



DesignWare® DW_axi

Databook

DW_axi – **Product Code**

Copyright Notice and Proprietary Information

© 2020 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
Organization	13
Related Documentation	14
Web Resources	14
Customer Support	14
Product Code	15
Chapter 1	
Product Overview	17
1.1 DesignWare Synthesizable Components for AMBA/ AXI Overview	17
1.2 General Product Description	19
1.2.1 DW_axi Block Diagram	19
1.3 Features	21
1.3.1 General	21
1.3.2 Master Port	21
1.3.3 Slave Port	22
1.3.4 System Decoder	22
1.3.5 Default Slave	22
1.3.6 Slave Visibility	22
1.3.7 Timing Isolation	23
1.3.8 Single Master System Optimization	23
1.3.9 Out-of-Order Deadlock Avoidance	23
1.3.10 Deadlock Notification	23
1.3.11 Low-Power Interface	23
1.3.12 Quality of Service (QoS) Controller	23
1.3.13 AXI4 Support	24
1.3.14 DW_axi Terminology	24
1.4 Standards Compliance	27
1.5 Verification Environment Overview	27
Chapter 2	
Functional Description	29
2.1 Master Port	29
2.2 Slave Port	30
2.3 Address and Data Bus Architectures	31
2.3.1 Default Multiple Address, Multiple Data (MAMD) Architecture	31
2.4 Hybrid Architectures	32
2.4.1 Shared Layer	33

2.4.2 Hybrid Shared and Dedicated Layers	34
2.4.3 Shared Layer to Dedicated Layer Link	35
2.4.4 Shared Layer Functions	37
2.4.5 Shared Layer Limitations	39
2.5 Default Slave	40
2.6 System Decoders	41
2.6.1 Overlapping Slave Addresses	41
2.6.2 Multiple Address Regions	42
2.6.3 Remap Operation	42
2.6.4 Slave Visibility	43
2.6.5 Trustzone Support	44
2.7 External Decoder	44
2.8 Arbitration	46
2.8.1 Arbitration Scenarios	46
2.8.2 Arbitration Types	46
2.8.3 Internal versus External Priorities	50
2.8.4 Arbitration and Waited Transfers	51
2.9 Atomic Accesses	52
2.9.1 Locked Transfers	52
2.9.2 Exclusive Accesses	53
2.10 ID Bus	53
2.11 Out-of-Order Deadlock Avoidance	54
2.11.1 Example of Out-of-Order Deadlock	54
2.12 Read Data/Write Response Re-ordering	56
2.13 Write Data Interleaving	56
2.13.1 Stalling of Write Data Channel and Write Data Interleaving	58
2.13.2 Write Data Interleaving and Single Master AXI System	60
2.14 Read Data Interleaving	60
2.15 Early Write Data	60
2.16 Clocks and Resets	61
2.17 Stalling of the Payload Source	61
2.17.1 Read Address Channel / Write Address Channel	61
2.17.2 Write Data Channel	62
2.17.3 Read Data Channel/Write Response Channel	63
2.18 Bidirectional Command Support	63
2.18.1 System Masters	64
2.18.2 Interconnecting Masters	64
2.19 Avoiding Combinatorial Loops Between Tiles	67
2.20 Multitile Deadlock Prevention	67
2.21 Deadlock Notification	68
2.22 Timing Mode Options	72
2.22.1 Avoiding Feedback Loops	72
2.22.2 Combinatorial	72
2.22.3 Forward Registered	73
2.22.4 Fully Registered	75
2.22.5 Pipeline Channel Arbiter	77
2.22.6 Multicycle Arbitration	78
2.22.7 Write Address Channel to Write Data Channel Registering	80
2.22.8 External Register Slice	80

2.23 Low-Power Interface	82
2.23.1 cactive Signal De-assertion	84
2.23.2 cactive Signal Assertion	84
2.24 QoS Controller	85
2.24.1 QoS Regulator	87
2.24.2 QoS Architecture	90
2.25 ACE-Lite Support	91
Chapter 3	
Programming the DW_axi	93
3.1 Software Registers	93
3.1.1 Programming Considerations	93
3.1.2 QoS Internal Register Write Operation	93
3.1.3 QoS Internal Register Read Operation	95
3.2 DW_axi Configuration for QoS	96
3.3 How to Program QoS Registers	97
3.3.1 Programming Sequence Examples	98
Chapter 4	
Parameter Descriptions	103
4.1 Top Level Parameters	105
4.2 AXI Source Code Configuration Parameters	110
4.3 AXI Deadlock Notification Configuration Parameters	111
4.4 Pipelining Options / Channel Pipelining Stages Parameters	112
4.5 Pipelining Options / AW to W Registering Parameters	116
4.6 Pipelining Options / Shared Layer Pipelining Parameters	117
4.7 Pipelining Options / Multi-Cycle Arbitration Parameters	119
4.8 Mult-Tile/Bi-directional Command / System Master Numbers Parameters	125
4.9 Mult-Tile/Bi-directional Command / Interconnecting Slave Ports Parameters	126
4.10 Mult-Tile/Bi-directional Command / Interconnecting Master Port configuration Parameters	127
4.11 Arbitration Options / Arbitration Type Parameters	128
4.12 Arbitration Options / Master and Slaves Priority Parameters	138
4.13 Sideband Signals Parameters	140
4.14 User Signals Parameters	142
4.15 Slave Visibility Parameters	144
4.16 Architecture Options / Architecture Options Parameters	145
4.17 Slave Address Map Parameters	150
4.18 Master Port Configuration Parameters	155
4.19 Slave Port Configuration Parameters	158
4.20 QoS Options / Register Interface Configuration Parameters	161
4.21 QoS Options / QoS Signals Parameters	162
4.22 QoS Options / QoS Regulator Module Configuration Parameters	163
Chapter 5	
Register Descriptions	165
5.1 axi_memory_map/axi_address_block Registers	169
5.1.1 AXI_VERSION_ID_REG	170
5.1.2 AXI_HW_CFG_REG	171
5.1.3 AXI_CMD_REG	174
5.1.4 AXI_DATA_REG	180

Chapter 6

Signal Descriptions	181
6.1 Clocks and Resets Signals	183
6.2 APB Interface Signals	185
6.3 Master Port x Read Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals	188
6.4 Master Port x Write Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals	194
6.5 Master Port x Write Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals	199
6.6 Master Port x Write Response Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals	202
6.7 Master Port x Read Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals	205
6.8 AXI Low Power Interface Signals	208
6.9 Deadlock Notification Signals	209
6.10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals	211
6.11 Slave Port j Write Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals	217
6.12 Slave Port j Write Response Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals	220
6.13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals	222
6.14 Slave Port j Read Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals	229
6.15 Default Slave Write Address Channel Signals	232
6.16 Default Slave Write Data Channel Signals	235
6.17 Default Slave Write Response Channel Signals	237
6.18 Default Slave Read Address Channel Signals	238
6.19 Default Slave Read Data Channel Signals	241

Chapter 7

Internal Parameter Descriptions	243
---------------------------------------	-----

Chapter 8

Verification	245
8.1 Overview of Test Environments	245
8.1.1 SystemVerilog Test Environment (SVTE)	245
8.1.2 VIP Wrapper Tasks	246
8.1.3 SVTE Task Usage	246
8.1.4 SVTE APB Usage	247
8.1.5 SVTE Tasks	247

Chapter 9

Integration Considerations	255
9.1 Performance	255
9.1.1 Power Consumption, Frequency, Area and DFT Coverage	255
9.1.2 Latency	258
9.2 Usage Requirements	259
9.3 Timing Exceptions	260
9.4 High Frequency Design Methods	260
9.4.1 Timing Mode Options	260
9.4.2 External Register Slice	260
9.4.3 Tiling Interconnects	261
9.5 Quality of Service (QoS) Integration	265
9.5.1 Considerations	266
9.5.2 Restrictions	267

Appendix A

DW_axi Application Notes	269
A.1 Multitile Deadlock Scenarios	269
A.1.1 Bidirectional Multitile Systems	269
A.1.2 Multitile Systems	270
A.1.3 Multiple Paths from Master to Slave	271
Appendix B	
Basic Core Module (BCM) Library	273
B.1 BCM Library Components	273
B.2 Synchronizer Methods	273
B.2.1 Synchronizers Used in DW_axi	274
B.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_axi)	274
Appendix C	
Glossary	275
Index	279

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 2.06b onward.

Version	Date	Description
4.04a	March 2020	<ul style="list-style-type: none"> Revision version change for 2020.03a release Updated synthesis results in “Performance” on page 255 Added “Clocks and Resets” on page 61 Added “Timing Exceptions” on page 260 Added “BCM Library Components” on page 273 Renamed Synchronizer Methods chapter to Appendix B, “Basic Core Module (BCM) Library” “Parameter Descriptions” on page 103, “Signal Descriptions” on page 181, “Register Descriptions” on page 165, “Internal Parameter Descriptions” on page 243 auto-extracted from the RTL
4.03a	February 2018	<ul style="list-style-type: none"> Revision version change for 2018.02a release Updated synthesis results in “Performance” on page 255 Updated the axi_memory_map/axi_address_block registers description Removed Chapter 2 Building and Verifying a Component or Subsystem from the Databook and added the contents in the newly created User Guide “Parameter Descriptions” on page 103, “Signal Descriptions” on page 181, “Register Descriptions” on page 165, “Internal Parameter Descriptions” on page 243 auto-extracted from the RTL with change bars
4.02a	March 2016	<ul style="list-style-type: none"> Revision version change for 2016.03a release Added “Running SpyGlass® Lint and SpyGlass® CDC” Added “Running Spyglass on Generated Code with coreAssembler” “Parameter Descriptions” on page 103, “Signal Descriptions” on page 181, and “Register Descriptions” on page 165 auto-extracted from the RTL Added “Internal Parameter Descriptions” on page 243 Modified “Master Port” on page 29 and “Write Data Channel” on page 62 sections to indicate that the slave lookup table is updated only when a write command is received. Added Appendix B, “Basic Core Module (BCM) Library” Updated area and power numbers in “Performance” on page 255

Version	Date	Description
4.01a	March 2014	<ul style="list-style-type: none"> Revision version change for 2014.10a release Updated for 64-bit address in “Slave Address Map” in “Parameter Descriptions” on page 103 Updated performance section in “Integration Considerations” chapter Modified Figure 2-4 Modified the note in “Write Address Channel to Write Data Channel Registering”
4.00a	June 2013	Added information about new and modified features, signals, and parameters to support the AMBA 4 AXI and ACE-Lite specifications. Added information regarding the SystemVerilog Test Environment.
3.01a	May 2013	<p>Added a section to describe the QoS Controller and updated the databook for QoS functionality. Added a section in the “Parameter Descriptions” on page 103 chapter for “QoS Options” and “APB Configuration”. Added QoS signals and APB configuration signals in the Signal Descriptions chapter. Added a section regarding Quality of Service (QoS) Integration in the Integration Considerations chapter.</p> <p>Added “Registers” chapter, and Chapter 3, “Programming the DW_axi”.</p> <p>Enhanced the maximum limit of the AXI_MAX_URIDA_M(x) and AXI_MAX_UWIDA_M(x) parameters.</p> <p>Enhanced sideband signal descriptions for increased width to 256 bits.</p> <p>Enhanced the maximum limit of the AXI_MAX_FARC_S(j), AXI_MAX_FAWC_S(j), AXI_MAX_RCA_ID_M(i), and AXI_MAX_WCA_ID_M(i) parameters.</p> <p>Enhanced the User-Defined Arbitration Type section.</p> <p>Added a note about connecting a slave to a single master in the Multicycle Arbitration section.</p> <p>Enhanced the Latency subsection.</p> <p>Added a section for integration with an External Register Slice in the Integration Considerations chapter.</p> <p>Updated the template.</p>
2.13a	Oct 2012	Added the product code on the cover and in Table 1-1 .
2.13a	Oct 2011	Made master and slave variable designations consistent.
2.12a	Jun 2011	Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface.
2.12a	Mar 2011	Removed number-specific region statement in “Slave Address Map” section.
2.12a	Jan 2011	Removed inappropriate mention of debug “mode” in Signals chapter.
2.12a	Jan 2011	Corrected title for Figure 3-4; added information for new deadlock notification feature.
2.11a	Nov 2010	Added information for overlapping addresses, and warning about combinatorial loops between tiles in bidirectional systems.

Version	Date	Description
2.09a	Sept 2010	Clarification that fair-among-equals arbitration does not imply first-come-first-served among equals; added labels for multicycle arbitration parameters; clarified AXI_MAX_UWIDA_M(i) and AXI_MAX_URIDA_M(i) descriptions; corrected names of include files and vcs command used for simulation; corrected value for AXI_REG_AW_W_PATHS when operating frequency reduced.
2.07a	Apr 2010	Enhancement in “Bidirectional Deadlock Prevention” section.
2.05a	Feb 2010	Added information regarding deadlock prevention; added latency optimization information in “Write Address to Write Data Channel Registering” section.
2.01a	Dec 2009	Removed demux in valid path of shared layer illustrations; added Master3 and Master4 information in “Hybrid Shared and Dedicated Layers Configuration” table; updated databook to new template for consistency with other IIP/VIP/PHY databooks.
2.01a	Sep 2009	Version change for 2009.08a release.
2.00a	Jul 2009	Added information for Hybrid Architecture; removed references to Quickstarts, as they are no longer supported; corrected AXI_MIDW_Sx to AXI_WID_S(j); changed AXI_WID_S(j) default value to 1; version change for 2009.07a.
1.18a	March 9, 2009	Edited AXI_WID_S(j) default to 2 in order to match GUI for current release.
1.18a	Mar 2009	Edited “Avoiding Feedback Loops” section, Figure 20, “Forward Registered” section, Figure 22, “Fully Registered” section, “Pipeline Channel Arbiter” section, and AXI_WID_S(j) default.
1.17a	Jan 2009	Version change for 2009.01a release.
1.16a	Dec 2008	Version change for 2008.12a release.
1.15a	Oct 2008	Note text changed in “Read Data Interleaving” section; version change for 2008.10a release.
1.14a	Jul 28, 2008	Added note about connecting two instances in tiled fashion; located in “Write Data Interleaving” and “Bidirectional Command Support” sections.
1.14a	Jul 18, 2008	Added “Read Data Interleaving” subsection and AXI_RI_LIMIT_M(i) parameter. Corrected default value and enhanced AXI_ALLOW_MSTn_ICM(i) parameter. Enhanced “Interconnecting Masters” section.
1.13a	Jun 2008	Version change for 2008.06a release.
1.12a	Apr 2008	AXI_NUM_SYS_MASTERS increased up to 64 system masters; corrected minimum value for AXI_ALLOW_MSTn_ICMi parameter.
1.10a	Jan 2008	Edited Multiple Address Regions section and Slave Address Map material; enhanced Timing Mode Options section.
1.10a	Oct 2007	Changes to subsection for the multicycle arbitration options.
1.09a	Sep 28, 2007	Additional subsection for the multicycle arbitration options.

Version	Date	Description
1.08a	Sep 10, 2007	Corrected information about interleaving depth of external master.
1.08a	Aug 21, 2007	Corrected AXI_MAX_U[R/W]IDA_M(<i>i</i>) parameter requirements.
1.08a	Aug 20, 2007	Added dependency to AXI_NUM_SYS_MASTERS.
1.08a	Aug 8, 2007	Additions to “Forward Registered” and “Fully Registered” timing mode options; additional subsections also added for “Pipeline Channel Arbiter” and “External Register Slice”.
1.07a	Jun 2007	Early Write Data section added; arbitration schemes revised; details added in Timing Mode Options; AXI_HAS_BICMD parameter description enhanced.

Preface

This databook provides information about the DesignWare Advanced eXtensible Interface (DW_axi) interconnect. This component conforms to the AMBA 3 AXI, AMBA 4 AXI, and ACE-Lite specifications defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

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

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.
- [Chapter 3, “Programming the DW_axi”](#) provides information needed to program the configured DW_axi.
- [Chapter 4, “Parameter Descriptions”](#) identifies the configurable parameters supported by the DW_axi.
- [Chapter 5, “Register Descriptions”](#) describes the programmable registers of the DW_axi.
- [Chapter 6, “Signal Descriptions”](#) provides a list and description of the DW_axi signals.
- [Chapter 7, “Internal Parameter Descriptions”](#) provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals, Registers, and Parameters chapters.
- [Chapter 8, “Verification”](#) provides information on verifying the configured DW_axi.
- [Chapter 9, “Integration Considerations”](#) includes information you need to integrate the configured DW_axi into your design.
- [Appendix A, “DW_axi Application Notes”](#) provides application notes for DW_axi.
- [Appendix B, “Basic Core Module \(BCM\) Library”](#), documents the synchronizer methods (blocks of synchronizer functionality) used in DW_axi to cross clock boundaries.
- [Appendix C, “Glossary”](#) provides a glossary of general terms.

Related Documentation

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

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 3 AXI, refer to the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI*.

Web Resources

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

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

- Prepare the following debug information, if applicable:
 - For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:
File > Build Debug Tar-file
Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the `<core tool startup directory>/debug.tar.gz` file.
 - For simulation issues outside of coreConsultant or coreAssembler:
 - Create a waveforms file (such as VPD or VCD).
 - Identify the hierarchy path to the DesignWare instance.
 - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- For the fastest response, enter a case through SolvNetPlus:
 - a. <https://solvnetplus.synopsys.com>

**Note**

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

- b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).
- c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

Ensure to include the following:

- **Product L1:** DesignWare Library IP
- **Product L2:** <name of L2>

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

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

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

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

Product Code

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

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

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multilayer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI and AMBA 4 AXI
DW_axi_a2x	Configurable bridge between AMBA 3 AXI/AMBA 4 AXI components and AHB components or between AMBA 3 AXI/AMBA 4 AXI components and AMBA 3 AXI/AMBA 4 AXI components.

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

Component Name	Description
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_hmx	Configurable high performance interface from an AHB master to an AMBA 3 AXI and AMBA 4 AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI or AMBA 4 AXI subsystems
DW_axi_x2h	Bridge from AMBA 3 AXI or AMBA 4 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or busses

1

Product Overview

The DW_axi component is a multilayer interconnect implementation of the AXI protocol, which is designed for high-performance, high-frequency system designs. DW_axi is highly configurable with the capacity to handle up to:

- 16 masters
- 16 slaves
- 64 address bits
- 512 data bits

1.1 DesignWare Synthesizable Components for AMBA/AXI 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 3 AXI/ AMBA 4 AXI (Advanced eXtensible Interface) components.

**Note**

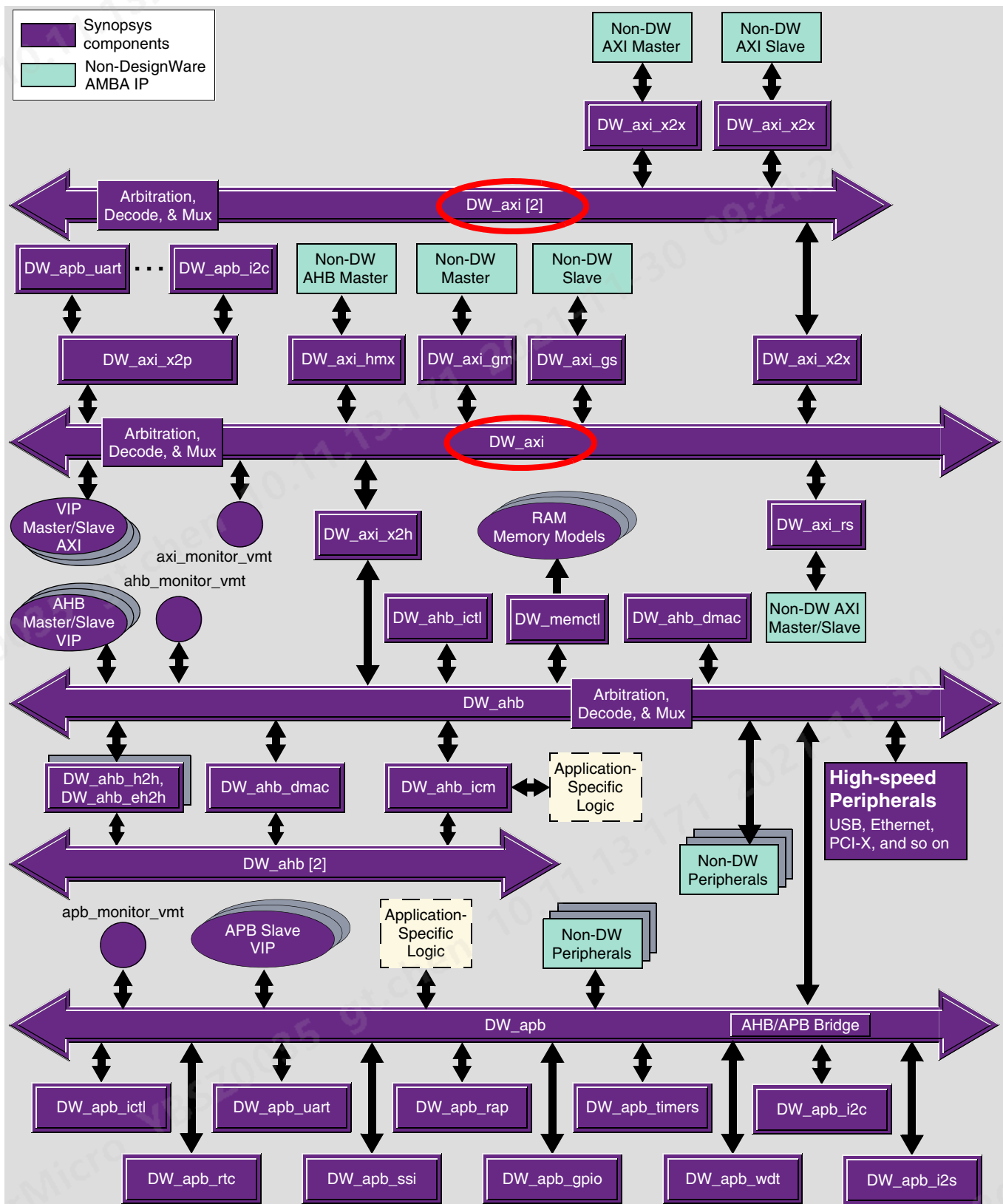
You must have a DWC-AMBA-Fabric-Source license to enable the AXI4 or ACE-Lite interface.

