	0
DMAX_CH1_DTW	0
DMAX_CH1_SHADOW_REG_EN	1
DMAX_CH1_MULTI_BLK_EN	1
DMAX_CH1_MULTI_BLK_TYPE	0x0
DMAX_CH1_LLI_PREFETCH_EN	1
DMAX_CH1_LMS	0x0
DMAX_CH1_SRC_STAT_EN	0
DMAX_CH1_DST_STAT_EN	0
DMAX_CH1_LLI_WB_EN	1
DMAX_CH1_RD_UID_EN	1
DMAX_CH1_WR_UID_EN	1
DMAX_CH1_RD_UID	2

Configuration Parameter	Value
DMAX_CH1_WR_UID	2
DMAX_CH2_FIFO_DEPTH	64
DMAX_CH2_MAX_MSIZE	128
DMAX_CH2_MAX_BLOCK_TS	524287
DMAX_CH2_MAX_AMBA_BURST_LENGTH	32
DMAX_CH2_LOCK_EN	0
DMAX_CH2_TT_FC	0x8
DMAX_CH2_FC	0
DMAX_CH2_TT	0
DMAX_CH2_SMS	0x0
DMAX_CH2_DMS	0x0
DMAX_CH2_STW	0
DMAX_CH2_DTW	0
DMAX_CH2_SHADOW_REG_EN	1
DMAX_CH2_MULTI_BLK_EN	1
DMAX_CH2_MULTI_BLK_TYPE	0x0
DMAX_CH2_LLI_PREFETCH_EN	1
DMAX_CH2_LMS	0x0
DMAX_CH2_SRC_STAT_EN	0
DMAX_CH2_DST_STAT_EN	0
DMAX_CH2_LLI_WB_EN	1
DMAX_CH2_RD_UID_EN	1
DMAX_CH2_WR_UID_EN	1
DMAX_CH2_RD_UID	2
DMAX_CH2_WR_UID	2
DMAX_CH3_FIFO_DEPTH	64
DMAX_CH3_MAX_MSIZE	128
DMAX_CH3_MAX_BLOCK_TS	524287
DMAX_CH3_MAX_AMBA_BURST_LENGTH	32

Configuration Parameter	Value
DMAX_CH3_LOCK_EN	0
DMAX_CH3_TT_FC	0x8
DMAX_CH3_FC	0
DMAX_CH3_TT	0
DMAX_CH3_SMS	0x0
DMAX_CH3_DMS	0x0
DMAX_CH3_STW	0
DMAX_CH3_DTW	0
DMAX_CH3_SHADOW_REG_EN	1
DMAX_CH3_MULTI_BLK_EN	1
DMAX_CH3_MULTI_BLK_TYPE	0x0
DMAX_CH3_LLI_PREFETCH_EN	1
DMAX_CH3_LMS	0x0
DMAX_CH3_SRC_STAT_EN	0
DMAX_CH3_DST_STAT_EN	0
DMAX_CH3_LLI_WB_EN	1
DMAX_CH3_RD_UID_EN	1
DMAX_CH3_WR_UID_EN	1
DMAX_CH3_RD_UID	2
DMAX_CH3_WR_UID	2
DMAX_CH4_FIFO_DEPTH	64
DMAX_CH4_MAX_MSIZE	128
DMAX_CH4_MAX_BLOCK_TS	524287
DMAX_CH4_MAX_AMBA_BURST_LENGTH	32
DMAX_CH4_LOCK_EN	0
DMAX_CH4_TT_FC	0x8
DMAX_CH4_FC	0
DMAX_CH4_TT	0
DMAX_CH4_SMS	0x0

Configuration Parameter	Value
DMAX_CH4_DMS	0x0
DMAX_CH4_STW	0
DMAX_CH4_DTW	0
DMAX_CH4_SHADOW_REG_EN	1
DMAX_CH4_MULTI_BLK_EN	1
DMAX_CH4_MULTI_BLK_TYPE	0x0
DMAX_CH4_LLI_PREFETCH_EN	1
DMAX_CH4_LMS	0x0
DMAX_CH4_SRC_STAT_EN	0
DMAX_CH4_DST_STAT_EN	0
DMAX_CH4_LLI_WB_EN	1
DMAX_CH4_RD_UID_EN	1
DMAX_CH4_WR_UID_EN	1
DMAX_CH4_RD_UID	2
DMAX_CH4_WR_UID	2
DMAX_CH5_FIFO_DEPTH	64
DMAX_CH5_MAX_MSIZE	128
DMAX_CH5_MAX_BLOCK_TS	524287
DMAX_CH5_MAX_AMBA_BURST_LENGTH	32
DMAX_CH5_LOCK_EN	0
DMAX_CH5_TT_FC	0x8
DMAX_CH5_FC	0
DMAX_CH5_TT	0
DMAX_CH5_SMS	0x0
DMAX_CH5_DMS	0x0
DMAX_CH5_STW	0
DMAX_CH5_DTW	0
DMAX_CH5_SHADOW_REG_EN	1
DMAX_CH5_MULTI_BLK_EN	1