[Figure 1-1](#) illustrates one example of this environment, including the AHB bus, the APB Bus (includes the APB Bridge), AHB multilayer interconnect IP, APB peripheral components, verification Master/Slave models, and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

**Attention**

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

Figure 1-1 Example of DW_axi in a Complete System



You can connect, configure, synthesize, and verify the DW_axi 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 component, you might prefer to use coreConsultant, documentation for which is available in the [coreConsultant User Guide](#).

1.2 General Product Description

DW_axi is an interconnect implementation of the AXI protocol, where simultaneous transfers between different masters and slaves can occur. The component can be configured to support up to 16 masters and 16 slaves. DW_axi is a multiple address, multiple data bus architecture. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction.

This multilayer interconnect architecture allows parallel traffic between different master/slave pairs on all five AXI channels. Therefore, system bandwidth is not limited by DW_axi but by the external master and slaves.

1.2.1 DW_axi Block Diagram

As shown in [Figure 1-2](#) on page 20, DW_axi consists of configurable master and slave ports. An external master connects to DW_axi through a master port; an external slave connects to DW_axi through a slave port. As shown in [Figure 1-2](#), each DW_axi master port is actually an AXI slave interface; similarly, each DW_axi slave port is an AXI master interface. Throughout this databook, the terms master port and slave port are used to have a consistent and simple terminology.

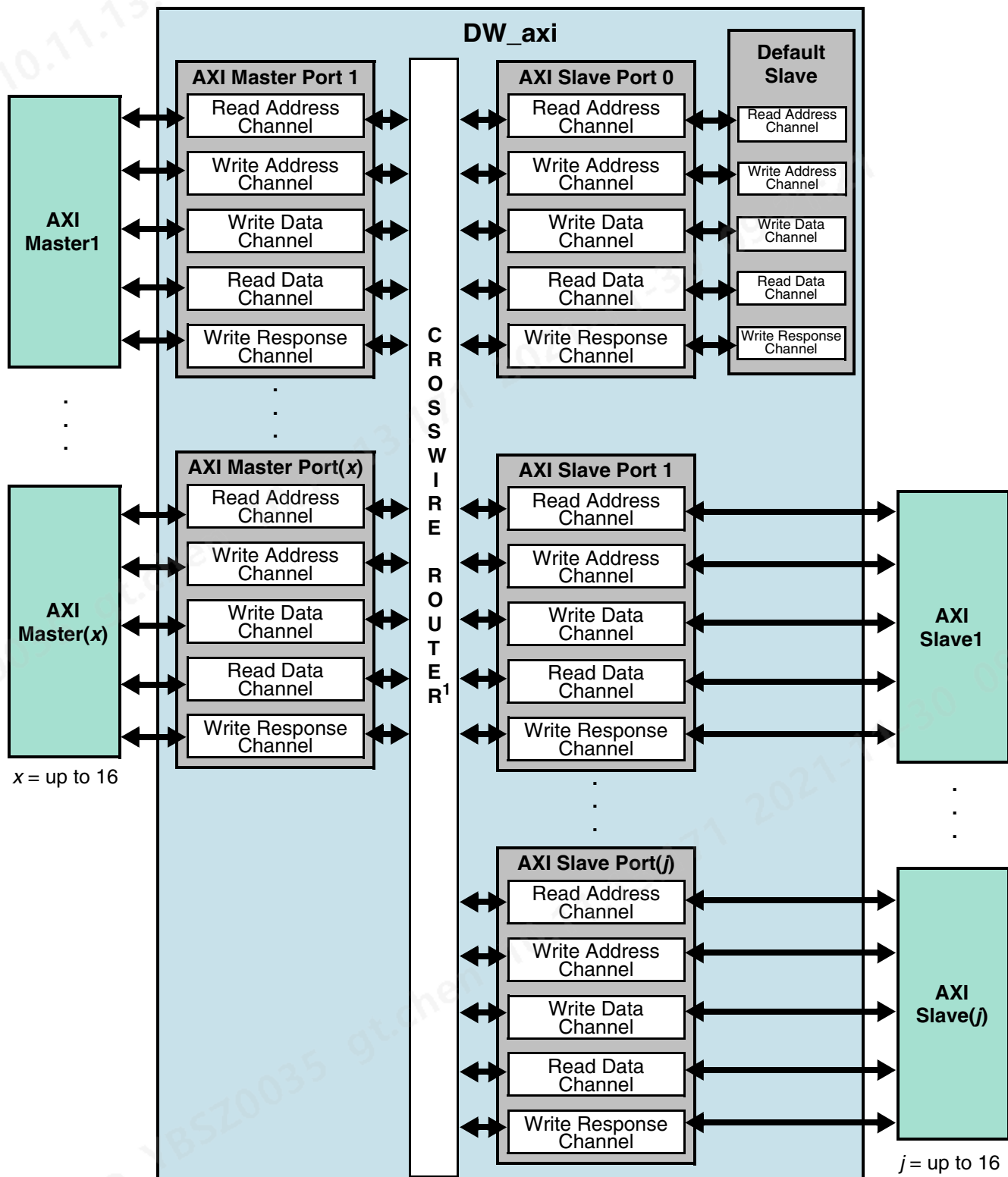
Each master and slave port has five channels:

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

For more information about how the DW_axi master and slave ports function, refer to [“Master Port”](#) on page 29 and [“Slave Port”](#) on page 30.

DW_axi also includes a default slave, which is shared between all address layers. The default slave is internal to the DW_axi and attaches to Slave Port0, which is not available as an external port. The default slave allows parallel processing of read and write commands. For more information, refer to [“Default Slave”](#) on page 40.

Figure 1-2 DW_axi Block Diagram



1. This block does not contain any logic. It generates buses that are driven to each slave and master interface from a concatenation of signals from all master and slave interfaces, respectively.

1.3 Features

The DW_axi supports the following features:

1.3.1 General

- High performance multiple address bus, multiple data bus AXI interconnect architecture
- Complies with the following specifications:
 - AMBA 3 AXI
 - AMBA 4 AXI
 - ACE-Lite (barrier transactions not supported)
- Configurable number of master and slave ports (up to 16 each)
- Configurable system decoder (optional)
- Configurable master and slave priorities used for arbitration can be configured statically or driven dynamically through input ports
- Configurable selection of arbitration scheme
- Default slave included
- Configurable slave visibility per master
- Flexible timing path options
- Built-in out-of-order deadlock avoidance
- Address width of 32 to 64 bits
- Data widths of 8, 16, 32, 64, 128, 256, and 512 bits
- Support for data bursts up to 256 beats
- Equal data widths of master and slave ports
- Single clock frequency for all master and slave ports
- Locked transfer support
- Automatically optimized in single AXI master subsystems
- Optional sideband/user signals for each AXI channel, allowing additional control/status per channel (for example, data parity)
- Support for Quality of Service (QoS) in AXI3, AXI4, and ACE-Lite configurations

1.3.2 Master Port

The following are features of the DW_axi master port:

- Configurable maximum number of:
 - Read commands that master port can accept per unique ARID value for active read commands at any one time

- ❑ Write commands that master port can accept per unique AWID value for active write commands at any one time
- ❑ Unique values of ARID that master port can accept for active read commands at any one time
- ❑ Unique values of AWID that master port can accept for active write commands at any one time
- Read re-order depth not limited by DW_axi master port and dependent on attached slaves
If slave can return data out-of-order, then DW_axi forwards data to targeted master.
- Write response re-order depth not limited by DW_axi master port and dependent on attached slaves
If slave can return write response out-of-order, then DW_axi forwards response to targeted master.
- Write interleave depth of external master(s)
 - ❑ Write data interleaving up to a configurable depth when DW_axi is configured with one master port
 - ❑ Limited to 1 for each master port when DW_axi configured with multiple master ports

1.3.3 Slave Port

The following are features of the DW_axi slave port:

- Configurable write interleave depth for each slave port
The data from multiple masters can be interleaved to the same slave if the slave supports write interleaving.
- In AXI 3 configurations, supports a configurable maximum number of active read/write transactions that slave port can generate

1.3.4 System Decoder

- Internal decoder – all address layers have same memory map
- Remap support – two memory maps supported (normal and boot)
- Multiregion address support
- Trustzone support
- External decoder option – user supplies external decoders, one for each address layer

1.3.5 Default Slave

- Shared by all address layers
- Attached to Slave Port0, which is not available as an external port

1.3.6 Slave Visibility

- Configurable master access to each slave for both normal and boot modes
- Area and timing performance optimized when slave not visible to all masters

1.3.7 Timing Isolation

- Option of adding internal register slices to DW_axi to isolate long timing paths. Register slice can be added to the forward control path of an AXI channel with no throughput penalty.

1.3.8 Single Master System Optimization

The DW_axi automatically eliminates the following unnecessary hardware functions when configured with only a single AXI master¹:

- Write data interleaving rules
- Tracking the number of active read/write transactions generated by a slave port
- AXI locking rules
- Arbitration for slave's read/write address and write data channels when multiple masters request simultaneously. Hence, no muxing of read/write address channels or write data channels in the slave port.

1.3.9 Out-of-Order Deadlock Avoidance

- Automatic read and write command flow control to avoid out-of-order deadlock

1.3.10 Deadlock Notification

- Timeout-based deadlock detection
- Configurable timeout value
- Deadlock debug port provides details about deadlocked transaction (master, slave, and ID)

1.3.11 Low-Power Interface

The features of the DW_axi low-power interface are:

- Transaction activity status to an external low-power controller (LPC) for enabling or disabling the clock.
- Configurable maximum number of clock cycles the system must remain idle before entering low-power mode.

1.3.12 Quality of Service (QoS) Controller

The features of the DW_axi QoS Controller are:

- Regulates incoming traffic on master port based on desired rate
- Prioritizes requests from masters
- Dynamic programming of QoS parameters through the APB interface (refer to “QoS Controller” on page 85 for more information)

1. Hardware parameters are disabled for single master configurations; enabled when AXI_NUM_MASTERS ≥ 2.

**Note**

Quality of Service (QoS) functionality is source-only; that is, you must have a DWC-AMBA-Fabric-Source license in order to use the QoS features.

1.3.13 AXI4 Support

The following features are added in the DW_axi to support the AMBA 4 AXI specification:

- QoS signaling in the AXI4 configuration
- Region signaling support (up to eight regions) through user parameters
- Optional user signals for each AXI channel allowing additional control or status per channel (for example, data parity)

**Note**

AXI4 and ACE-Lite features are source-only; that is, you must have a DWC-AMBA-Fabric-Source license to use AXI4 and ACE-Lite features.

1.3.14 DW_axi Terminology

The following terms are used throughout this databook.

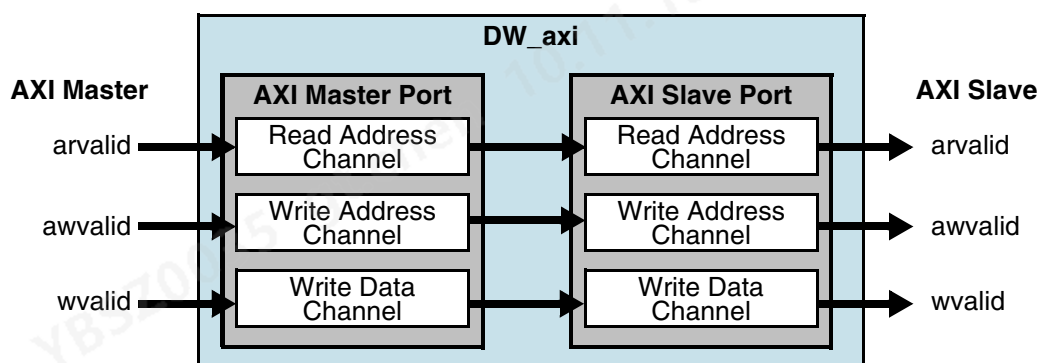
- **Active read/write command** – A command that has been generated by an external AXI master and accepted by an external AXI slave but has not completed.

A read command completes when the last data beat of the read burst completes; for instance, when the AXI slave asserts rlast and rvalid signals, and the AXI master asserts rready.

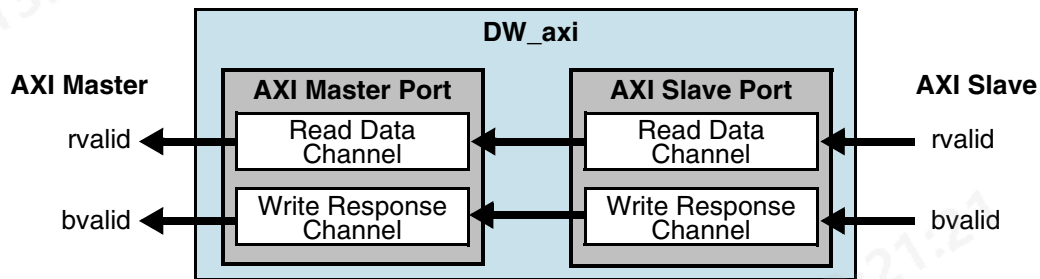
A write command completes when the AXI slave returns the write response and it is accepted by the AXI master; for instance, when the AXI slave asserts bvalid, and the AXI master asserts bready.

- **Forward control path** – For the read/write command channels and the write data channel, the forward control path is in the direction of arvalid/awvalid/wvalid from an external master to an external slave, as shown in [Figure 1-3](#).

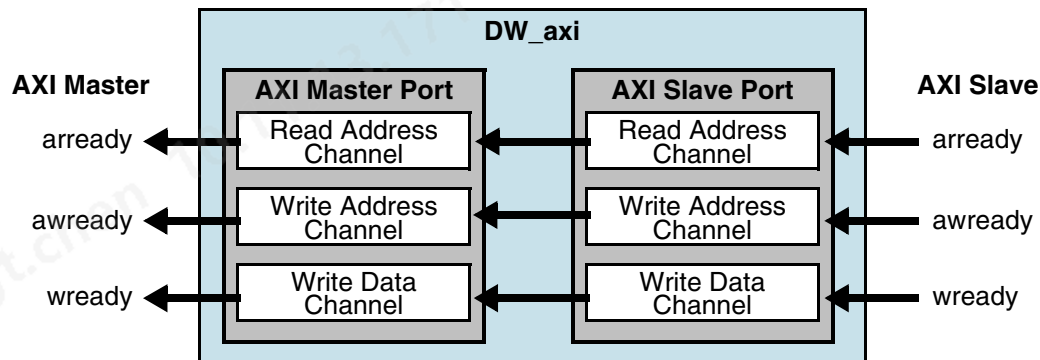
Figure 1-3 Forward Control Path – Read/Write Command and Write Data Channels



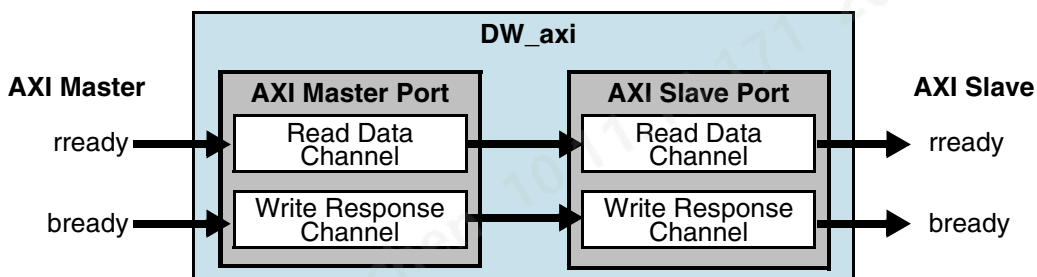
For the read data and write response channels, the forward control path is in the direction of rvalid/bvalid from an external slave to an external master, as shown in [Figure 1-4](#).

Figure 1-4 Forward Control Path – Read/Write Command and Write Data Channels

- **Backward control path** – For the read/write command channels and the write data channel, the backward control path is in the direction of aready/awready/wready from an external slave to an external master, as shown in [Figure 1-5](#).

Figure 1-5 Backward Control Path – Read/Write Command and Write Data Channel

For the read data and write response channels, the backward control path is in the direction of ready/bready from an external master to an external slave, as shown in [Figure 1-6](#).

Figure 1-6 Backward Control Path – Read Data Channel and Write Response Channel

- **AXI payload** – The payload is always in the direction of the forward control path. It is specific to the AXI channel. [Table 1-1](#) provides the payload for each channel group.

Table 1-1 DW_axi Payload^a

Channel	Payload
Read/Write Command	<ul style="list-style-type: none"> Read/write address Control signals Optional signals (such as sideband/user signals, QoS signals, or ACE-Lite signals), if selected
Read/Write Data	<ul style="list-style-type: none"> Read/write data Control signals Optional signals (such as sideband/user signals), if selected
Write Response	<ul style="list-style-type: none"> Burst response Control signals Optional signals (such as sideband/user signals), if selected

a. The control signals mentioned do not include the flow control signals *valid and *ready.

- **Payload source** – The payload source is the sender of the AXI channel payload.
- **Payload destination** – The payload destination is the receiver of the AXI channel payload.

Table 1-2 provides the payload source and destination for each channel.

Table 1-2 DW_axi Payload Source and Destination

Channel	Payload Source	Payload Destination
Read/Write Command	AXI Master	AXI Slave
Write Data	AXI Master	AXI Slave
Read Data	AXI Slave	AXI Master
Write Response	AXI Slave	AXI Master

- **Waited transfer** – An AXI channel is waited if the payload destination de-asserts its *ready signal when the payload source has its *valid signal asserted. Wait states are inserted into the AXI channel transfer.
- **Locked sequence** – A sequence of locked read or write commands from an external AXI master that locks the read and write command channels of the addressed slave. The locking sequence includes both the initial locking command and the final unlocking command that is used to terminate the locked sequence.
- **Locking command** – Initial locking command of the locked sequence.
- **Payload accepted** – AXI channel payload is accepted when both AXI channel flow control signals, *ready and *valid are asserted on the same cycle.
- **MxS DW_axi** – This is a DW_axi interconnect with *M* master ports and *S* slave ports.

1.4 Standards Compliance

The DW_axi component conforms to the *AMBA AXI and ACE Protocol Specification* from ARM. Readers are assumed to be familiar with this specification.

1.5 Verification Environment Overview

The DW_axi includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation. For more information on DW_axi verification, refer to [“Verification”](#) on page 245.

For more information about configuring, synthesizing, and verifying just your DW_axi component, refer to *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_axi component within a DesignWare subsystem using coreAssembler, refer to *DesignWare Synthesizable Components for AMBA 2 User Guide*.

Functional Description

DW_axi is an AXI interconnect that allows AXI masters and slaves to communicate to each other. It is designed for high-performance, high-frequency systems. This chapter provides a functional overview of the components of DW_axi, and then describes each functional feature of the component.

2.1 Master Port

As illustrated in [Figure 1-2](#), a DW_axi master port connects to an external AXI master. You can configure a master port for the following functions:

- Maximum number of active read commands, with the same ARID, that an external master can generate before the DW_axi stalls the master read address channel
- Maximum number of active write commands, with the same AWID, that an external master can generate before the DW_axi stalls the master write address channel
- Maximum number of active read commands, with a different ARID, that an external master can generate before the DW_axi stalls the master read address channel
- Maximum number of active write commands, with different AWID, that an external master can generate before the DW_axi stalls the master write address channel

The master port is responsible for the following tasks:

- Ensures different ARID/AWID values of active read/write commands forwarded to different slaves:

When a master attempts read commands to different slaves, the master port checks to see if the ARID values of the read command is different to a previously issued active read command.

Similarly, when a master attempts write commands to different slaves, the master port checks to see if the AWID values of the write command is different to a previously issued active write command.

The master port enforces these rules by holding off the read and write address channels until the previously issued active command has completed. This is required for out-of-order deadlock avoidance. For more details, see [“Out-of-Order Deadlock Avoidance”](#) on page 54.

- Implements system address decoder:

Because DW_axi is a multiple address architecture, each master port instance has a read decoder and write decoder.

- Generates a request to the slave port connected to the addressed slave for use of the slave's read/write address channels.
- A lookup table with an index is used to route the write data to the correct slave. An entry is added to the slave lookup table when a write command is received by DW_axi master port.
 - In AXI3 configurations the index is taken from the wid_m(x) signal from the master port.
 - In AXI4/ACE-Lite configurations the index is taken from the awid_m(x) signal from the master port.
- Sends a request to slave port connected to targeted AXI slave for use of slave's write data bus when AXI master generates valid write data.