Configuration Parameter	Value
DMAX_CH5_MULTI_BLK_TYPE	0x0
DMAX_CH5_LLI_PREFETCH_EN	1
DMAX_CH5_LMS	0x0
DMAX_CH5_SRC_STAT_EN	0
DMAX_CH5_DST_STAT_EN	0
DMAX_CH5_LLI_WB_EN	1
DMAX_CH5_RD_UID_EN	1
DMAX_CH5_WR_UID_EN	1
DMAX_CH5_RD_UID	2
DMAX_CH5_WR_UID	2
DMAX_CH6_FIFO_DEPTH	64
DMAX_CH6_MAX_MSIZE	128
DMAX_CH6_MAX_BLOCK_TS	524287
DMAX_CH6_MAX_AMBA_BURST_LENGTH	32
DMAX_CH6_LOCK_EN	0
DMAX_CH6_TT_FC	0x8
DMAX_CH6_FC	0
DMAX_CH6_TT	0
DMAX_CH6_SMS	0x0
DMAX_CH6_DMS	0x0
DMAX_CH6_STW	0
DMAX_CH6_DTW	0
DMAX_CH6_SHADOW_REG_EN	1
DMAX_CH6_MULTI_BLK_EN	1
DMAX_CH6_MULTI_BLK_TYPE	0x0
DMAX_CH6_LLI_PREFETCH_EN	1
DMAX_CH6_LMS	0x0
DMAX_CH6_SRC_STAT_EN	0
DMAX_CH6_DST_STAT_EN	0

Configuration Parameter	Value
DMAX_CH6_LLI_WB_EN	1
DMAX_CH6_RD_UID_EN	1
DMAX_CH6_WR_UID_EN	1
DMAX_CH6_RD_UID	2
DMAX_CH6_WR_UID	2
DMAX_CH7_FIFO_DEPTH	64
DMAX_CH7_MAX_MSIZE	128
DMAX_CH7_MAX_BLOCK_TS	524287
DMAX_CH7_MAX_AMBA_BURST_LENGTH	32
DMAX_CH7_LOCK_EN	0
DMAX_CH7_TT_FC	0x8
DMAX_CH7_FC	0
DMAX_CH7_TT	0
DMAX_CH7_SMS	0x0
DMAX_CH7_DMS	0x0
DMAX_CH7_STW	0
DMAX_CH7_DTW	0
DMAX_CH7_SHADOW_REG_EN	1
DMAX_CH7_MULTI_BLK_EN	1
DMAX_CH7_MULTI_BLK_TYPE	0x0
DMAX_CH7_LLI_PREFETCH_EN	1
DMAX_CH7_LMS	0x0
DMAX_CH7_SRC_STAT_EN	0
DMAX_CH7_DST_STAT_EN	0
DMAX_CH7_LLI_WB_EN	1
DMAX_CH7_RD_UID_EN	1
DMAX_CH7_WR_UID_EN	1
DMAX_CH7_RD_UID	2
DMAX_CH7_WR_UID	2

Configuration Parameter	Value
DMAX_CH8_FIFO_DEPTH	64
DMAX_CH8_MAX_MSIZE	128
DMAX_CH8_MAX_BLOCK_TS	524287
DMAX_CH8_MAX_AMBA_BURST_LENGTH	32
DMAX_CH8_LOCK_EN	0
DMAX_CH8_TT_FC	0x8
DMAX_CH8_FC	0
DMAX_CH8_TT	0
DMAX_CH8_SMS	0x0
DMAX_CH8_DMS	0x0
DMAX_CH8_STW	0
DMAX_CH8_DTW	0
DMAX_CH8_SHADOW_REG_EN	1
DMAX_CH8_MULTI_BLK_EN	1
DMAX_CH8_MULTI_BLK_TYPE	0x0
DMAX_CH8_LLI_PREFETCH_EN	1
DMAX_CH8_LMS	0x0
DMAX_CH8_SRC_STAT_EN	0
DMAX_CH8_DST_STAT_EN	0
DMAX_CH8_LLI_WB_EN	1
DMAX_CH8_RD_UID_EN	1
DMAX_CH8_WR_UID_EN	1
DMAX_CH8_RD_UID	2
DMAX_CH8_WR_UID	2