The DW_axi master port only generates the request if a write command, with matching ID, has been previously issued by the external AXI master.

DW_axi stalls the master's write data channel if the DW_axi master port has not previously accepted a matching write command.

- Inserts WAIT states on arready, awready, and wready flow control outputs to the attached external master. For more information, refer to ["Stalling of the Payload Source"](#) on page 61.
- Arbitrates when multiple slaves request simultaneous access to a master's read data and write response buses.
- Multiplexes the read data and write burst response from multiple slaves:

The read data from the AXI slave that wins ownership of the AXI master's read data channel is forwarded to the master. The write response from the AXI slave that wins ownership of the AXI master's write response channel is forwarded to the master.
- Drives rvalid and bvalid signals back to the AXI master.
- Appends awid, arid, and wid values with AXI master number information before forwarding to the DW_axi slave ports.
- Because external masters are only permitted to have active read/write commands to different slaves, if the ARID/AWID values of these active read/write commands are different, then the interconnect does not need to re-order read data/write response coming from multiple slaves to the same master.

2.2 Slave Port

As illustrated in [Figure 1-2](#), a DW_axi slave port connects to an external AXI slave. You can configure a slave port for the following functions:

- Write interleave depth of the slave port; fixed as 1 for AXI4 configurations
- Maximum number of active write commands that can be forwarded by the slave port
- Maximum number of active read commands that can be forwarded by a slave port
- Decoded region signals on ports for write address channel and read address channel

The slave port is responsible for the following tasks:

- Arbitrates when multiple masters request simultaneous access to the read command, write command, or write data buses.

- Multiplexes the read command, write command, or write data buses from the granted master onto the slave's read command, write command, and write data buses, respectively.
- Enforces the following write data interleaving rules:
 - When DW_axi combines write transactions from different AXI masters to one AXI slave, it must ensure that it combines the first beat of write data from each transaction in address order.
 - When DW_axi combines write transactions from different AXI masters, it must do so under the constraints of the write interleave depth supported by the slave port.
- Decodes the read data channel and write response channel IDs (rid and bid) on valid read data and write response assertions from the attached slave, and requests ownership of the targeted master's read data and write response channels.
- Inserts WAIT states on rready and bready inputs to the attached slave. For more details, see [“Stalling of the Payload Source”](#) on page 61.
- Strips AXI master number information from rid and bid before forwarding them to the targeted DW_axi master port.
- For a locked sequence in AXI3 configurations, ensures that both the read and write command arbiters are locked to the same external AXI master. Also, ensures the following:
 - That no other active commands are waiting to complete before the slave port starts a locked sequence of either read or write commands.
 - That the final unlocking command has fully completed before any further commands are accepted and forwarded to the external AXI slave.

2.3 Address and Data Bus Architectures

By default, the DW_axi operates in the multiple address, multiple data (MAMD) architecture mode, but it is also capable of operating in other optional architectures. In the MAMD architecture, each slave has independent read/write address and write data channels. This is opposed to a shared address and data bus architecture where all slaves share the same read/write address and write data buses. Each slave has its own flow control signals, *valid and *ready, regardless of the architecture.

Additionally, in a multiple data bus architecture, each master has a read data bus. This is opposed to the shared data bus architecture where all masters share the same read data bus. Each master has its own flow control signals, *valid and *ready, regardless of the architecture. The same holds for the write response channels.

The following subsections describe the different architectures for which the DW_axi can be configured.

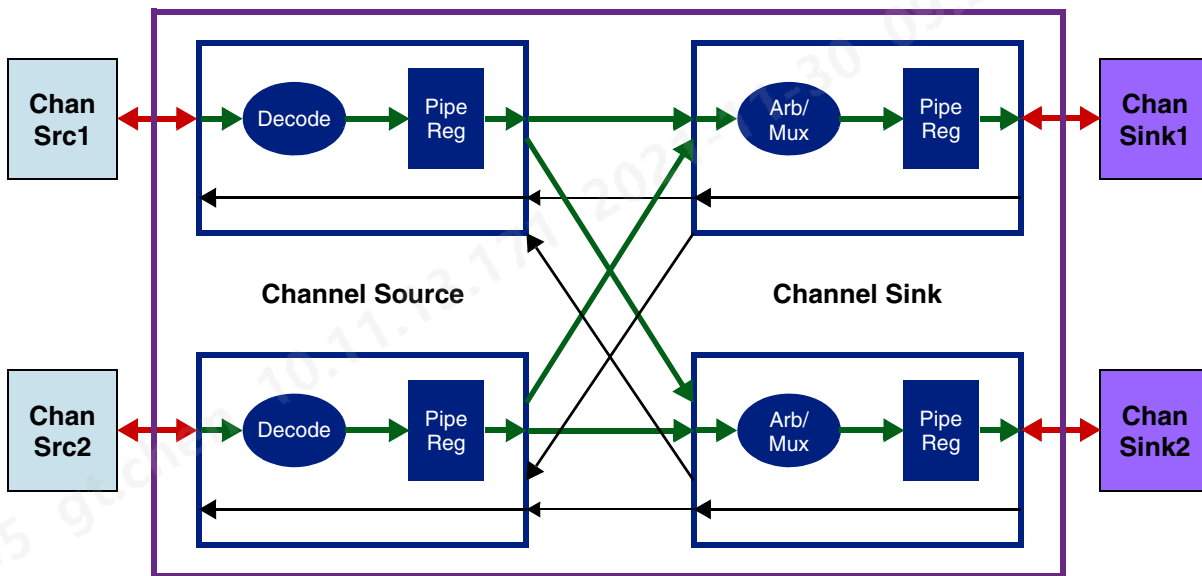
2.3.1 Default Multiple Address, Multiple Data (MAMD) Architecture

[Figure 1-2](#) illustrates the default architecture of the DW_axi. Each master and slave port block contains a block for each of the five AXI channels. Each slave port arbitrates on the read and write address channels and on the write data channel, while each master port arbitrates on the read data and burst response channels. Every slave port has three arbiters – two if there is no write data interleaving to the attached slave – and every master port has two arbiters. Because each master and slave port arbitrate independently for each master or slave that attempts to access it, the masters and slaves operate on dedicated layers.

Figure 2-1 illustrates the architecture of a generic single DW_axi channel. Every channel in the DW_axi by default follows this structure:

- If Figure 2-1 were a write address channel, the channel sources would be masters (master ports) and the channel sinks would be slaves (slave ports).
- If Figure 2-1 were a read data channel, the channel sources would be slaves (slave ports) and the channel sinks would be masters (master ports).

Figure 2-1 Single-Channel Default Architecture



Every channel source generates its own requests, and they are arbitrated at the channel sink block.

This architecture results in the highest performance, since arbitration occurs when two sources attempt to access the same sink on the same channel, and parallel accesses are possible. In this way, the system bandwidth is not limited by the interconnect, but by the external master and slaves.

The penalty of this architecture is logic redundancy for low bandwidth/high latency master/slave links or channels.

2.4 Hybrid Architectures

The DW_axi is capable of operating in several different types of architectures. In particular, the default multiple address, multiple data (MAMD) architecture can be used in different combinations with other architectures:

- Single address, single data (SASD)
- Single address, multiple data (SAMD)
- Multiple address, single data (MASD)

This is known as a hybrid architecture whereby some master-slave links and channels can operate in one architecture, while others can operate in other architectures – all within the same DW_axi instance.



The hybrid architecture functions are available to only those customers who own the DWC AMBA Fabric package. Presence of the source license is checked every time coreConsultant and coreAssembler are invoked to configure DW_axi. If the source license is not found, the hybrid parameters are greyed out, and it is not possible to enable hybrid functionality.

The hybrid architecture allows you to tailor the architecture of the DW_axi to the particular bandwidth requirements of the system. For example, low-bandwidth/high-latency master-slave links or channels can be combined into a shared layer to save gates, power, and wire routing, while high-bandwidth/low-latency master-slave links or channels can be given dedicated layers, just as they would have been in the default MAMD architecture.

You can configure the DW_axi architecture using the Architecture Options tab in the coreConsultant GUI. Under this tab are five tables—one for each of the AXI channels. Each table contains master numbers in the columns and slave numbers in the rows, or vice versa depending on where arbitration takes place for the channel; the port that does arbitration (sink) is on the rows, while the sources are in the columns. At each intersection in the table, you can select whether that master-slave link on that channel is shared or dedicated.

It is from these options that the underlying bus architecture is controlled. With these configuration options, you can tailor the DW_axi bus architecture to suit any application. The following are a few examples of which the DW_axi is capable:

- Multiple address, multiple data
- Single address, single data
- Shared address, multiple data
- Shared address, shared data
- Multiple address, multiple data, shared burst response
- Slaves 1 to 6 share read address, write address, and write data layers; Slaves 7 to 10 have dedicated read address, write address, and write data layers
- Masters 1 to 5 share read data layers; Masters 5 to 10 have dedicated read data layers
- Slave 1 has a dedicated write data layer for Masters 1 to 5, and is accessed through the shared write data layer for Masters 6 to 10

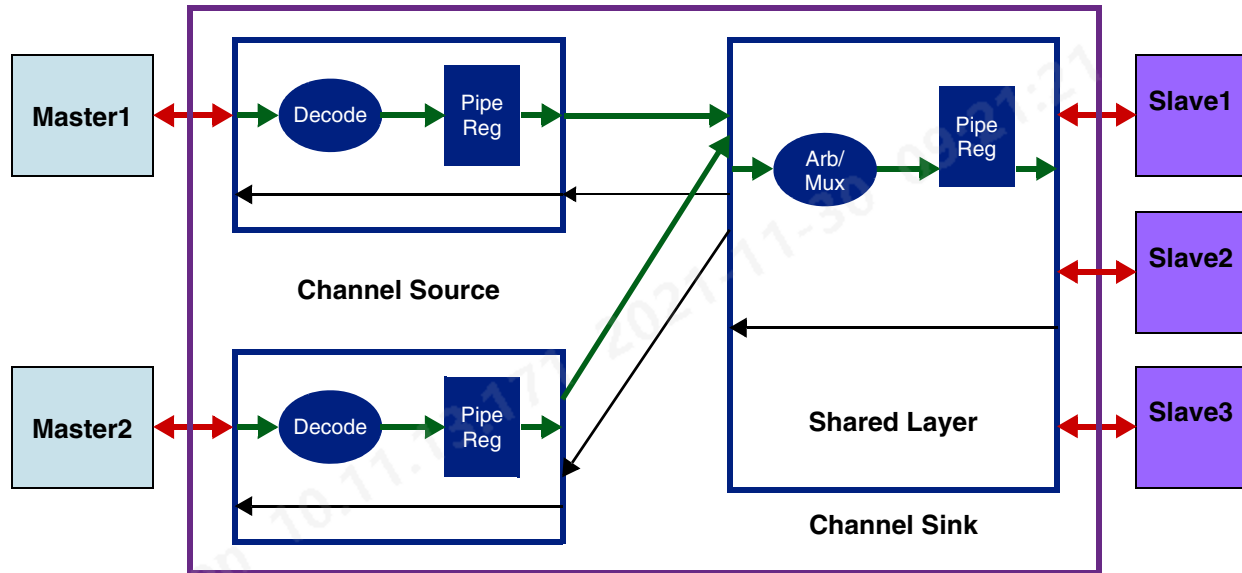
2.4.1 Shared Layer

The DW_axi architecture options are configured using the following coreConsultant parameters:

- AXI_AR_LAYER_S(j)_M(x)
- AXI_AW_LAYER_S(j)_M(x)
- AXI_W_LAYER_S(j)_M(x)
- AXI_R_LAYER_M(x)_S(j)
- AXI_B_LAYER_M(x)_S(j)

You can set each of these parameters to either Shared or Dedicated, with the default being Dedicated. [Figure 2-2](#) illustrates an example of a shared layer. In this case, Master 1 and Master 2 connect to Slave 1, Slave 2, and Slave 3 via the shared layer.

Figure 2-2 Shared Read Address Layer



[Table 2-1](#) lists the parameter settings for the architecture shown in [Figure 2-2](#).

Table 2-1 Shared Layer Configuration

Read Address Channel	Master1	Master2
Slave1	AXI_AR_LAYER_S1_M1 Shared	AXI_AR_LAYER_S1_M2 Shared
Slave2	AXI_AR_LAYER_S2_M1 Shared	AXI_AR_LAYER_S2_M2 Shared
Slave3	AXI_AR_LAYER_S3_M1 Shared	AXI_AR_LAYER_S3_M2 Shared

By selecting the shared read address channel layer, a single shared read address channel block is instantiated, and all channel sinks that select to use the shared read address channel layer are connected to this shared layer. For example, in [Figure 2-2](#) the read address channel arbiters and multiplexing in Slave Ports 1, 2 and 3 are removed because this function is handled by the shared read address channel layer. This illustrates how area is saved by using the hybrid architecture feature. While this example uses the read address channel, the same applies to all DW_axi channels.

2.4.2 Hybrid Shared and Dedicated Layers

The “Shared Layer” section illustrated an example of a two-master, three-slave DW_axi interconnect where all the read address channels are shared. [Figure 2-3](#) illustrates a hybrid architecture configuration on the read address channel of a two-master, three-slave DW_axi instance, where Master 1 and Master 2 access

Slave1 and Slave2 through the shared layer, but access Slave3 through the dedicated read address channel layer of Slave3.

Figure 2-3 Hybrid Shared and Dedicated Layers Read Address Channel

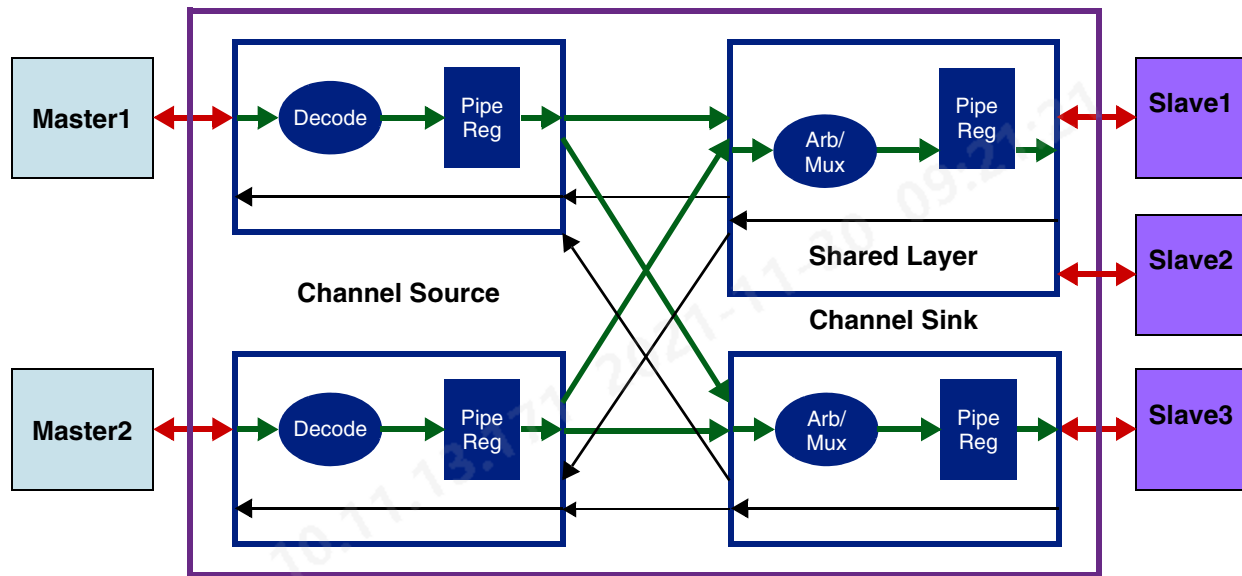


Table 2-2 lists the parameter settings for the architecture shown in Figure 2-3.

Table 2-2 Hybrid Shared and Dedicated Layers Configuration

Read Address Channel	Master1	Master2
Slave1	AXI_AR_LAYER_S1_M1 Shared	AXI_AR_LAYER_S1_M2 Shared
Slave2	AXI_AR_LAYER_S2_M1 Shared	AXI_AR_LAYER_S2_M2 Shared
Slave3	AXI_AR_LAYER_S3_M1 Dedicated	AXI_AR_LAYER_S3_M2 Dedicated

Configuring an architecture like the one shown in Figure 2-3 shows the Master 1-to-Slave 3 and Master 2-to-Slave 3 links as high performance—that is, requiring a dedicated layer—whereas the links from Master 1 and Master 2 to Slave 1 and Slave 2 are relatively low-performance links.

2.4.3 Shared Layer to Dedicated Layer Link

Figure 2-4 shows an example of a channel sink—in this case, the Slave 4 read address channel—connected to both its own dedicated layer and the read address channel shared layer. This type of architecture is ideal when a channel sink has a high bandwidth/low latency requirement with some sources, and a low bandwidth/high latency requirement with other channel sources. This provides maximum flexibility in tailoring the architecture to the application. For example in Figure 2-4, Master 4 could be a central DMA master that requires high bandwidth to Slave 3, whereas Master 1 may only perform initial setup programming to Slave 4 and as such does not require a high bandwidth link.

Figure 2-4 Shared to Dedicated Layer Link

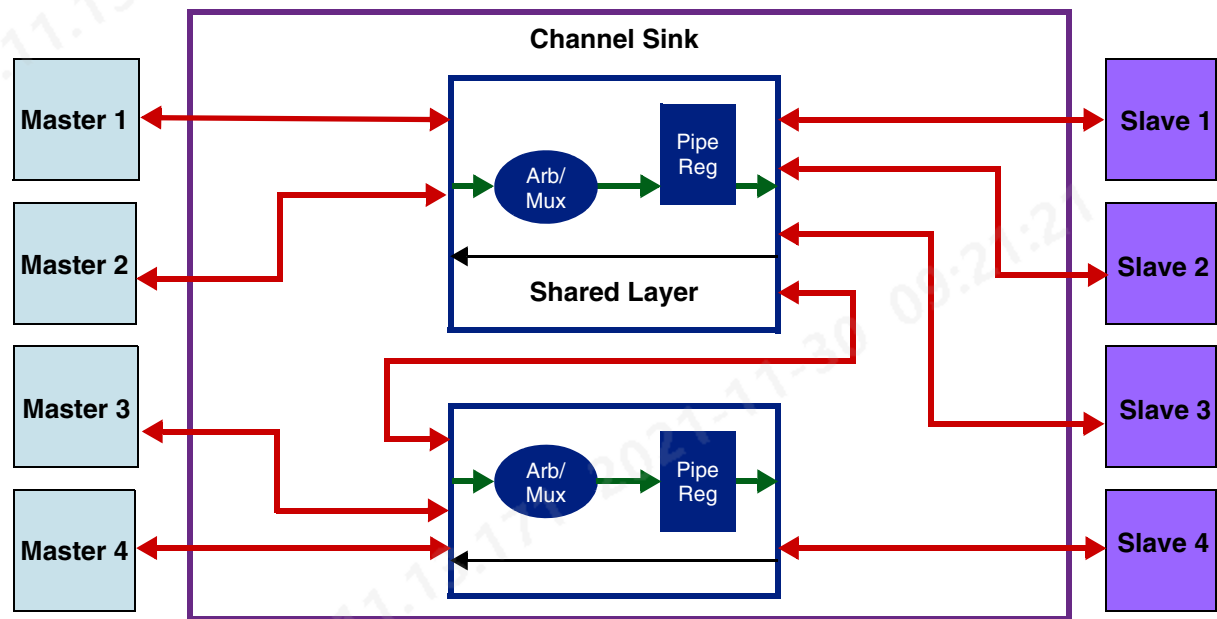


Table 2-3 lists the parameter settings for the architecture shown in Figure 2-4.

Table 2-3 Shared to Dedicated Layer Link Configuration

Read Address Channel	Master1	Master2	Master3	Master4
Slave1	AXI_AR_LAYER_S1_M1 Shared	AXI_AR_LAYER_S1_M2 Shared	No Connect	No Connect
Slave2	AXI_AR_LAYER_S2_M1 Shared	AXI_AR_LAYER_S2_M2 Shared	No Connect	No Connect
Slave3	AXI_AR_LAYER_S3_M1 Shared	AXI_AR_LAYER_S3_M2 Shared	No Connect	No Connect
Slave4	AXI_AR_LAYER_S4_M1 Shared	AXI_AR_LAYER_S4_M2 Shared	AXI_AR_LAYER_S4_M3 Dedicated	AXI_AR_LAYER_S4_M4 Dedicated

When a channel source on a shared layer attempts to access a sink on a dedicated layer, it must first win arbitration on the shared layer and then win arbitration on the dedicated layer. While the shared layer is waiting for a `READY=1` to return from the dedicated layer, the entire shared layer is stalled – transactions buffered into shared layer pipeline stages notwithstanding. With this in mind, you should use only the shared to dedicated layer link for a low bandwidth source-to-sink link; otherwise the source should connect to the particular sink on its dedicated layer.

**Note**

When this type of configuration is used, the shared layer pipeline stage must be used to avoid having a combinatorial path of two arbiters in series; this pipeline stage is described in “[Pipelining](#)” on page 37.

With this type of configuration, the combinatorial logic paths are shortened in this channel. There is full connectivity from each master to each slave, but instead of 3 * 3-client arbiters, there are 2 * 2-client arbiters. If the architecture of the other AXI channels in the DW_axi instance are similarly configured, the longest logic paths can be shortened through the whole instance and synthesized to a higher frequency.

When each shared layer channel arbitrates at a dedicated layer, its priority is set by these coreConsultant parameters or signals:

- AXI_SHARED_LAYER_MASTER_PRIORITY parameter or mst_priority_shared signal– Sets the priority of the read address, write address, and write data shared layer channels. In these channels, the shared layer is requesting on behalf of masters, and the maximum priority value is limited by the number of masters in the DW_axi instance.
- AXI_SHARED_LAYER_SLAVE_PRIORITY parameter or slv_priority_shared signal – Sets the priority of the read data and burst response channels. On these channels, the shared layer is requesting on behalf of slaves, and the maximum priority value is limited by the number of slaves in the DW_axi instance.

**Note**

If AXI_HAS_EXT_PRIORITY is set to 1, the shared layer priority values are controlled by the input ports mst_priority_shared and slv_priority_shared.

2.4.4 Shared Layer Functions

The following subsections describe functions related to the shared layer.

2.4.4.1 Pipelining

Because of the additional multiplexing and demultiplexing required in a shared channel, adding a shared channel to a DW_axi instance has a negative effect on the maximum frequency that can be achieved. To avoid this effect, you can add a pipeline stage to each of the shared channels. Using the shared layer pipeline stage, configurations with a shared layer can synthesize to clock frequencies close to similar configurations with no selected shared layers.

The shared layer pipeline stage is a forward-registered pipeline, with one buffer stage used to avoid any throughput penalty.

Enabling the shared pipeline stage on a particular channel adds one instance of the forward-registered pipeline to that shared channel layer. This is unlike the AXI_*_TMO and AXI_*_PL_ARB timing mode options that add a pipeline to every channel source in the instance – master port for AR, AW and W, and slave port for R and W.

The shared layer pipeline is enabled by default for every shared layer that is configured. It can be disabled with the AXI_*_SHARED_PL parameters, accessible through the coreConsultant and coreAssembler **Pipelining Options > Shared Layer Pipelining** menu item in the Configuration activity.

2.4.4.2 Shared Write Data Channel Pipelining

Due to the extra logic required to implement write data ordering rules, there are longer combinatorial paths in the shared write data channel than the other shared channels. Enabling the AXI_W_SHARED_PL parameter addresses this issue by adding an additional pipelining register to the shared write data channel (additional to the shared layer pipelining stage already inferred by the parameter). This additional register does not add a buffer stage, nor does it add latency to the channel. The register introduces one additional cycle immediately after the point where a master and slave are completing a write data transaction, where the completing master and slave cannot access each other.

**Note**

This additional cycle is not a dead cycle for the whole shared write data channel, as other masters may access other slaves during the cycle.

2.4.4.3 Arbiter Pipeline Stage and Shared Layers

If the arbiter pipeline stage is enabled for an AXI channel, and if the shared layer for that channel is connected with a dedicated layer of that channel, a buffer stage is added on the output of the shared layer. The following parameters determine which arbiter pipeline stage is enabled:

- AXI_AR_PL_ARB
- AXI_R_PL_ARB
- AXI_AW_PL_ARB
- AXI_W_PL_ARB
- AXI_B_PL_ARB

The buffer stage is used for sinks accessed through dedicated layers only and is used to allow the shared layer to operate correctly with registers after the arbiter in the dedicated layers, which are added using the above parameters.

For more information on the links between shared and dedicated layers, refer to [“Shared Layer to Dedicated Layer Link”](#) on page 35.

**Note**

Arbiters in a shared layer are pipelined if the arbiter pipeline stage is enabled for that channel.

2.4.4.4 Write Data Interleaving

The shared write data channel layer supports the following interleaving depths:

- From masters – write interleaving depth of 1
- To slave – write interleaving depth of 1

**Note**

Write data interleaving is an AXI3 feature. In all AXI4 configurations, the write interleaving depth is set at 1.

In single-master DW_axi configurations, the single master is allowed to interleave to a dedicated write data channel. However, the shared write data channel does not support interleaving from masters in either single-master or multiple-master configurations.

The shared write data channel layer interleaves data from different masters to different slaves, but does not support interleaving from a single master or to a single slave.

If you require a write data interleaving depth greater than 1 to a slave, that slave should be given its own dedicated write data channel layer.

When the write data channel of a slave has a dedicated layer and is also accessed from the shared write data channel layer via the shared-to-dedicated layer link, the dedicated write data layer enforces the write data interleaving depth for that slave. From the point of view of the dedicated write data layer, the shared write data layer appears to be one additional master with a write interleaving depth of 1.

2.4.5 Shared Layer Limitations

The following subsections describe limitations related to the shared layer.

2.4.5.1 Deadlock Avoidance on Write Address Channel

On the write address channel, for configurations with one or more shared-to-dedicated layer links and with either the shared layer pipeline stage (AXI_AW_SHARED_PL) or the arbiter pipeline stage (AXI_AW_PL_ARB) enabled, a deadlock condition exists.

To avoid this deadlock condition in these types of configurations, whenever a master port issues a transaction that is destined for a shared-to-dedicated link, all further transactions are masked until the shared-to-dedicated link transaction has been issued to the endpoint slave.

2.4.5.2 Shared Write Data Channel With Dedicated Write Address Channel

Configuring the write address channel on a master slave link to be dedicated, while the write data channel on the same link is shared, is not supported. Since the overall bandwidth of write transactions is mainly dependent on the write data channel, the dedicated write address link is redundant if the write data channel on the same link is shared.

2.4.5.3 Locking Sequences

Locked sequences are not supported in the following configurations:

- AXI4 or ACE-Lite configurations
- When QoS is enabled in interconnect; that is, AXI_HAS_QOS = 1
- Hybrid architecture

If locking sequences are enabled (that is, AXI_HAS_LOCKING=1), all hybrid architecture parameters are disabled and only the default MAMD architecture can be used.

2.4.5.4 Read Data Interleave Limiting

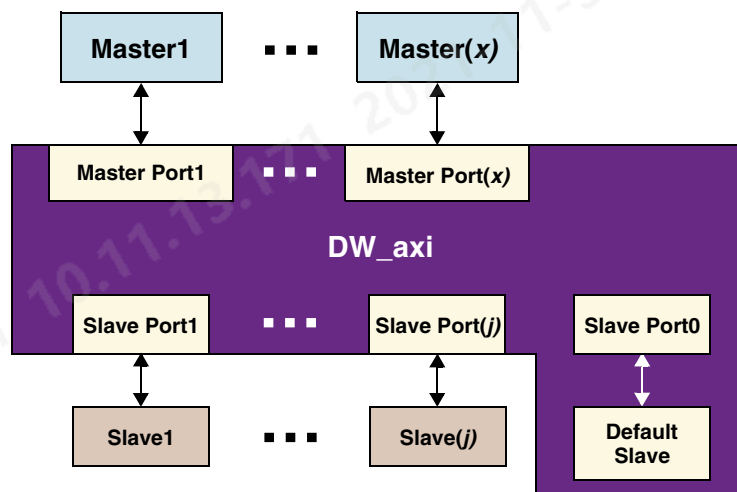
When enabled for a particular master, the AXI_RI_LIMIT_M(x) parameter limits the read data interleaving depth of that master to a value of 1. For more details, refer to [“Read Data Interleaving”](#) on page 60.

Read data interleave limiting is not supported for masters that connect only to the shared read data channel. If a master requires this feature, at least one of the read data channel links to the master must be through the dedicated read data layer.

2.5 Default Slave

The default slave is shared between all address layers and is attached to Slave Port0, which is not available as an external port. The default slave can respond simultaneously to one read and one write command. However, in the case of single-master systems, the default slave accepts an unlimited number of write commands.

Figure 2-5 DW_axi Default Slave



Read and write commands from external masters are routed to the default slave when:

- An external master generates a read/write command where the read/write address is not memory mapped. For more details, see [“System Decoders”](#) on page 41.
- A master is trying to access a secure slave with an insecure access. For more details, see [“Trustzone Support”](#) on page 44.
- An AXI master generates a read/write command to a slave that is not visible to it. For more information, see [“Slave Visibility”](#) on page 43.

The default slave performs the following:

- Accepts at most one active read command and one active write command.
- For a read command:
 - Returns DECERR on the rresp bus for each beat of the read burst. The read data is held constant at 0.
 - Asserts rvalid until the burst is complete. When a read command is accepted by the default slave, rvalid is immediately asserted and remains asserted until the burst completes.
 - Asserts rlast on the last beat of the transfer.
- For a write transfer:

- ❑ Does not insert WAIT states on a data transfer – wready is tied high.
- ❑ Ignores write data.
- ❑ Generates the write response immediately after the write data burst has completed. The bresp signals a DECERR.

2.6 System Decoders

For the multiple address interconnect architecture, each master port has two decoders – a read channel and write channel decoder – that allow parallel processing of read and write commands. Therefore, DW_axi configured with x master ports has a total of $2 \times x$ system decoders.

When an external AXI master generates a valid read/write command, the attached master port's read/write decoder generates the slave number of the addressed slave by decoding the master's read/write address bus. The slave number is used to request access to the attached slave's read/write address channel from the slave port's arbiter.

The default slave is selected by the system decoder if one of the following is true:

- The address bus targets an unmapped system memory location;
- The slave is invisible to that master; or
- The master attempts a non-secure access to a secure slave.

Each system slave address region can be specified with a starting and ending address that must be aligned to 4 KB boundaries.

Each decoder implements the functions discussed in the following sections.

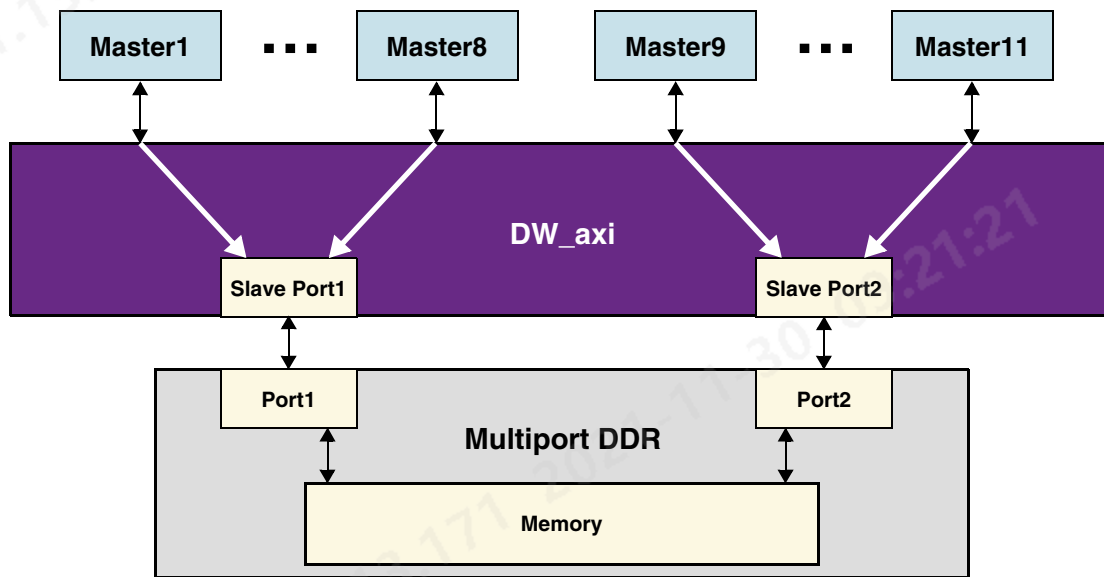
2.6.1 Overlapping Slave Addresses

Any number of slaves can be configured with overlapping address ranges, as long as no master is visible to any two or more slaves that have overlapping address ranges; this applies individually in both normal and boot address modes. For example, given the following:

- Two-master system
- Normal address mode
- M1 is visible to S1, but not visible to S2
- M2 is visible to S2, but not to S1

Under these circumstances, S1 and S2 may have overlapping addresses in normal address mode, regardless of the boot address mode visibility settings.

This can be required in certain situations, such as a multiport DDR that has multiple AXI slave interfaces for performance reasons. In this case, it is desirable for every master to have access to the entire memory range of the DDR memory, regardless of the port to which it connects. [Figure 2-6](#) illustrates this scenario.

Figure 2-6 DDR Memory Address Overlap

Overlapping slave addresses can also be useful in bidirectional multile systems. For more details, refer to [“Bidirectional Command Support”](#) on page 63.

**Note**

coreConsultant and coreAssembler generate a warning message if legal overlapping slave addresses are detected. This occurs in order to guard against unintentionally selecting such a configuration.

2.6.2 Multiple Address Regions

A memory controller with a single select line has a region for internal registers and a region for external memory. From the memory map’s perspective, the internal registers could be at a different locations than the external memory. These can be treated as separate regions without having to assign the entire memory space between the two regions to the memory controller.

A slave does not have to occupy a continuous area of the memory map. DW_axi is designed so that any slave can have up to eight regions (refer to [Figure 2-7](#)) in both Boot and Normal modes; they do not need the same number of regions in both modes. This allows a memory controller, for instance, to have banks at non-contiguous memory locations. For instance, in [Figure 2-7](#), Slave 1 is split into eight regions (S1_R1 through S1_R8) in both Boot and Normal modes.

An AXI4 implementation provides an option for a 4-bit region output for the write address channel and read address channel on slave ports. These region signals must be present only on an interface that is downstream from an address decode function.

2.6.3 Remap Operation

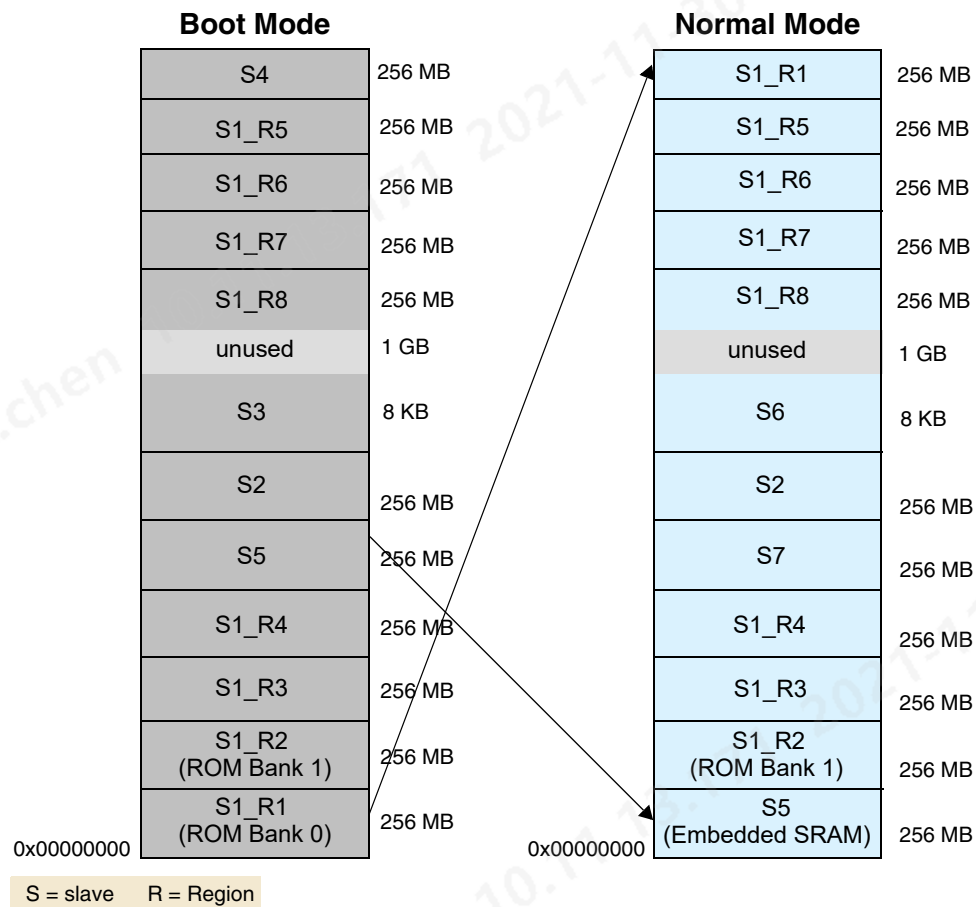
Processors may boot from one memory and then run from another. Often this means that address 0x00000000 needs to be remapped from one memory to another. The DW_axi has two modes of operation to allow for this type of scenario: Boot Mode and Normal Mode. Each mode has its own memory map, which can be identical to or different from the other, depending on your configuration. This functionality is

referred to as the Remap Feature that allows you to configure the slave for both modes. You enable Remap by setting the AXI_REMAP_EN configuration parameter to True.

If this feature is not enabled, configuration options are available for only Normal Mode. [Figure 2-7](#) illustrates memory maps for both Boot and Normal modes. In the Boot Mode example, a ROM (Slave 1) occupies the base address 0x00000000, whereas an embedded RAM (Slave 5) is remapped to occupy the address location of 0x00000000 in Normal Mode.

When the remap feature is enabled, selection of the different memory maps is under the control of the input signal remap_n. In Boot Mode, the remap_n signal is logic 0, whereas it is logic 1 in Normal Mode.

Figure 2-7 Example DW_axi Memory Map – Boot Mode and Normal Mode



2.6.4 Slave Visibility

The AXI system can be viewed as a collection of layers. On each layer there is an AXI master and all the slaves with which it can communicate.

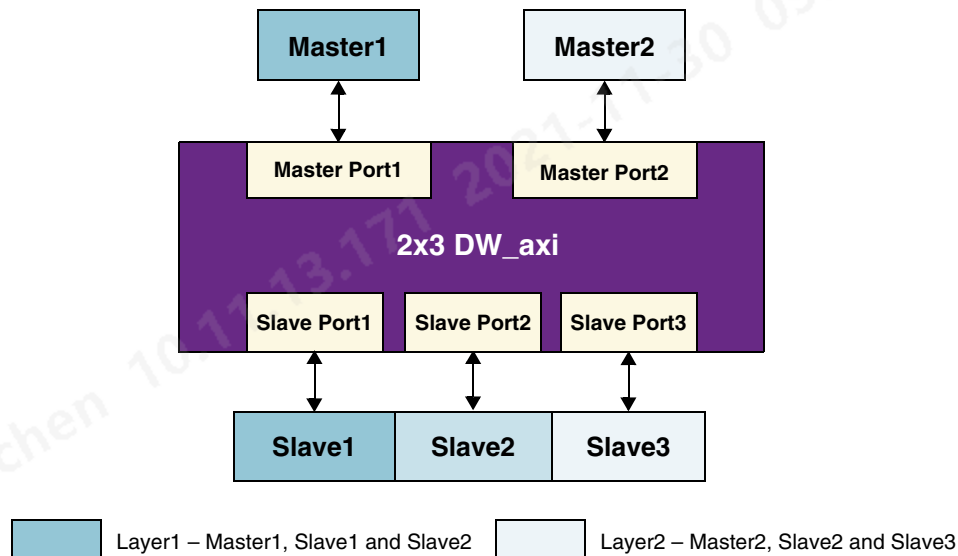
The visibility feature can be used to make slaves visible or invisible to each master. For each mode, Boot or Normal, each slave has a visibility parameter with each master. The logic required to implement each DW_axi layer is reduced for each AXI slave that is made not visible on that layer.

Figure 2-8 shows a two-layer system in Normal Mode. Layer 1 contains Master 1 and Slave 1 and Slave 2. Layer 2 includes Master 2 and Slave 2 and Slave 3. Slave 2 is shared between Layer 1 and Layer 2. Slave 1 is unique to Layer 1, and Slave 3 is unique to Layer 2.

**Note**

The default slave is not shown because it is always visible to all external AXI masters.