B

Internal Parameter Descriptions

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

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

Table B-1 Internal Parameters

Parameter Name	Equals To
DMAX_ADDR_PARITY_PROT_EN	DMAX_PARITY_PROT_EN
DMAX_AR_PYLD0_USER_WIDTH	[function_of: 0 0 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_AR_PYLD0_WIDTH]
DMAX_AR_PYLD0_WIDTH	(DMAX_M_ADDR_WIDTH)
DMAX_AR_PYLD1_USER_WIDTH	[function_of: 0 1 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_AR_PYLD1_WIDTH]
DMAX_AR_PYLD1_WIDTH	((DMAX_HAS_QOS == 1) ? (DMAX_M_ID_WIDTH + DMAX_M_BURSTLEN_WIDTH + 3 + 2 + DMAX_AXI_LOCK_WIDTH + 4 + 3 + 4) : (DMAX_M_ID_WIDTH + DMAX_M_BURSTLEN_WIDTH + 3 + 2 + DMAX_AXI_LOCK_WIDTH + 4 + 3))
DMAX_AW_PYLD0_USER_WIDTH	[function_of: 1 0 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_AW_PYLD0_WIDTH]
DMAX_AW_PYLD0_WIDTH	(DMAX_M_ADDR_WIDTH)
DMAX_AW_PYLD1_USER_WIDTH	[function_of: 1 1 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_AW_PYLD1_WIDTH]

Parameter Name	Equals To
DMAX_AW_PYLD1_WIDTH	((DMAX_HAS_QOS == 1) ? (DMAX_M_ID_WIDTH + DMAX_M_BURSTLEN_WIDTH + 3 + 2 + DMAX_AXI_LOCK_WIDTH + 4 + 3 + 4) : (DMAX_M_ID_WIDTH + DMAX_M_BURSTLEN_WIDTH + 3 + 2 + DMAX_AXI_LOCK_WIDTH + 4 + 3))
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_B_PYLD0_USER_WIDTH	[function_of: 4 0 DMAX_MXIF_B_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_B_PYLD0_WIDTH]
DMAX_B_PYLD0_WIDTH	(DMAX_M_ID_WIDTH + 2)
DMAX_CH1_EXTMEM_DATA_WIDTH	((DMAX_ECC_MEMIF_EN == 1) ? DMAX_CH1_FIFO_WIDTH + DMAX_CH1_FIFO_ECC_WIDTH : DMAX_CH1_FIFO_WIDTH)
DMAX_CH1_FIFO_ECCIN_WIDTH	[function_of: 1]
DMAX_CH1_FIFO_ECC_WIDTH	[function_of: DMAX_CH1_FIFO_ECCIN_WIDTH]
DMAX_CH1_FIFO_WIDTH	((DMAX_CH1_STW ==0 || DMAX_CH1_DTW == 0) ? DMAX_M_DATA_WIDTH : (DMAX_CH1_STW > DMAX_CH1_DTW ? DMAX_CH1_STW : DMAX_CH1_DTW))
DMAX_CH1_REORDER_BUFF_DEPTH	DMAX_CH1_FIFO_DEPTH
DMAX_CH1_UIDMEM_DATA_WIDTH	((DMAX_ECC_UIDMEMIF_EN == 1) ? DMAX_CH1_FIFO_WIDTH + DMAX_CH1_UIDMEM_ECC_WIDTH : DMAX_CH1_FIFO_WIDTH)
DMAX_CH1_UIDMEM_ECCIN_WIDTH	[function_of: 1]
DMAX_CH1_UIDMEM_ECC_WIDTH	[function_of: DMAX_CH1_UIDMEM_ECCIN_WIDTH]
DMAX_COMP_VER	0x3230312a
DMAX_DEF_PARITY_EN	0x1
DMAX_ECC_MXIF_EN	0
DMAX_HAS_LLI_PARAM	[function_of:]
DMAX_HAS_RUID_PARAM	[function_of:]
DMAX_HS_CLKRST_WIDTH	(DMAX_HS_SAME_ASYNC_CLK ==1 ? 1: DMAX_NUM_HS_IF)