Figure 2-8 Two-Layer AXI Subsystem

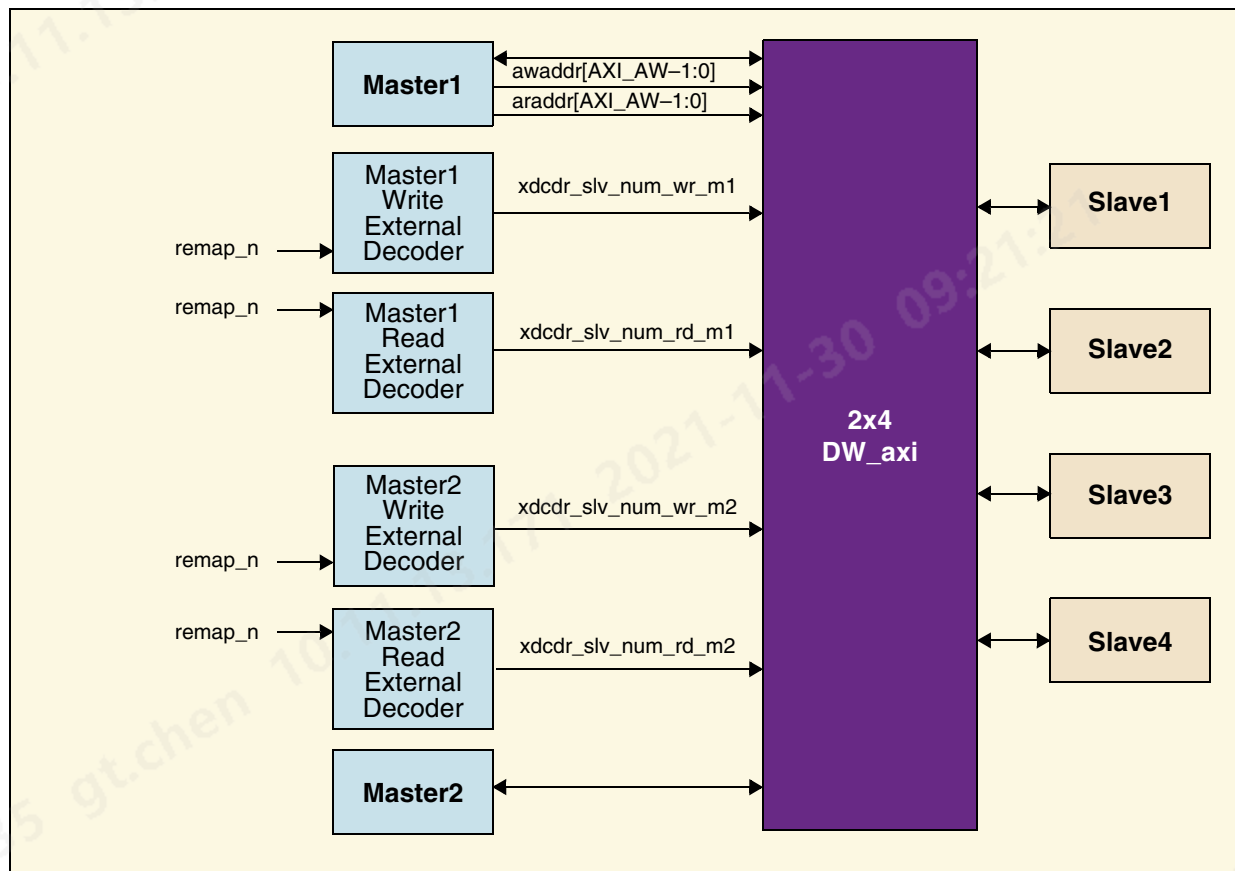


2.6.5 Trustzone Support

This is an optional feature that can be enabled by setting the Trustzone Support (AXI_HAS_TZ_SUPPORT) configuration parameter to True. Each slave port has an input pin (`tz_secure_s(j)`) indicating whether the attached slave is a secure slave. If a non-secure read or write command access is addressed to a secure slave, as indicated by the `awprot/arprot` control signals, then this command is routed to the default slave. For reads, each beat of the burst indicates a DECERR on `rresp`. For writes, the default slave generates a DECERR `bresp` when the master has completed a write transaction.

2.7 External Decoder

DW_axi supports the use of an external decoder for the read and write command channels of each external master as shown in Figure 2-9.

Figure 2-9 Example Use of External Decoders**Note**

There is a single parameter that selects between internal decoders on all read/write address layers or external decoders on all read/write address layers. If you want to use external decoders, set the configuration parameter “Has External Decoders” (AXI_HAS_XDCR) to True. If this parameter is set to False, internal decoders (two for each master port) are implemented in the design.

When you use an external decoder, you must adhere to the following rules:

1. The slave number output of the external decoder must use the same index as that used in DW_axi; that is, the default slave is Slave 0, Slave 1 is attached to DW_axi Slave Port1, Slave2 is attached to DW_axi Slave Port2, and so on.
2. There must be a combinatorial path between the external decoder address control inputs and the slave number output.

Apart from the previous rules, you can implement the decoder with as many remap options as required. Unlike the internal DW_axi decoders, the external decoders do not need the same memory map on all address layers.

When DW_axi is configured for external decoders and to support Trustzone, then DW_axi routes the read/write command to the default slave, regardless of the external decoder output, if the read/write

command is an attempted non-secure access to a secure slave. The external decoders do not need to support Trustzone, since it is handled in the DW_axi.

2.8 Arbitration

This section describes the arbitration function in DW_axi.

2.8.1 Arbitration Scenarios

The following scenarios require the interconnect to use arbitration:

1. Multiple masters try to access the same slave with a read address request at the same time. This arbitration takes place in the slave port and arbitrates for ownership of the addressed slave's read command bus.
2. Multiple masters try to access the same slave with a write address request at the same time. This arbitration takes place in the slave port and arbitrates for ownership of the addressed slave's write command bus.
3. Multiple write data from different masters destined to the same slave port at the same time. The write data in this case is interleaved. This arbitration takes place in the slave port and arbitrates for ownership of the addressed slave's write data bus. The write data channel arbiter only needs to arbitrate between AXI_WID_S(j) (configuration parameter) clients, where AXI_WID_S(j) is the write interleaving depth of Slave Port(j).



Note

In AXI4/ACE-Lite configurations AXI_WID_S(j) is always set to 1.

4. Multiple write responses from different slaves, destined to the same master at the same time. This arbitration takes place in the master port and arbitrates for ownership of the targeted master's write response bus. The default slave is always the lowest priority slave.
5. Multiple read data from different slaves, destined to the same master at the same time. This arbitration takes place in the master port and arbitrates for ownership of the targeted master's read data bus. The default slave is always the lowest priority slave.
6. Multiple channel sources attempt to access a channel sink on the shared layer at the same time.

Therefore, three arbiters are required in each instantiated slave port and two arbiters for each instantiated master port. This gives a total of $(3*S) + (2*M)$ arbiters, where $S = \text{AXI_NUM_SLAVES}$ and $M = \text{AXI_NUM_MASTERS}$.

2.8.2 Arbitration Types

You can use the following parameters to select between arbitration types for each channel for every master and slave port:

- AXI_AR_ARB_TYPE_S(j)
- AXI_AW_ARB_TYPE_S(j)
- AXI_W_ARB_TYPE_S(j)

- AXI_R_ARB_TYPE_M(x)
- AXI_B_ARB_TYPE_M(x)

The available arbitration types are:

- “Priority Arbitration Type” on page 47
- “First-Come First-Served Arbitration Type” on page 48
- “Fair-Among-Equals Arbitration Type” on page 48
- “User-Defined Arbitration Type” on page 48
- “QoS Arbitration Type (Dynamic Priority)” on page 50

If a shared layer is selected for any channel, the arbitration on that shared layer can be configured similarly to any other arbitration point in the design. The following coreConsultant parameters select the arbitration type for each shared layer channel:

- AXI_AR_SHARED_ARB_TYPE
- AXI_AW_SHARED_ARB_TYPE
- AXI_W_SHARED_ARB_TYPE
- AXI_R_SHARED_ARB_TYPE
- AXI_B_SHARED_ARB_TYPE

If AXI_HAS_LOCKING = 1, then there is a restriction that the read and write address channels for each slave must have the same arbitration type; this does not include the default slave.



Note

There can be different arbitration types in the address channels of different slaves, but in each slave port, the arbitration type of the read and write address channels must be identical if AXI_HAS_LOCKING = 1. This is also required on the shared layer; that is, if AXI_HAS_LOCKING = 1, then AXI_AR_SHARED_ARB_TYPE must be equal to AXI_AW_SHARED_ARB_TYPE.

2.8.2.1 Priority Arbitration Type

If the configuration parameter AXI_HAS_EXT_PRIORITY is set to False, then static priorities are configured for every requesting client. The priorities of requesting clients can be statically configured at configuration time. For scenarios 1-3, each master is statically configured with a priority using the configuration parameter AXI_PRIORITY_M(x). Note that a single priority value is assigned for a master that is used for all three arbitration scenarios outlined in scenarios 1-3.

For scenarios 4-5, each slave is statically configured with a priority using the configuration parameter AXI_PRIORITY_Si. Note that a single priority value is assigned for a slave that is used for both arbitration scenarios outlined in 4 and 5.

When two or more clients of the arbiter are requesting, then the highest priority client is granted. In cases where two or more clients of the arbiter have the same priority, the arbiter uses the index of inputs to resolve a tie among the requesting clients. The client connected to the input request[0] has the highest priority, while the client connected to request [n-1] has the lowest priority.

2.8.2.2 First-Come First-Served Arbitration Type

Clients are granted in the order they request; the configured priorities of the clients (masters/slaves) are not used. Instead the arbiter itself internally maintains the priority of the clients. This operates such that the longest waiting client has the highest priority. The priority of the currently-granted client is set to the lowest priority value, and the priorities of all the requesting clients not yet granted are incremented by 1. The priorities are updated each time a transfer is accepted with an assertion of the ready signal for the particular channel.

For requests that are issued in the same cycle, the arbiter uses the index of the client to resolve the arbitration, with the lowest-numbered client winning.

Unlike the Priority arbitration type, this is a fair arbitration scheme where each client is guaranteed to get a share of the available bandwidth. However, there is no concept of an external client priority, so a client has to wait its turn to get granted.

There are more logic levels through the first-come first-serve arbiter than the priority arbiter. As such, from an achievable clock frequency point of view, the first-come first-serve arbiter is slower than the priority arbiter.

2.8.2.3 Fair-Among-Equals Arbitration Type

This is a two-tier arbitration scheme. The first tier is similar to the Priority arbitration option discussed earlier, where the external priority of a client is used to decide the granted client. The higher priority requesting client wins the grant.

The second tier of arbitration shares the grant equally between requesting clients of the same highest priority. The grant is issued fairly among these clients, and the internal priorities are updated every time transfer is accepted with an assertion of ready for the particular channel.

This is a mixture of a fair arbitration scheme and a priority-based arbitration scheme. There is a fair sharing of available bandwidth between clients of the same priority, but the first tier of arbitration allows a higher-priority client to get immediate access.

**Note**

Fair-among-equals does not mean “first-come-first-served among equals”. While a higher priority client is granted, the internal priority of lower-priority clients will increment. These internal priorities can wrap around—depending on how many *VALID and *READY handshakes for which the higher priority client is granted—and result in a request that arrived later being granted before an earlier request of the same priority.

There are more logic levels through the fair-among-equals arbiter than the priority arbiter. As such, from an achievable clock frequency point of view, the fair-among-equals arbiter is slower than the priority arbiter.

2.8.2.4 User-Defined Arbitration Type

When the “User defined Arbitration” option is selected for any channel on any port, a module called DW_axi_arb_user is instantiated in the DW_axi_arb arbiter module. A text file called DW_axi_arb_user.v — located in the *workspace/src* directory — is available for users to edit and implement their own arbitration scheme.

By default, the DW_axi_arb_user.v module instantiates a dynamic priority arbiter; this allows the packaged test environment to run through the coreConsultant GUI. The same arbiter module is used for all channels

where user-defined arbitration is selected. Thus, only one type of user-defined arbitration can be used for all channels of the DW_axi subsystem.

The value of “ N ” that is passed to the user-defined arbiter for the read address channel and write address channel at any slave port is equal to the number of masters visible to that port on that address channel.

For example, if you have sixteen masters, and eight of them are visible to the slave port read address channel, then N for that arbiter is equal to 8.

**Note**

If only one master is visible at this slave port, then $N = 2$. The value of “4” that is seen in the DW_axi_arb_user.v file is the default value of the parameter “ N ”. The actual value is passed to it from the hierarchy above when DW_axi_arb_user is instantiated, and it overrides the default value.

In the case of the write data channel, the value of N depends on the parameters that control the following:

- Whether the channel is Shared or Dedicated
- Write Interleaving Depth of the slave port

If the channel is Shared, then $N = \text{number_visible_masters}$ at that slave port. If the channel is Dedicated, then $N = \text{write_interleaving_depth}$ (WID).

**Note**

If AXI_HAS_LOCKING=1, it is not possible to select user arbiters on the read and write address channels. This is due to custom behavior required of the arbiter to implement locking functionality in the address channels.

The user-defined arbiter is passed the following parameters, which can change within a single instantiation of DW_axi:

- N - Number of clients to the arbiter
- PRIOR_WIDTH - Width of priority input per client
- INDEX_WIDTH - Log base 2 of N

The user-defined arbiter has the following inputs:

- clk_i - system clock
- rst_n_i - asynchronous reset
- enable_i - Asserted to freeze the arbiter to it's current state; applies to timing-based arbiters only
- request_i [$N-1:0$] - request per client
- prior_i [$\text{PRIOR_WIDTH} \times N-1:0$] - Priority value per client

The user-defined arbiter has the following outputs:

- granted_o - asserted when the arbiter is granted to a client
- bus_granted_o [$N-1:0$] - granted bit per client
- grant_index_o [$\text{INDEX_WIDTH}-1:0$] - index of the currently granted client.

For the user-defined arbiter to function correctly within DW_axi, it should follow these restrictions:

- The `granted_o` output should be asserted combinatorially when any request is asserted to the arbiter. Any cycle delay in this breaks the operation of the DW_axi.
- The `bus_granted_o` output should be asserted in the same cycle as `granted_o`. The bit corresponding to the granted client should be asserted, while all other bits should be deasserted.
- For timing-based arbiters, the `enable_i` input freezes the arbiter to its current state. This holds the arbiter to its current state while a valid is asserted, but ready has not yet asserted; additionally, this in turn allows the order of granted clients to more closely follow the arbitration algorithm.

2.8.2.5 QoS Arbitration Type (Dynamic Priority)

If the configuration parameter `AXI_HAS_QOS` is set to `True`, then the QoS arbitration type can be selected on each slave address channel. The priorities of requesting clients can be dynamically selected with the corresponding QoS signal value.

When two or more clients of the arbiter are requesting, then the highest priority client is granted. In case two or more clients of the arbiter have the same priority, the arbiter uses the index of inputs to resolve a tie among the requesting clients. The client connected to the input `request[0]` has the highest priority, while the client connected to `request[n-1]` has the lowest priority.

The QoS Arbiter is configurable only on the write address channel and/or read address channel. Arbiters on data channels can be configured to any type.



Note

Quality of Service (QoS) functionality is source-only; that is, you must have a DWC-AMBA-Fabric-Source license in order to use the QoS features.

2.8.3 Internal versus External Priorities

By default, master and slave priorities are configured using the `AXI_PRIORITY_[M(x)/S(j)]` parameters. You can set the `AXI_HAS_EXT_PRIORITY = 1` to configure DW_axi input ports to drive the master and slave priority values, named `mst_priority_m(x)` and `slv_priority_s(j)`, respectively.



Note

Dynamic priority values—external priorities that are dynamic at whatever level the system is being synthesized—allow less optimization of the logic in the internal arbiters and can result in higher levels of logic through the arbiters and a lower achievable clock frequency.

2.8.3.1 Internal Priorities

If the configuration parameter `AXI_HAS_EXT_PRIORITY` is set to `False (0)`, then static priorities are configured for every requesting client. The priorities of requesting clients can be statically configured at configuration time. For scenarios 1-3, each master is statically configured with a priority using the configuration parameter `AXI_PRIORITY_M(x)`. Note that a single priority value is assigned for a master that is used for all three arbitration scenarios outlined in scenarios 1-3.

For scenarios 4-5, each slave is statically configured with a priority using the configuration parameter `AXI_PRIORITY_S(j)`. Note that a single priority value is assigned for a slave that is used for both arbitration scenarios outlined in 4 and 5.

2.8.3.2 External Priorities

If the configuration parameter AXI_HAS_EXT_PRIORITY is set to True (1), then the priorities of the master and slaves are inputs to DW_axi. The signal inputs used for arbitration are mst_priority_m(x) and slv_priority_s(j) and are described in “Signal Descriptions” on page 181. These inputs are sampled every clock cycle, so arbitration priorities can be adjusted dynamically. This provides maximum flexibility to the IP integrator to create any conceivable arbitration scheme.

When the configuration parameter AXI_HAS_EXT_PRIORITY is set to True, then each master's priority can be driven to any value in the range 0 to $(\text{ceil}(\log_2(\text{AXI_NUM_MASTERS})) - 1)$. So, for example, if the AXI_NUM_MASTERS = 5, then the priority bus width for each master's priority is 3 bits and a master's priority can be driven to any value in the range 0 to 7. This is different to the scenario where the configuration parameter, AXI_HAS_EXT_PRIORITY is set to False and a master's priority is statically configured through coreAssembler or coreConsultant to a value in the range [0, AXI_NUM_MASTERS - 1].

When using dynamic arbitration, each slave's priority can be driven to any value in the range 0 to $(\text{ceil}(\log_2(\text{AXI_NUM_SLAVES} + 1)))$. However, since priority 0 (the lowest priority) is used by the default slave (see “Default Slave” on page 40), it is recommended to not use this slave priority with slaves attached to DW_axi. If slave priority 0 is assigned by dynamic arbitration to any attached slaves, those slaves are considered lower priority than the default slave and incur additional response latency if arbitration clashes occur.



Note

- For the write data channel, the connection to the respective arbiter request lines is not fixed and will depend on history of traffic through the DW_axi. For example, when a new master is allowed to start interleaving write data, it is assigned the request input to the write data channel arbiter that the previous master—which has just finished its write data burst—had been assigned.

The net effect of this is that when two or more masters with the same priority request the same slave's write data channel, then it is difficult to predict the winner of this arbitration, as it is a function of previous traffic through the channel for the particular slave. If predictable arbitration is required, then all slaves and all masters should be given unique priorities.

- The default slave is assigned a priority 0 (lowest priority). For static arbitration, it is guaranteed the default slave will always be the lowest priority, since priority 0 is not allowed to be assigned to any attached slave via static arbitration configuration. For dynamic arbitration, when arbitration values are driven as inputs to DW_axi, it is recommended that priority 0 not be assigned to any attached slave so that the default slave is kept as the lowest priority slave.

2.8.4 Arbitration and Waited Transfers

For all AXI channels, the payload source must ensure that the payload remains stable during a waited transfer. The DW_axi channel arbiter does not change its grant when the payload destination inserts wait states. For an example, if Master 1 is granted Slave 2's read address channel and Slave 2 has deasserted arready_s2, then the read address channel arbitration in Slave Port2 is frozen and Master1 remains granted until the read command is accepted by the external slave, such as until arready_s2 is asserted.

2.9 Atomic Accesses

DW_axi supports both exclusive and locked transactions. The following sections describe how these atomic accesses are handled by the interconnect.

2.9.1 Locked Transfers

To enable DW_axi to support AXI locked sequences, set `AXI_HAS_LOCKING` to 1.

When a master sends a locked transfer to any slave – internally to DW_axi – a locked request is sent to both the read and write address channel arbiters of that slave.

**Note**

Locked access is a feature of only AXI3; it is not supported in AXI4 configurations. Lock functionality is removed in AXI4 and ACE-Lite implementations.

When a locked transfer is granted on either channel, the locked transfer is not sent to the slave until the following conditions have been satisfied:

- There are no outstanding transfers.
- Both address channels have granted the locked request.

It is possible that one address channel grants a locked transfer while the other does not; for example, a master's locked request is granted on the write address channel, but not granted on the read address channel due to a higher-priority master performing a non-locked read transfer.

While this is the case, the channel that is locked waits for the other channel to grant the locked transfer. The channel that has not yet locked masks all new locking requests from other masters, but continues to grant non-locked transfers from other higher-priority masters.

Eventually, the non-locked channel grants the locked request from the master that locked the other address channel. From this point on, arbitration to the slave is locked, and the locking request is sent to the slave once all outstanding transfers have completed.

The following are limitations of DW_axi support for locked sequences:

- It is assumed that an external master will not generate a locking command on one command channel and a simultaneous command that is not locking on the other command channel.
- It is only required to lock a slave's read and write command channels to a master; it is not necessary to lock the slave's write data channel.
- The DW_axi does not support locked transfers that cross slave boundaries. All locked transfers in a locked sequence must address the same slave.

This includes a slave transition to the default slave during a locked sequence. If a master issues an unlocked command that is routed to the default slave, then the interconnect does not release the locking of any command channels that are locked to that master.

- During a locked sequence to Slave(*j*), the `remap_n` and `tz_secure_s(j)` input pins must remain constant.

- The slave port connected to the default slave does not support locking of either the read or write command channels. The locked attributes of an AXI command that is routed to the default slave are ignored.

**Note**

If any of the hybrid architecture functions are used, then locking sequences are not supported; for more details, refer to [“Arbiter Pipeline Stage and Shared Layers”](#) on page 38.

If AXI_HAS_LOCKING = 1, then user arbiters on the read and write address channels are not supported.

If any QoS feature is enabled, then locking sequences are not supported.

2.9.2 Exclusive Accesses

DW_axi makes no special considerations for exclusive access transfers and processes them in the same way as non-exclusive accesses.

2.10 ID Bus

All AXI masters have the same ID widths for all AXI channels, which can be configured by setting the AXI_MIDW configuration parameter. If any master connecting to the DW_axi master port has an ID width less than the configured value for any AXI channel, then the upper most significant bits (MSBs) of the channel's ID bus should be tied to zero.

AXI_SIDW is a read-only parameter for the slave ID width, which is a function of the number of masters and the master ID width. For instance:

$$\text{AXI_SIDW} = \text{AXI_MIDW} + \text{ceil}(\log_2(\text{AXI_NUM_MASTERS}))$$

The AXI_SIDW according to the number of masters is reflected in [Table 2-4](#).

Table 2-4 AXI Slave ID Width

AXI_NUM_MASTERS	AXI_SIDW
1	AXI_MIDW
2	AXI_MIDW + 1
3-4	AXI_MIDW + 2
5-8	AXI_MIDW + 3
9-15	AXI_MIDW + 4

For the read and write address channels and the write data channel, DW_axi appends additional bits to the MSBs of the arid, awid, and wid signals that represent the master number. For read data and write response channels, DW_axi uses the additional bits to identify the target master. DW_axi strips the additional bits from the MSBs before forwarding the ID to the master.

For more information about the ID width parameters for AXI masters and slaves, see [“Parameter Descriptions”](#) on page 103.

2.11 Out-of-Order Deadlock Avoidance

Using the AXI protocol, a deadlock can occur if an interconnect stalls a slave at the slave port in order to comply with ordering rules at the master port. For reads, a deadlock can occur if an interconnect needs to re-order read data from multiple slaves to the same master. For writes, a deadlock can happen if an interconnect needs to re-order a write burst response from multiple slaves to the same master.

DW_axi automatically avoids out-of-order deadlock. Each master is permitted to have active read/write commands to different slaves by applying the following out-of-order deadlock avoidance rules:

- No two active read commands to two different slaves can have the same ARID value. Each master can have multiple active read commands to the same or different slaves if this rule is not violated.
- No two active write commands to two different slaves can have the same AWID value. Each master can have multiple active write commands to the same or different slaves if this rule is not violated.

If DW_axi cannot forward a read/write command — because to do so would violate the out-of-order deadlock avoidance rules — the interconnect stalls the master's read/write address channel until the rule is no longer violated. For example, for a read with a given ARID intended for a given slave, all outstanding read commands with the same ARID to different slaves must complete before the rule no longer would be violated.

A read/write command with a given ARID/AWID value is stalled until all active read/write commands with the same ARID/AWID value and to different slaves have completed.

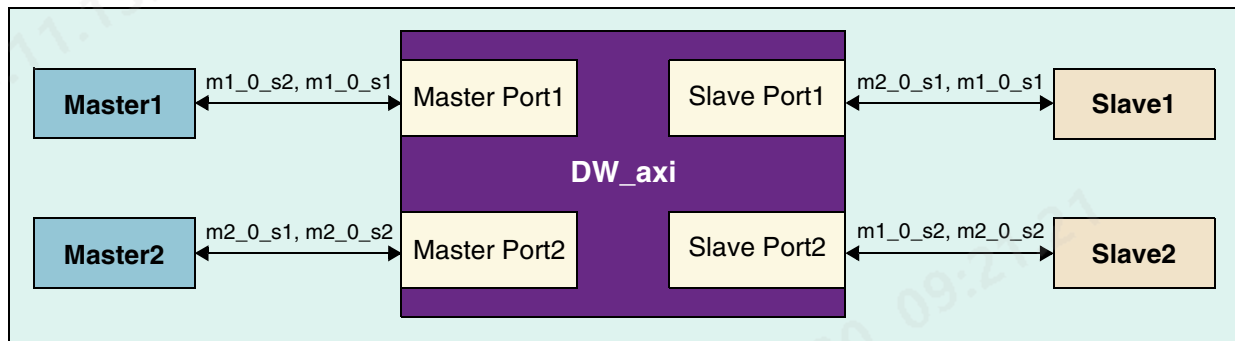
The implementation of these rules requires hardware in DW_axi to track the targeted slave numbers and ARID values of all active read commands. In order to limit the tracking hardware, the maximum number of read commands that Master Port(x) can accept for a given ARID value is limited by the configuration parameter, AXI_MAX_RCA_ID_M(x). Also, the maximum number of unique values of ARID that Master Port(x) can accept for read commands at any one time is limited by the configuration parameter, AXI_MAX_URIDA_M(x).

Similar configuration parameters are defined for the write address channel. For more information about these configuration parameters, see [“Parameter Descriptions”](#) on page 103.

For more deadlock information in bidirectional configurations, refer to [“Multitile Deadlock Prevention”](#) on page 67.

2.11.1 Example of Out-of-Order Deadlock

[Figure 2-10](#) shows a command sequence from Master 1 and Master 2 that would result in a deadlock situation if the DW_axi did not implement out-of-order deadlock avoidance rules.

Figure 2-10 Example of Out-Of-Order Deadlock

Note The $m(x)_X_s(j)$ notation is a command from Master(x) to Slave(j) with ARID equal to X .

Master 1 sends out two commands: the first to Slave 1 with ARID equal to 0; the second to Slave 2 with ARID equal to 0. Because both read commands have the same ARID value, the AXI protocol requires read data returned in the same order as Master 1 issued the read commands (specifically, read data from Slave 1, then read data from Slave 2).

Master 2 sends out two commands: the first to Slave 2 with ARID equal to 0; the second to Slave 1 with ARID equal to 0. Because both read commands have the same ARID value, the AXI protocol requires read data returned in the same order as Master 2 issued the read commands (specifically, read data from Slave 2, then read data from Slave 1).

The following scenario explains how deadlock can occur if the out-of-order deadlock avoidance rules are not followed:

- Both Slave 1 and Slave 2 have read re-order depths equal to two.
- Slave 1 returns data to Master 2. However, the AXI protocol requires data from Slave 2 first, so DW_axi stalls Slave 1's read data channel until all read data from Slave 2 is returned.
- Slave 2 returns data to Master 1. However, the AXI protocol requires data from Slave 1 first, so DW_axi stalls Slave 2's read data channel until all read data from Slave 1 is returned. Therefore, deadlock occurs.
- Slave 1's read data channel is stalled waiting for Slave 2 to return read data to Master 2, and Slave 2's read data channel is stalled waiting for Slave 1 to rerun read data to Master 1.
- Both read data channels are waiting for the other to complete.

The previous example explains a second-order deadlock. Higher-order deadlock can occur when Slave 1's read data channel is waiting for Slave 2's read data channel to complete; when Slave 2's read data channel is waiting for Slave 3's read data channel to complete; and so on, up to slave n 's read data channel waiting for Slave 1's read data channel to complete.

The previous scenario talks of a deadlock scenario that can occur when multiple slaves return read data out of order. For writes, the same deadlock can occur when multiple slaves return write responses out of order.

2.12 Read Data/Write Response Re-ordering

Because the out-of-order deadlock avoidance rules do not permit active read or write commands to different slaves with the same ARID or AWID value, respectively, DW_axi never needs to re-order read data or write responses from multiple slaves to the same master.

2.13 Write Data Interleaving

DW_axi allows write data streams from different masters to be interleaved to the same slave. This functionality is supported only if the slave is configured to have its own dedicated write data channel layer. Slaves that connect to only the shared write data channel are limited to a write interleaving depth of 1. The depth of the write interleaving is equal to the write interleave depth of the slave port, which is a static configuration parameter (AXI_WID_S(j)).

For more details on write interleaving with shared layers, refer to [“Write Data Interleaving”](#) on page 38.

**Note**

Write data interleaving is a feature of AXI3 only; it is not supported in AXI4 configurations. Write data interleaving functionality is removed in AXI4/ACE-Lite implementations.

The write interleave depth of a slave port must be less than or equal to the write interleave depth of the attached slave. For optimum performance, the write interleave depth of a slave port should be configured to be equal to the write interleave depth of the attached slave. Setting the write interleave depth of a slave port to a value greater than that supported by an attached slave is illegal and results in undefined behavior.

DW_axi does not support the interleaving of write data from the same master to the same or different slaves when the number of masters is configured to be greater than 1 (AXI_MASTER_NUM > 1). This means that the write interleave depth of all master ports is hardcoded to 1 when the number of masters is configured to be greater than 1.

**Note**

The shared write data channel does not support interleaving depths greater than 1 (AXI_WID_S(j) > 1) from masters in both single- and multiple-master configurations. For more details, refer to [“Write Data Interleaving”](#) on page 56.

[Figure 2-11](#) shows an example where the external slave has a write interleaving depth of 2. The attached slave port is configured to have a matching write interleaving depth of 2. Write data from Master 1 and Master 2 can be interleaved by DW_axi and forwarded to the slave.

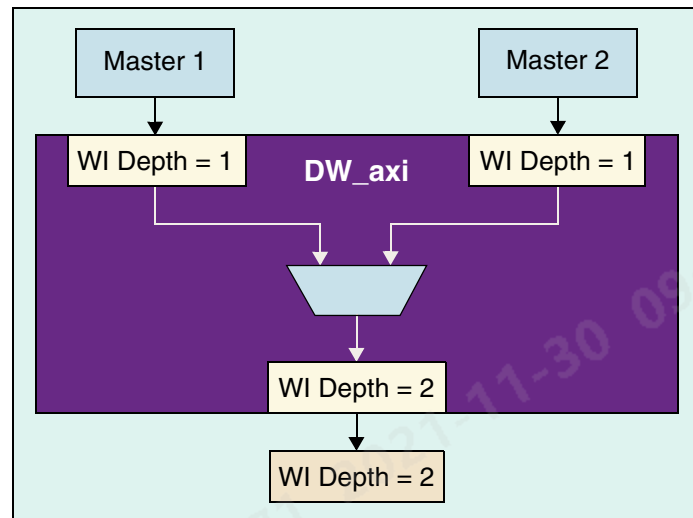
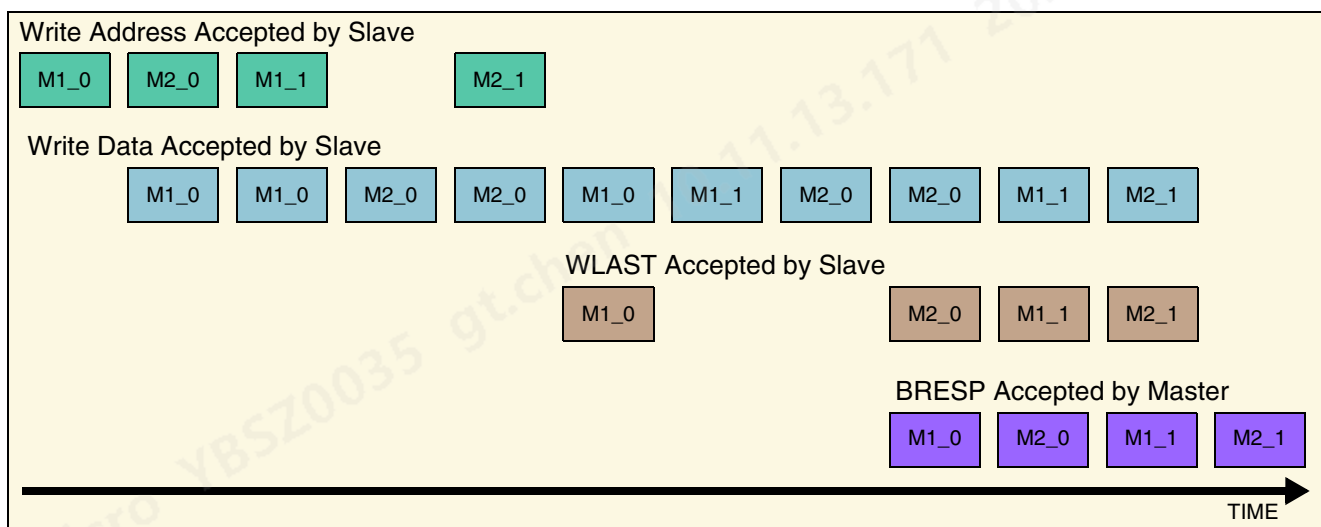
Figure 2-11 Write Data Interleaving Example

Figure 2-12 shows an example transfer where two external masters target the same external slave where the external slave has a write data interleaving depth of 2. For the write address channel, M1_0 signifies a clock cycle where the external slave accepts a write command from Master 1 with AWID 0. M1_1 represents a clock cycle where the external slave accepts a write command from Master 1 with AWID 1. The slave receives the write commands in the following order:

The slave receives the write commands in the following order:

1. Write command from Master 1 with AWID 0 and burst length equal to 3 followed by
2. Write command from Master 2 with AWID 0 and burst length equal to 4 followed by
3. Write command from Master 1 with AWID 1 and burst length equal to 2 followed by
4. Write command from Master 2 with AWID 1 and burst length equal to 1.

Figure 2-12 Write Interleaving Example

The “Write Data Accepted by Slave” layer in Figure 2-12 signifies a clock cycle where the external slave accepts write data from an external master. The “WLAST Accepted by Slave” layer in the figure represents a clock cycle where the external slave accepts the last beat of write data from an external master. Finally, the “BRESP Accepted by Master” layer signifies a clock cycle where the external master accepts the write response from an external slave. For all of these layers, the $M(x)_y$ notation is used, where x is the master number and y is the transaction ID.

For active write commands, DW_axi can start to interleave write data associated with the next active write command once the last beat of write data from a previous active write command has been accepted by an external slave. It is not required to wait for a previous active command to complete, which occurs when the external master accepts the write response from the external slave.

In this example, M1_0 must complete all write data before M1_1 can start sending write data (DW_axi master ports in this example have a write interleave depth of 1, since there are multiple masters attached). Similarly, M2_0 must complete all write data before M2_1 can start sending write data. However, notice in the write data accepted by the slave, that write data interleaving to a depth of 2 occurs in the following order:

1. Write data from Master 1, ID 0, is interleaved with write data from Master 2, ID 0.
2. Write data from Master 1, ID 1, is interleaved with write data from Master 2, ID 0.

DW_axi ensures that the following AXI protocol write ordering rules are adhered to:

1. When DW_axi combines write transactions from different masters to one slave, the interconnect must ensure that the slave receives the first beat of write data for each transaction in the same order in which the transaction addresses were received.
2. When DW_axi combines write transactions from different masters, it must do so under the constraints of write interleave depth supported by the slave port.

DW_axi stalls a master's write data channel if forwarding the master's write data onto the slave's write data channel violates the ordering rules specified previously.

2.13.1 Stalling of Write Data Channel and Write Data Interleaving

This section highlights a scenario where the write data channel of multiple masters is stalled when they target the same slave even when that slave has a write interleaving depth that should allow the interleaving of write data from the multiple master sources. This occurs because of the following conditions:

1. The interconnect must ensure that the slave receives the first beat of write data for each transaction in the same order in which the slave received the write commands addresses.
2. When the interconnect accepts multiple write commands from the same master and forwards them to the slave, as each master has a write interleave depth of 1, it can only supply the first beat of write data for the second command after the first command has completed in its entirety, regardless of the value of AWID.

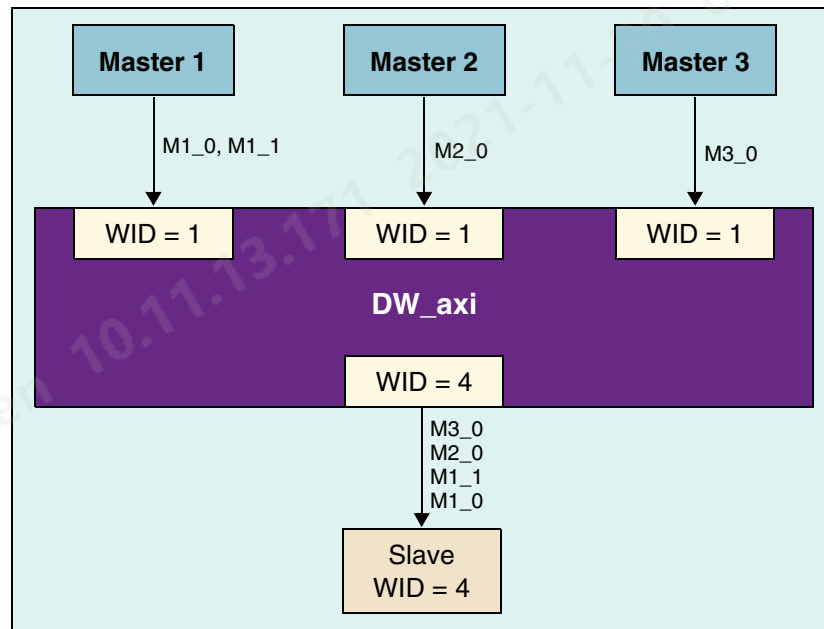
Therefore, if two consecutive commands from the same master (Master x) are forwarded to the slave followed by write commands from different masters, DW_axi must first wait for the first beat of write data from the second command from Master(x) to be propagated to the external slave before the write data of any of the other masters can be forwarded. However, this can only occur after the first write data burst from Master(x) has completed in its entirety, such as when the last beat has completed.

In this scenario, while waiting for the assertion of the `wlast` signal corresponding to the first command from Master(x), there can be no interleaving of write data from the other masters even if the slave supports it.

This situation can occur when any master port is configured to accept more than one active write command regardless of AWID.

Figure 2-13 shows a scenario where the write data channels of Master 2 and Master 3 are stalled even though the slave supports a write data interleaving depth of 4. The $M(x)_y$ notation represents a write command from Master(x) with ID(y).

Figure 2-13 Example: Stalling of Write Data Channel



The example illustrated in Figure 2-13 shows a command stream to an external slave with back-to-back write commands from Master 1. As Master 1 has a write interleave depth of 1, it can only supply the first beat of write data for the second command after the first write burst has completed in its entirety, regardless of AWID. This effectively stalls the write data channels of Master 2 and Master 3 until the first active write command forwarded to the slave from Master 1 completes its write data phase, for example, `wlast` is asserted. This occurs because the commands from Master 2 and Master 3 are forwarded to the slave after the second active write command from Master 1. Therefore, the first beat of write data from these commands can only be forwarded to the external slave after the first beat of write data from the second active command from Master 1.

If you configure **DW_axi** so that for all master ports ($x = 1$ to $x \leq \text{AXI_NUM_MASTERS}$) there can never be more than one active write command from the same external master generated by any **DW_axi** slave port, then the previous scenario can never happen. This occurs when:

1. The maximum number of write commands that an external Master(x) can generate for a particular AWID value before **DW_axi** stalls the master's write address channel is 1, where the configuration parameters `AXI_MAX_WCA_ID_M(x) = 1`.

2. The maximum number of write commands that an external Master(x) can generate with unique AWID values before DW_axi stalls the master's write address channel is 1, where configuration parameters $AXI_MAX_URIDA_M(x) = 1$.

2.13.2 Write Data Interleaving and Single Master AXI System

When the number of masters is configured to equal 1 ($AXI_NUM_MASTERS = 1$), make note of the following rules:

- DW_axi allows the external master to interleave write data up to a depth equal to the maximum number of active unique write IDs configured at the master port ($AXI_MAX_UWIDA_M1$).
- The external master must ensure that it does not interleave write data to a depth that is not supported by an attached slave. This is an AXI protocol requirement. If this protocol rule is violated, the behavior of the system is undefined.



Note

DW_axi does not explicitly prevent a violation of this rule; it is a requirement of the external master not to violate this rule.

2.14 Read Data Interleaving

By default, the DW_axi allows read data to be interleaved to a master to a depth equal to the number of outstanding unique IDs configured for that master port – $AXI_MAX_URIDA_M(x)$.

For masters that do not support read data interleaving but that do initiate multiple outstanding transactions with different ID values, the DW_axi master port can be configured to enforce a read interleaving depth of 1.

By setting the parameter $AXI_RI_LIMIT_M(x) = 1$, DW_axi limits read interleaving to that master to a depth of 1, which means that no read interleaving will occur to that master.

To avoid a bus deadlock condition when using this setting, a slave that is sending read data to a master with $AXI_RI_LIMIT_M(x) = 1$ must not interleave read data to another master that has $AXI_RI_LIMIT_M(x) = 1$. The safest way to guarantee this is for slaves returning read data to a master that has $AXI_RI_LIMIT_M(x) = 1$ to never interleave read data.



Note

Read data interleave limiting is not supported by the shared read data channel. Because of this, masters that do not have a dedicated read data channel cannot use the read data interleave-limiting functionality.

For the read interleaving limiting feature to function correctly, and also to avoid a bus deadlock condition, a slave that is sending data to a master with $AXI_RI_LIMIT_M(x) = 1$ must not interleave read data to that master or to other masters with $AXI_RI_LIMIT_M(x) = 1$.

The safest way to guarantee this is for slaves to never interleave read data if they return read data to a master with $AXI_RI_LIMIT_M(x) = 1$.

2.15 Early Write Data

Early write data in the AXI is when write data for a transaction is sent by a master before the corresponding command for the write data has been issued.

The DW_axi supports masters that issue early write data. Depending on the pipelining mode selected, one or two beats of early write data may be accepted by the DW_axi. After this point, the write data channel stalls until the master issues the relevant command. Write data cannot be routed until it has been determined to which slave the data is going, which introduces a dependency between the write address channel and the write data channel for the master port.

This restriction does not exist at the slave port in the DW_axi. However, because of write interleaving rules, the DW_axi never issues early write data for a transaction. The DW_axi can send write data for a transaction if the command is still waiting to be accepted on the write address channel at the slave port, but this is the earliest time the write data is sent; any earlier risks violating the first-beat address order rule.

2.16 Clocks and Resets

The DW_axi uses the AXI interface clock and reset signals `aclk` and `aresetn`.

When the DW_axi is configured with QoS feature (`AXI_HAS_QOS=1`), APB interface clock and corresponding reset `pclk` and `presetn` are also present in the design.

Both the resets `aresetn` and `presetn` must be asserted at the same. One reset must not be asserted while the other is de-asserted. It is assumed that both the reset inputs are asserted and de-asserted at the same time with de-assertion synchronous to the corresponding clocks. De-assertion may not be exactly at the same time as there may be delays associated with respect to the synchronization to each clock domain.



Note

DW_axi does not support asserting of one reset while the other is de-asserted; doing so results in unpredictable behavior.

2.17 Stalling of the Payload Source

This section provides conditions that cause payload sources to be stalled.