Parameter Name	Equals To
DMAX_M_ARCH0_ECC_WIDTH	[function_of: DMAX_M_ADDR_WIDTH]
DMAX_M_ARCH1_ECCIN_WIDTH	[function_of:]
DMAX_M_ARCH1_ECC_WIDTH	[function_of: DMAX_M_ARCH1_ECCIN_WIDTH]
DMAX_M_ARUSER_WIDTH	((DMAX_ECC_MXIF_EN == 1) ? (DMAX_M_ARCH0_ECC_WIDTH + DMAX_M_ARCH1_ECC_WIDTH) : (DMAX_AR_PYLD0_USER_WIDTH + DMAX_AR_PYLD1_USER_WIDTH))
DMAX_M_AWCH0_ECC_WIDTH	[function_of: DMAX_M_ADDR_WIDTH]
DMAX_M_AWCH1_ECCIN_WIDTH	[function_of:]
DMAX_M_AWCH1_ECC_WIDTH	[function_of: DMAX_M_AWCH1_ECCIN_WIDTH]
DMAX_M_AWUSER_WIDTH	((DMAX_ECC_MXIF_EN == 1) ? (DMAX_M_AWCH0_ECC_WIDTH + DMAX_M_AWCH1_ECC_WIDTH) : (DMAX_AW_PYLD0_USER_WIDTH + DMAX_AW_PYLD1_USER_WIDTH))
DMAX_M_BCH_ECCIN_WIDTH	[function_of:]
DMAX_M_BCH_ECC_WIDTH	[function_of: DMAX_M_BCH_ECCIN_WIDTH]
DMAX_M_BUSER_WIDTH	((DMAX_ECC_MXIF_EN == 1) ? (DMAX_M_BCH_ECC_WIDTH) : (DMAX_B_PYLD0_USER_WIDTH))
DMAX_M_RCH0_ECC_WIDTH	[function_of: DMAX_M_DATA_WIDTH]
DMAX_M_RCH1_ECCIN_WIDTH	[function_of:]
DMAX_M_RCH1_ECC_WIDTH	[function_of: DMAX_M_RCH1_ECCIN_WIDTH]
DMAX_M_RUSER_WIDTH	((DMAX_ECC_MXIF_EN == 1) ? (DMAX_M_RCH0_ECC_WIDTH + DMAX_M_RCH1_ECC_WIDTH) : (DMAX_R_PYLD0_USER_WIDTH + DMAX_R_PYLD1_USER_WIDTH))
DMAX_M_WCH0_ECC_WIDTH	[function_of: DMAX_M_DATA_WIDTH]
DMAX_M_WCH1_ECCIN_WIDTH	[function_of:]
DMAX_M_WCH1_ECC_WIDTH	[function_of: DMAX_M_WCH1_ECCIN_WIDTH]
DMAX_M_WUSER_WIDTH	((DMAX_ECC_MXIF_EN == 1) ? (DMAX_M_WCH0_ECC_WIDTH + DMAX_M_WCH1_ECC_WIDTH) : (DMAX_W_PYLD0_USER_WIDTH + DMAX_W_PYLD1_USER_WIDTH))

Parameter Name	Equals To
DMAX_MXIF_B_ECC_PAR_MODE	((DMAX_MXIF_ECC_PAR_MODE==2) ? 0 : DMAX_MXIF_ECC_PAR_MODE)
DMAX_MXIF_PAR_MODE	1
DMAX_NUM_CHANNELS_MUL_3	(DMAX_NUM_CHANNELS*3)
DMAX_R_PYLD0_USER_WIDTH	[function_of: 3 0 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_R_PYLD0_WIDTH]
DMAX_R_PYLD0_WIDTH	(DMAX_M_DATA_WIDTH)
DMAX_R_PYLD1_USER_WIDTH	[function_of: 3 1 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_R_PYLD1_WIDTH]
DMAX_R_PYLD1_WIDTH	(DMAX_M_ID_WIDTH + 2 + 1)
DMAX_S_ADDR_WIDTH	32
DMAX_W_PYLD0_USER_WIDTH	[function_of: 2 0 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_W_PYLD0_WIDTH]
DMAX_W_PYLD0_WIDTH	(DMAX_M_DATA_WIDTH)
DMAX_W_PYLD1_USER_WIDTH	[function_of: 2 1 DMAX_MXIF_ECC_PAR_MODE DMAX_MXIF_PAR_MODE DMAX_W_PYLD1_WIDTH]
DMAX_W_PYLD1_WIDTH	((DMAX_MSTIF_MODE == 0) ? (DMAX_M_ID_WIDTH + (DMAX_M_DATA_WIDTH/8) + 1) : ((DMAX_M_DATA_WIDTH/8) + 1))
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)
LOG2_DMAX_ARB_WR_REQ_WIDTH	((DMAX_HAS_LLI_PARAM ==1) ? LOG2_3_DMAX_NUM_CHANNELS : LOG2_2_DMAX_NUM_CHANNELS)
LOG2_DMAX_CH1_FIFO_DEPTH	[function_of: DMAX_CH1_FIFO_DEPTH]
LOG2_DMAX_CH1_UIDMEM_DEPTH	[function_of: DMAX_CH1_REORDER_BUFF_DEPTH]
LOG2_DMAX_NUM_CHANNELS	[function_of: DMAX_NUM_CHANNELS]
LOG2_DMAX_NUM_HS_IF	[function_of: DMAX_NUM_HS_IF]
MXIF_VLD_RDY_ERRINTR_WIDTH	(5 * DMAX_NUM_MASTER_IF)
SNPS_RSVDPARAM_120	[function_of:]
SNPS_RSVDPARAM_148	DMAX_LS_DIAG_EN

C

Glossary

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

decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWare Synthesizable Components.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
Manager	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.
Subordinate	Device or model that is controlled by and responds to a manager.
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.