2.17.1 Read Address Channel / Write Address Channel

DW_axi inserts WAIT states on an attached master's read/write address channel if any of the following conditions occur:

- The read/write address channels have limits for the following:
 - Maximum number of unique values of ARID/AWID that a DW_axi master port can accept for active read/write commands at any one time; defined by `AXI_MAX_URIDA_M(x)`/`AXI_MAX_UWIDA_M(x)`
 - Maximum number of active read/write commands that a master port can accept for a given ARID/AWID value; defined by `AXI_MAX_RCA_ID_M(x)`/`AXI_MAX_WCA_ID_M(x)`

If either of these two maximum limits has been reached, then the master's read/write address channel is stalled until an active command completes and the limiting maximum condition becomes false.
- If the read/write command's ARID/AWID value matches the ARID/AWID value of an active read/write command but to a different slave, then the master's read/write address channel is stalled. DW_axi stalls in this instance because of the out-of-order deadlock prevention rule, which is described in more detail in [“Out-of-Order Deadlock Avoidance”](#) on page 54.

- Arbitration can cause stalls when multiple masters try to access a slave's read/write address channel simultaneously.
- Once the master has been granted the slave's read/write address channel, then the slave's `arready_s(j)`/`awready_s(j)` is propagated to the master's `arready_m(x)`/`awready_m(x)`.

If the slave cannot accept the read/write command, then the deasserted `arready_s(j)`/`awready_s(j)` from the slave is propagated to the master's `arready_m(x)`/`awready_m(x)` and the command stalls. DW_axi does not buffer the read/write command when the slave is not ready to accept it.

2.17.2 Write Data Channel

DW_axi inserts WAIT states on an attached master's write data channel if any of the following conditions occur:

- DW_axi must align the write data channel with the write address channel. A DW_axi master port stalls the write data channel of a connected master if it receives valid write data with a WID that does not match a write command, with matching AWID, that has been previously accepted by the DW_axi master port.

When a DW_axi master port receives a write command, the AWID and slave number are stored in a lookup table. When the master generates valid write data, the WID of the write data channel is used as an index to the lookup table. If a match between WID and a stored AWID is found, then the write data is routed to the corresponding slave. If not, the write data needs to be aligned to the write address channel, and the write data channel is stalled.

In AXI4/ACE-Lite implementations, the WID dependency is removed; in this case, the incoming AWID sequence is stored in a buffer. When the master generates valid write data, then the write data is routed to the corresponding slave targeted by the first pending AWID in the buffer.



Note

When `awvalid_m(x)` and `wvalid_m(x)` are asserted on the same cycle with the same WID, then the write data cannot be immediately accepted and `wready_m(x)` is de-asserted on that clock cycle.

- The DW_axi slave port must comply with the write data interleaving rules stated in [“Write Data Interleaving”](#) on page 56. When the slave port cannot forward write data to an attached slave because to do so would violate the write data interleaving rules, then the master's write data channel is stalled until the offending condition becomes false. Once both the write data and the matching write command are aligned and when the write data interleaving rules are adhered to, then the master arbitrates for the targeted slave's write data bus.
- Arbitration can cause stalls when multiple masters try to access a slave's write data channel at the same time.
- Once the master has been granted the slave's write data channel, then the slave's `wready_s(j)` is propagated to the master, `wready_m(x)`.

If the slave cannot accept the write data, then the slave's deasserted `wready_s(j)` is propagated to the master's `wready_m(x)` and the write data stalls. DW_axi does not buffer the write data when the slave is not ready to accept it.

2.17.3 Read Data Channel/Write Response Channel

DW_axi inserts WAIT states on an attached slave's read data/write response channel, if any of the following situations occur:

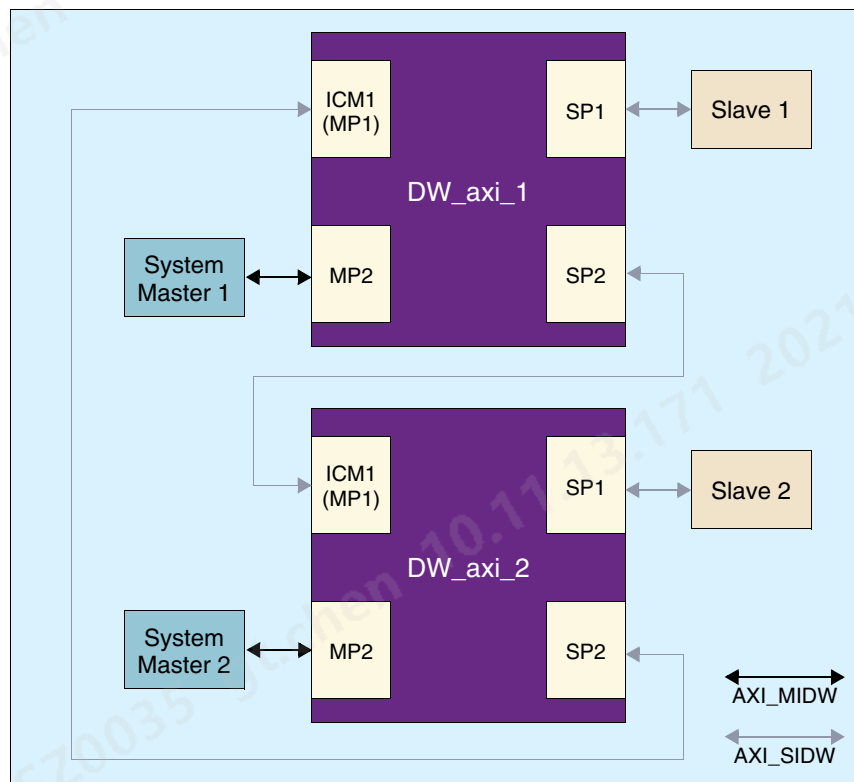
- Arbitration can cause stalls when multiple slaves try to access the master's read data/write response channel simultaneously.
- Once the master has been granted the slave's read data/write response channel, then the master's ready_m(x)/bready_m(x) is propagated to the slave's ready_s(j)/bready_s(j).

If the master cannot accept the read data/write response, then the master's deasserted ready_m(x)/bready_m(x) is propagated to the slave's ready_s(j)/bready_s(j) and the read data/write response stalls. DW_axi does not buffer the read data/write response when the master is not ready to accept it.

2.18 Bidirectional Command Support

In order to allow bidirectional transactions across multiple cascaded interconnects, you must enable the bidirectional AXI_HAS_BICMD parameter in the configuration. Figure 2-14 illustrates cascaded DW_axi interconnects with system masters.

Figure 2-14 Cascaded DW_axi Interconnects with System Masters



**Note**

When connecting two instances of the DW_axi in a tiled fashion, you should not connect a DW_axi slave port with a write interleaving depth greater than 1 ($AXI_WID_S(j) > 1$) to the master port of another DW_axi, since this breaks the requirement that a master must not interleave write data to the DW_axi unless there is only one master on the DW_axi instance.

2.18.1 System Masters

Master ports on any connected DW_axi interconnect that are directly connected to a master – rather than to an interconnecting master (ICM) port – are referred to as “system masters” when bidirectional command support is enabled.

You must use the `AXI_NUM_SYS_MASTERS` parameter to set the total number of system masters across all interconnects. Up to 64 system masters are supported. All system masters must have the same `AXI_MIDW` ID width, which effectively increases the `AXI_SIDW` ID width across the entire system to:

$$\text{ceil}(\log_2 (AXI_NUM_SYS_MASTERS)) + AXI_MIDW$$

You can then assign to each direct master port a unique system number using the `AXI_SYS_NUM_FOR_M(x)` parameter. For example, in order to assign a second master port to a third System Master, you would set `AXI_SYS_NUM_FOR_M2` to 3. This causes the ID bits received by a slave to indicate that the transaction came from System Master 3; that is, when `AXI_HAS_BICMD` is enabled, the system master is appended to the ID, rather than to the master port number, as is normally the case.

2.18.2 Interconnecting Masters

You can use the `AXI_NUM_ICM` parameter to configure multiple master ports on a given interconnect as either a system master port or an interconnecting master (ICM) port.

**Note**

The lowest numbered master ports are configured as ICM ports. For example, if `AXI_NUM_ICM` is set to 2 in a six-master DW_axi configuration, then Master Port1 and Master Port2 are ICM ports. You must connect only interconnecting masters to an interconnecting master port; that is, the slave port of another DW_axi instantiation.

An ICM port is used to connect to a slave port on another interconnect, so its ID width is defined by `AXI_SIDW` rather than `AXI_MIDW`, which is used by a system master. ICM ports have no system numbers because they either forward or receive the ID from the slave port or interconnect to which they are connected; they do not append any ID bits to transactions that pass through them.

If there is only one ICM on a given interconnect, there is no restriction on the number of system masters on other interconnects that can pass transactions through it. However, if there are multiple connecting masters on an interconnect:

- The number of system masters that can pass transactions through a given connecting master is limited to 8 and is set via the `AXI_NUM_MST_THRU_ICM(x)` parameter. For example, if three system masters numbered 10, 12, and 51 require access through ICM1, the following parameter settings are used:
 - `AXI_ALLOW_MST1_ICM1` 10
 - `AXI_ALLOW_MST2_ICM1` 12

□ AXI_ALLOW_MST3_ICM1 51

- The system masters that are allowed to pass transactions through an ICM are set at configuration time via the AXI_ALLOW_MSTj_ICM(x) parameters.
- A system master can pass transactions through only one ICM per interconnect.

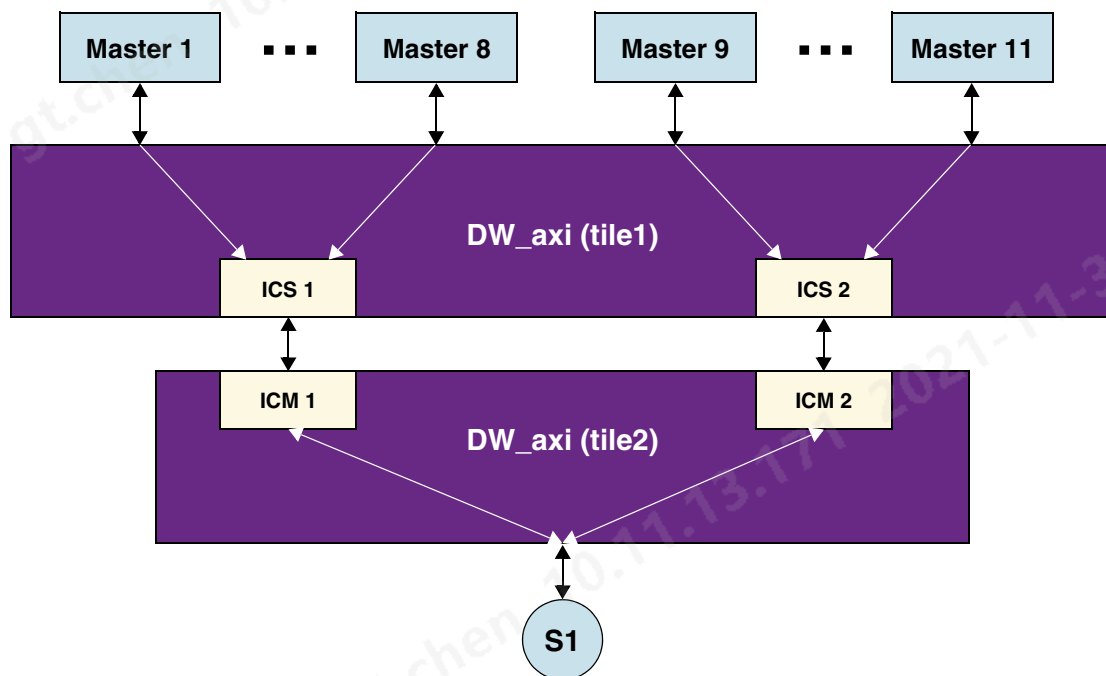


Note

When connecting multiple DW_axi components, it is recommended that system masters not configured to be allowed to pass through an ICM port should be made invisible to the connecting slave port, described in “[Slave Visibility](#)” on page 43. This prevents unconfigured master ID values being passed through the ICM port, which is not supported by the DW_axi.

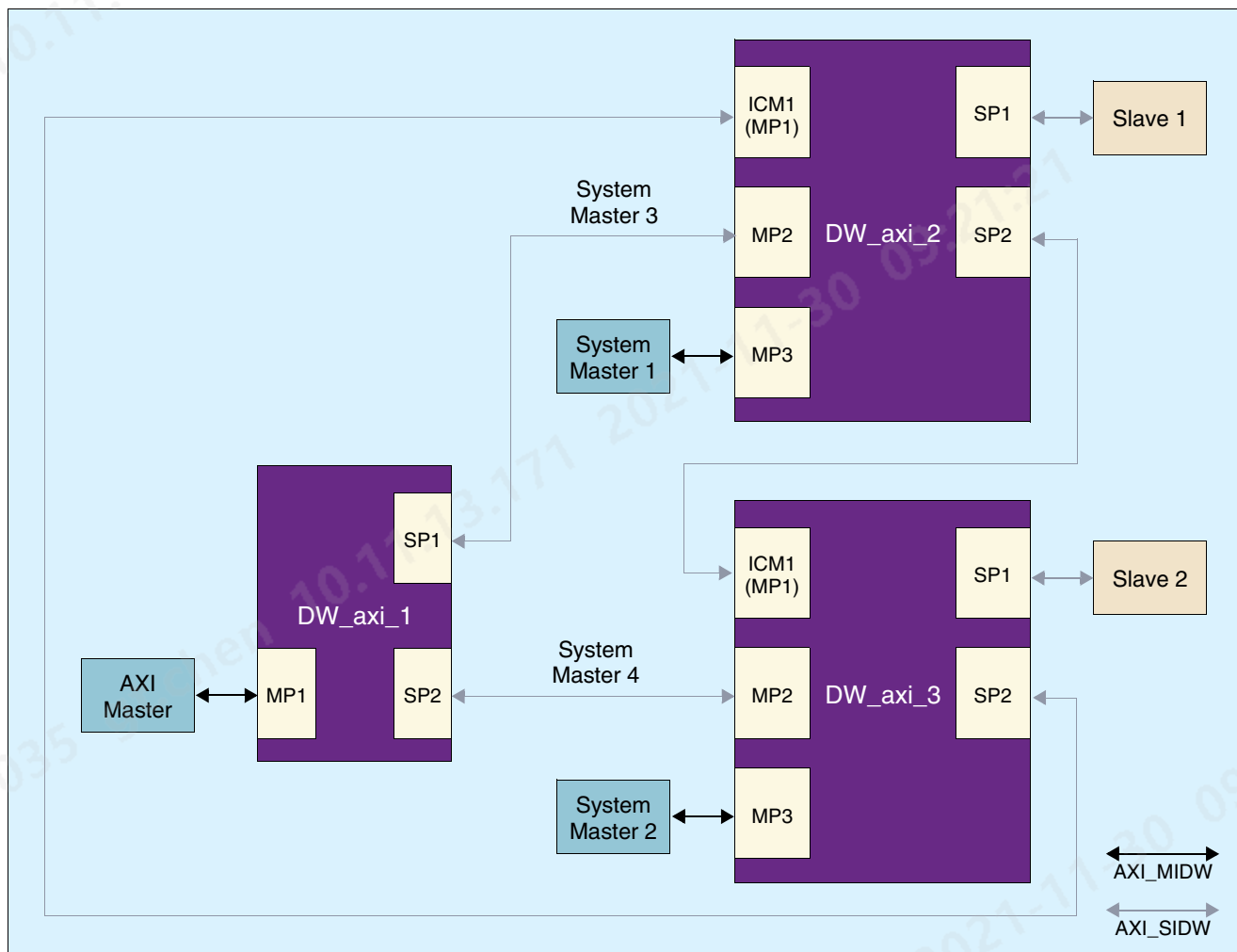
If there is more than one ICM connecting two DW_axi tiles, and there are more than eight masters on one tile that need to access the same slave on the other tile, overlapping slave addresses can be used. In [Figure 2-15](#) there are eleven masters on tile1 that need to access S1 on tile2. Because there are two ICMs on tile2, a maximum of eight masters can pass through either ICM. By configuring the memory ranges of both tile1.ICS1 and tile1.ICS2 to include tile2.S1, Masters 1 to 8 on tile1 can access tile2.S1 through tile1.ICS1 → tile2.ICM1, and Masters 9 to 11 can access tile2.S1 through tile1.ICS2 → tile2.ICM2.

Figure 2-15 Bi-Directional Overlapping Slave Address (tile2 to tile1 Connection Not Shown)



Configuration of the DW_axi does not allow you to enable the bidirectional command flow (AXI_HAS_BICMD = 1) when the number of interconnecting masters (AXI_NUM_ICM) is set to 0. This type of configuration would create an RTL design identical to a DW_axi configuration with AXI_HAS_BICMD=0. However, such a configuration could form part of a tiled interconnect system. This is illustrated in [Figure 2-16](#) where DW_axi_1 has the following:

- One master port connected to one system master
- Two slave ports, one of which is connected to DW_axi_2 and one connected to DW_axi_3

Figure 2-16 Cascaded DW_axi Interconnects with Interconnecting Masters

For the above system, DW_axi_1 should be configured with:

- AXI_HAS_BICMD = 0
- AXI_MIDW independent of multi-interconnect considerations

If DW_axi_2 and DW_axi_3 are then configured with AXI_HAS_BICMD = 1, the following occurs when calculating the number of system masters (AXI_NUM_SYS_MASTERS):

- Two slave ports on DW_axi_1 that connect to other interconnects are considered as system masters, instead of the “real” master on DW_axi_1
- ID width of these slave ports is used when setting AXI_MIDW of DW_axi_2 and DW_axi_3

If you use coreAssembler to configure a system in this way, you may get the following warning:

"Some AXI components in your subsystem have bi-directional command support enabled (AXI_HAS_BICMD==1) and some do not. In most cases this is an error that should be corrected"

In this case, you can safely ignore the warning.

2.19 Avoiding Combinatorial Loops Between Tiles

When two DW_axi tiles are connected in a bi-directional multi-tile manner, you should take care to avoid combinatorial loops between the two tiles.

Referring to [Figure 2-14](#), if the following conditions are true, then a combinatorial loop will exist between DW_axi_1 and DW_axi_2.

- No internal pipelining on both DW_axi_1 and DW_axi_2, on any channel.
- DW_axi_1.ICM1 and DW_axi_1.SP2 are visible in either normal or boot (if AXI_REMAP_EN = 0) address modes.
- DW_axi_2.ICM1 and DW_axi_2.SP2 are visible in either normal or boot (if AXI_REMAP_EN = 0) address modes.

To avoid this, and to eliminate redundant logic, the master port (ICM) and slave port on a DW_axi tile that connect to the same external DW_axi tile, should be made non-visible to each other in both normal and boot (if AXI_REMAP_EN = 0) address modes.

For example, in the system shown in [Figure 2-14](#), the following parameter settings should be made:

- DW_axi_1
 - AXI_NV_S2_BY_M1 = 0
 - AXI_BV_S2_BY_M1 = 0 (if AXI_REMAP_EN == 0)
- DW_axi_2
 - AXI_NV_S2_BY_M1 = 0
 - AXI_BV_S2_BY_M1 = 0 (if AXI_REMAP_EN == 0)

2.20 Multitile Deadlock Prevention

There can be a deadlock condition on the write data channel in DW_axi systems with either of the following:

- Bidirectional multitile systems with two or more DW_axi instances
Masters on the DW_axi tiles access both slaves on the tile local to the masters as well as slaves on other tiles across either internal or external pipelines.
- Multitile (non-bidirectional) systems with three or more DW_axi instances

This deadlock condition will manifest if either of these conditions is true:

- Internal pipelining—AXI_AW_TMO, AXI_AW_PL_ARB—has been used in the write address channel of two or more of the DW_axi instances.
- External pipelining—for example, a register slice between the DW_axi instances—has been used in the write address channel of two or more of the DW_axi instances.

These deadlock scenarios are explained in greater detail in [Appendix A.1, “Multitile Deadlock Scenarios”](#).

The DW_axi has a feature that prevents deadlocks from occurring. When configuring the DW_axi, you must configure which slave ports of the DW_axi instance are used to access slaves that are not local to the DW_axi

instance currently being configured. For example, you must configure slave ports that are connected – possibly through other components such as a DW_axi_rs, DW_axi_x2x, and so on – to the interconnecting master port of another DW_axi instance.

The DW_axi uses this information to prevent a master from issuing transactions to a local slave or a different non-local slave if that master has outstanding transactions at a non-local slave. Once the last write data beat of the outstanding transactions at the non-local slave have been issued by the master port, transactions to a different slave – local or non-local – are immediately issued. This situation adds no throughput penalty to the write data channel.

The AXI_ACC_NON_LCL_SLV_S(j) parameters denote which slave ports access non-local slaves. These slave ports are called “interconnecting slave ports” because they are connected to another bus instead of a system slave. Setting the parameter to true (1) indicates that the slave is used to access non-local slaves.

The AXI_ACC_NON_LCL_SLV_S(j) parameters are enabled in multi-tile/bidirectional configurations where AXI_HAS_BICMD = 1. The AXI_ACC_NON_LCL_SLV_S(j) parameters are also enabled in multi-tile configurations where AXI_EN_MULTI_TILE_DLOCK_AVOID = 1.

In the coreConsultant GUI, the AXI_ACC_NON_LCL_SLV_S(j) parameters are accessed through the “Interconnecting Slave Ports” portion of the Multi-Tile/Bidirectional Command section of the Specify Configuration activity; for information on this section, refer to “Multitile/Bidirectional Command” in [“Parameter Descriptions”](#) on page 103. For information on avoiding out-of-order deadlocks, refer to [“Out-of-Order Deadlock Avoidance”](#) on page 54.

2.21 Deadlock Notification

While functions have been built into the DW_axi to prevent the onset of out-of-order or multi-tile deadlock, it is still possible to have a partial or complete deadlock due to events beyond the control of the DW_axi. Examples of possible causes for deadlock are:

- Incorrect power-down of a slave or master
- Internal deadlock on a slave or master

This type of deadlock condition can be detected by monitoring the non-completion of outstanding transactions. If any outstanding transaction does not complete within a predetermined time interval, a deadlock notification is issued. Setting the AXI_DLOCK_NOTIFY_EN parameter to 1 enables this feature and adds the deadlock notification pins to the I/O interface.

The period of time over which the deadlock condition is verified is defined by the AXI_DLOCK_TIMEOUT parameter. On the first clock cycle of each deadlock detection period – which has the duration of AXI_DLOCK_TIMEOUT+1 clock cycles – every ID is evaluated to determine if outstanding transactions exist. A deadlock condition is then detected on an ID if none of its outstanding transactions completes before the end of the same timeout period.

Deadlock debug information is provided for one master at a time. It is possible that deadlock occurs simultaneously on more than one master and/or on more than one ID at a master. In these cases, debug information for subsequent masters or IDs will be provided as the deadlock being reported is resolved. In deciding which deadlock has its debug information reported, the following rules apply:

- If multiple transactions from a master time out within the same timeout period, it cannot be determined which timed-out transaction will have its ID and target slave number routed to the deadlock output ports.

- If multiple masters time out within the same timeout period, the lowest numbered master will have its debug information routed to the top-level signals.
- If there are timed-out read and write transactions from a master during the same timeout period, the debug details of the write transaction will be routed to the top-level signals.
- Once `dlock_irq` asserts, it will remain asserted until all timed-out transactions have completed or until the DW_axi interconnect is reset.

**Note**

Deadlock detection is a two-step process:

1. Identifying IDs with outstanding transactions, which takes place at the beginning of each detection cycle.
2. Checking that none of the outstanding transactions for that ID completed, which happens at the end of the detection cycle.

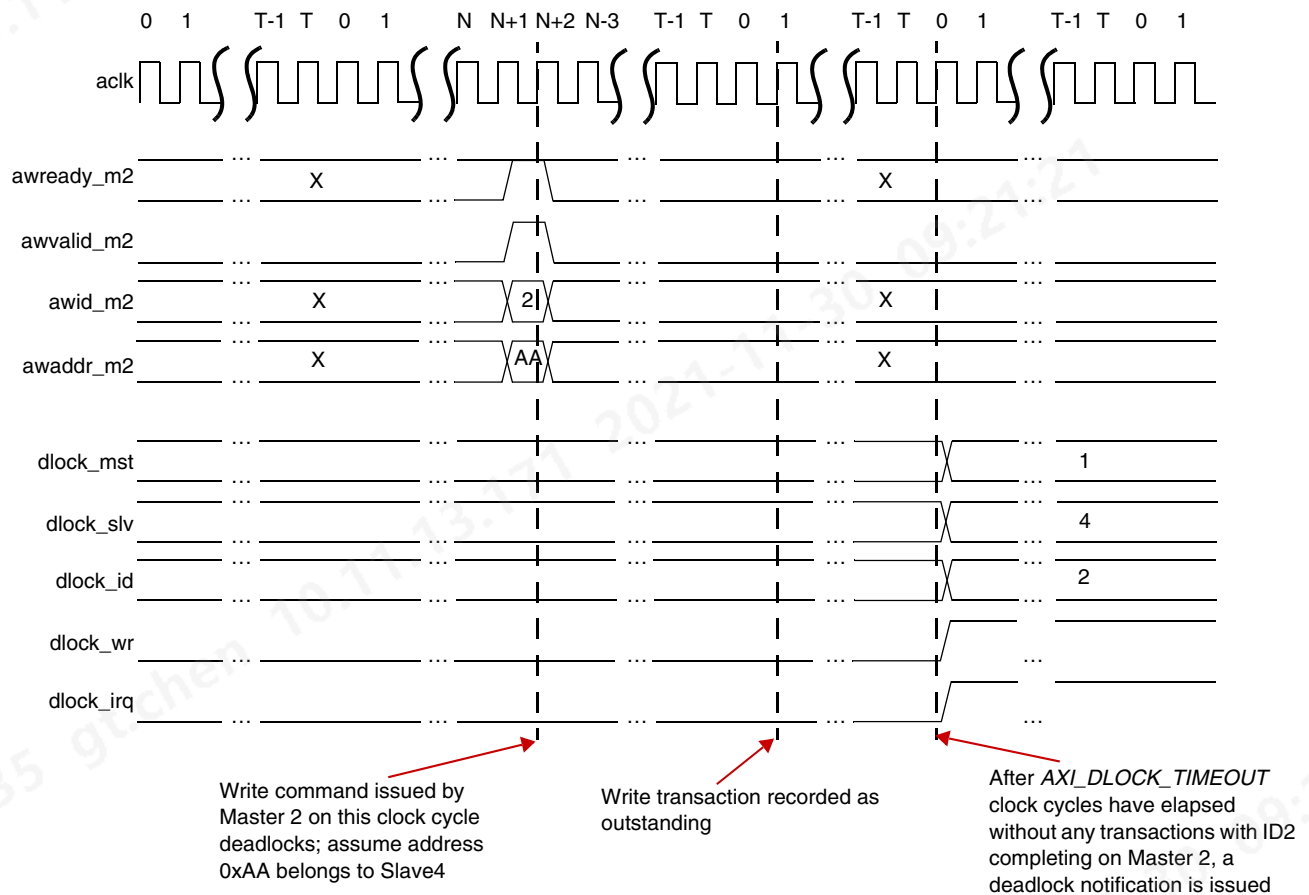
Therefore, the actual maximum time that can elapse before a deadlocked transaction is reported is $2 \times AXI_DLOCK_TIMEOUT + 1$.

For example, a transaction might be initiated in the clock cycle immediately after the start of the detection period on an ID that had no previous outstanding transactions. If this transaction deadlocks, it takes a delay of `AXI_DLOCK_TIMEOUT` before an outstanding transaction is detected on this ID at the beginning of the next detection period, and another delay of `AXI_DLOCK_TIMEOUT + 1` before it is detected that the transaction has not completed, at which point a deadlock notification is issued.

The timing diagram in [Figure 2-17](#) illustrates the concepts described above.

Figure 2-17 Timeout Detection and Deadlock Notification

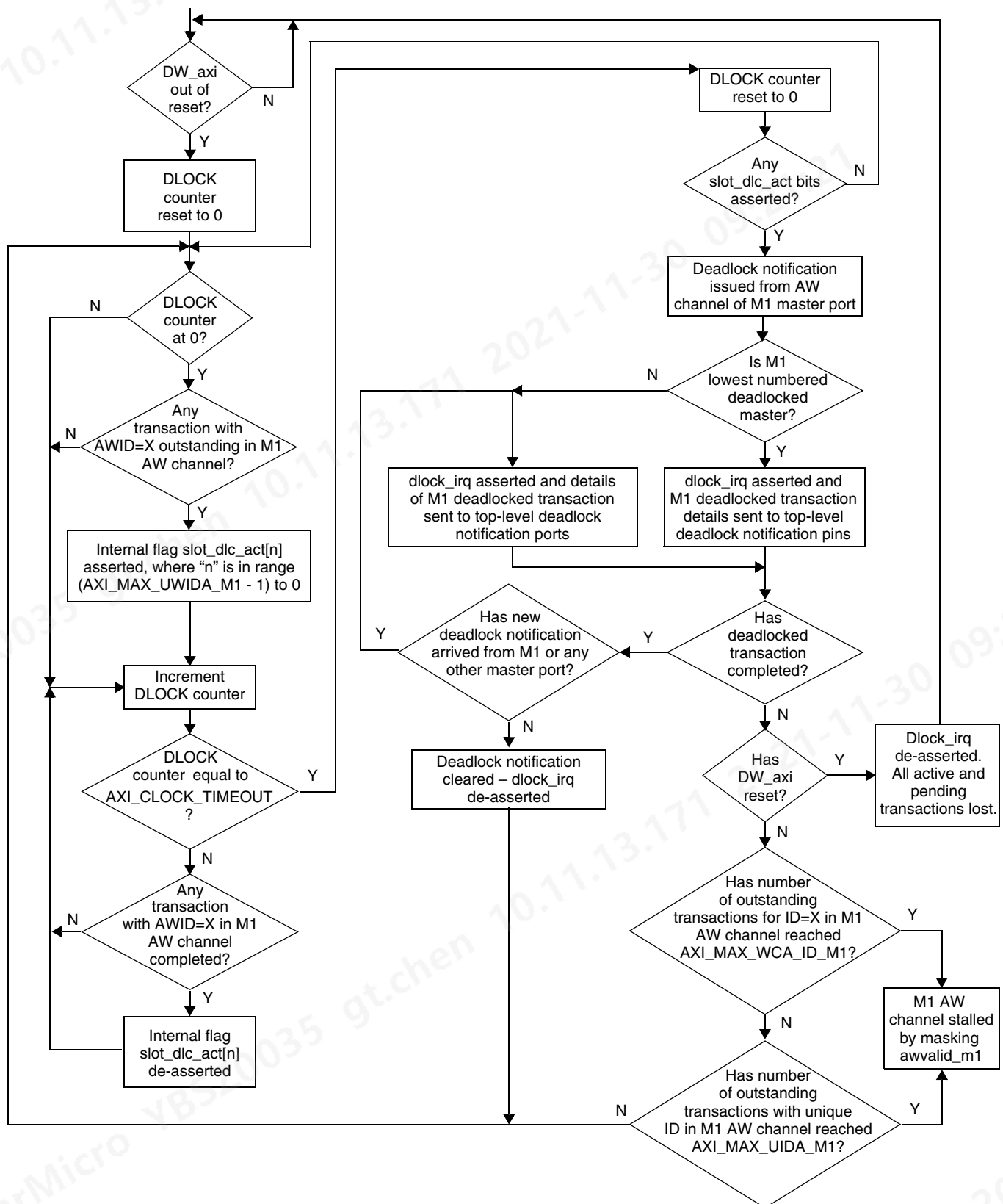
Timeout Detection

**Note**

If a deadlock occurs in a DW_axi configured for bidirectional operation, the local (not system) master and slave numbers are reported. Thus, in a bidirectional multitile system where deadlock notification is required, all DW_axi instances should have this feature enabled for full debug capability.

The flow diagram in [Figure 2-18](#) provides a complete description of how the deadlock notification logic functions in the DW_axi. It uses the AW channel of the M1 master port as an example. The same logical flow applies to the AR and AW channels of all master ports of the DW_axi.

Figure 2-18 Functional Flowchart for Deadlock Notification



2.22 Timing Mode Options

There are three timing mode options for DW_axi—Combinatorial, Forward Registered, and Fully Registered. The configuration parameters AXI*_TMO are used to independently select the timing mode option for each of the five AXI channels. Each channel is assigned its own timing mode.

2.22.1 Avoiding Feedback Loops

The AXI master or slave connected to the DW_axi must adhere to the AXI Protocol Specification requirement, which requires no timing paths from inputs to outputs on masters and slaves. There are two groups of paths that must be broken in order to achieve this.

- For read/write address channels and write data channels in *only* combinatorial mode:
 - The master must break the timing path between valid_src and ready_src:
 - arvalid_m(x) and arready_m(x)
 - awvalid_m(x) and awready_m(x)
 - wvalid_m(x) and wready_m(x)
- For the write response and read data channels in *only* combinatorial mode:
 - The slave must break the timing path between valid_src and ready_src.
 - rvalid_s(j) and rready_s(j)
 - bvalid_s(j) and bready_s(j)

2.22.2 Combinatorial

Figure 2-19 shows the timing paths in a single DW_axi channel when the AXI*_TMO parameter for that channel has been configured with the combinatorial option.

The following combinatorial paths exist through DW_axi.

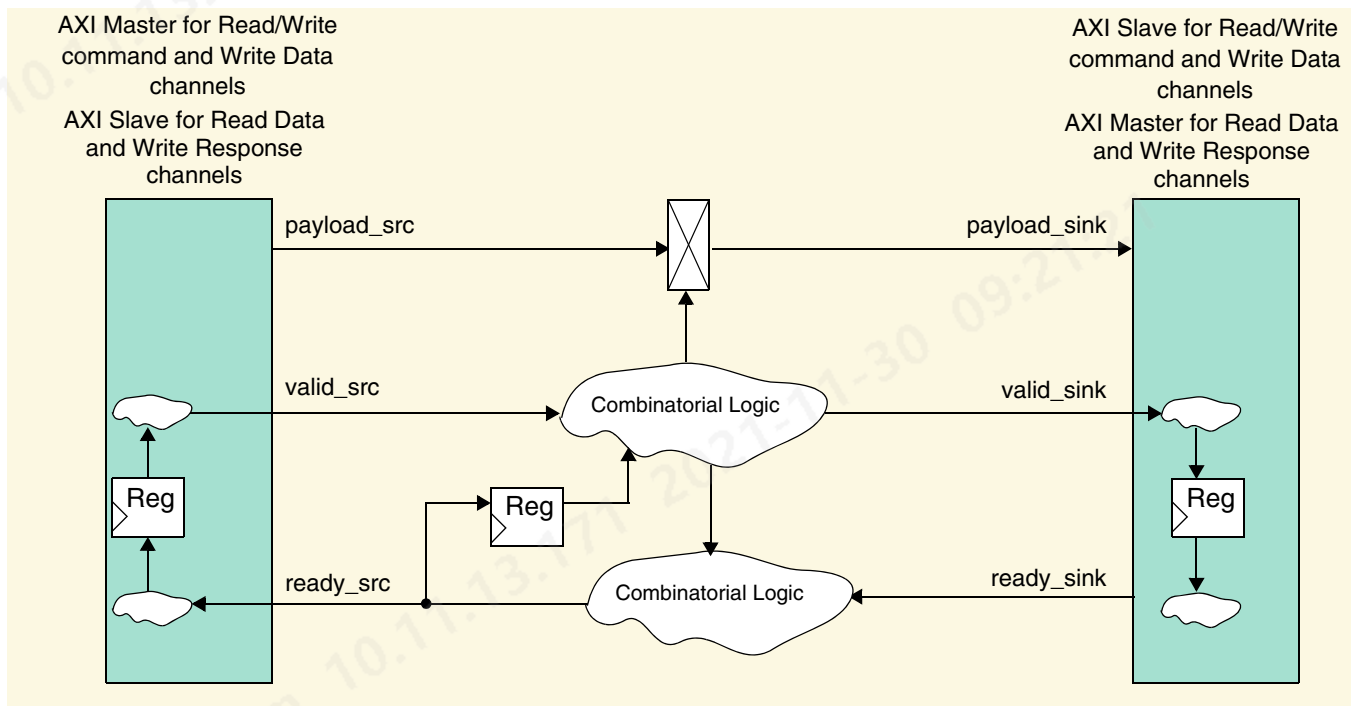
1. From payload_src to payload_sink
2. From valid_src and payload_src to ready_src
3. From valid_src and payload_src to valid_sink
4. From ready_sink to ready_src



Note

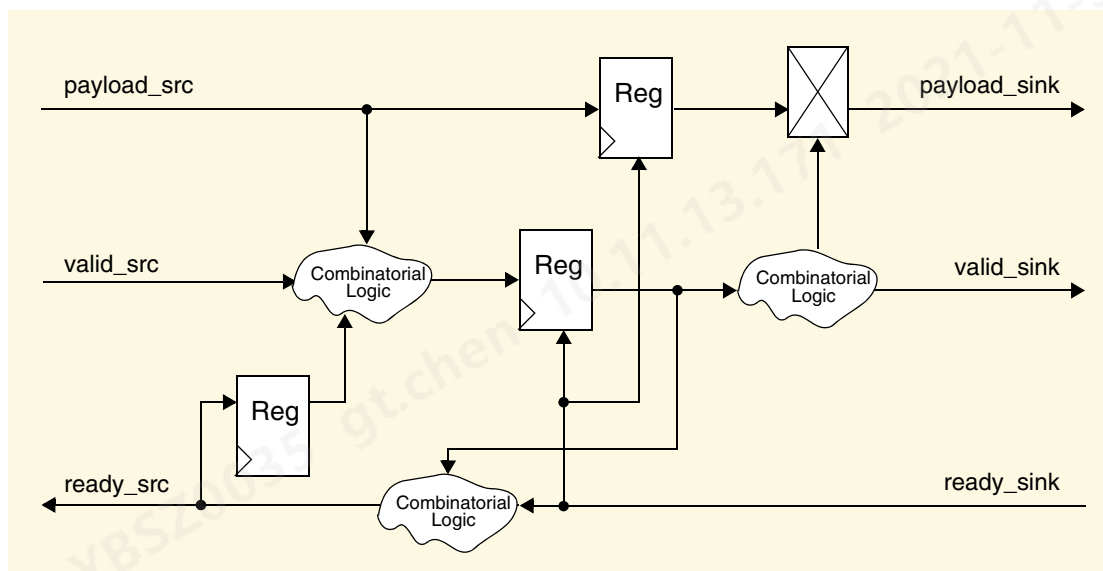
An AXI master is the payload source for the write and read address and write data channels, and the payload destination for the write response and read data AXI channels. An AXI slave is the payload destination for the write and read address and write data channels, and the payload source for the write response and read data AXI channels. For more details about payload sources, refer to Table 1-2.

Compared to the other registered timing modes, Combinatorial timing mode has the lowest area, but the longest critical path, because the AXI channel is not pipelined.

Figure 2-19 DW_axi Timing Paths for Combinatorial Timing Mode

2.22.3 Forward Registered

Figure 2-20 shows the timing paths in a single DW_axi channel when the AXI_*_TMO parameter for that channel has been configured with the Forward Registered option.

Figure 2-20 DW_axi Timing Paths for Forward Registered Timing Mode

The combinatorial paths that exist through DW_axi for this timing mode option are:

- From ready_sink to ready_src for:

- ❑ Write address channel
- ❑ Read address channel
- ❑ Write data channel
- ❑ Read data channel
- ❑ Write response channel

This option results in shorter timing paths compared to the Combinatorial option, but it has an area penalty. The AXI channel payloads of all payload sources are registered, breaking the forward timing path. For example, if the AXI_R_TMO configuration parameter is set to Forward Registered, then the read data of all AXI slaves is registered in DW_axi.

The write data or address channel operations are as follows:

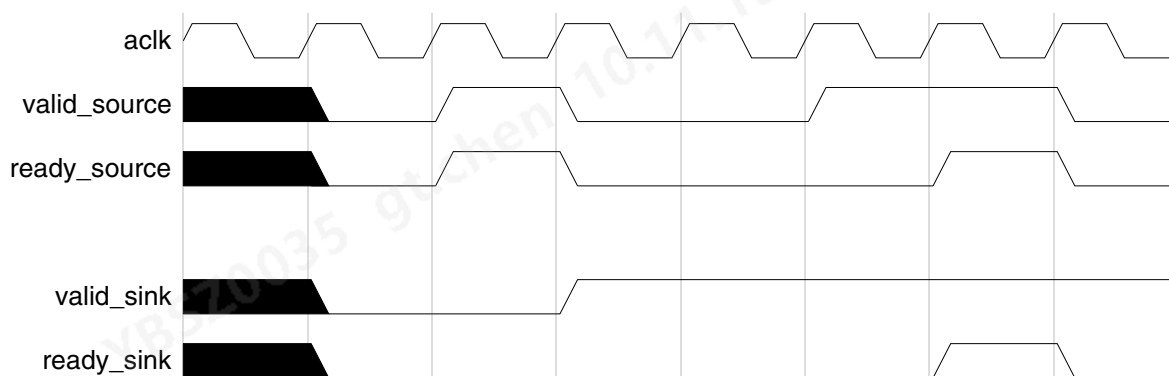
- If there is no transfer on the slave channel, then the DW_axi stores the payload from the master and returns the master ready signal high.
- If there is a transfer on the slave channel, then the DW_axi de-asserts the master ready signal if the slave ready signal is low or if the master is not granted the AXI slave channel. If the slave ready signal is high and the master is granted the AXI slave channel, then the DW_axi stores the AXI payload from the master and returns the master ready signal high.

The read data and write response operations are as follows:

- If there is no transfer on the master channel, then the DW_axi stores the payload from the slave and returns the slave ready signal high.
- If there is a transfer on the master channel, then the DW_axi de-asserts the slave ready signal if the master ready signal is low or if the slave is not granted the AXI master channel. If the master ready signal is high and the slave is granted the AXI master channel, then the DW_axi stores the AXI payload from the slave and returns the slave ready signal high.

In this Forward Registered timing mode, a buffer stage is needed when the DW_axi cannot forward a transfer onto a channel to ensure no throughput penalty. The throughput penalty is described in [Figure 2-21](#), which shows an example timing diagram of the Forward Registered timing mode option.

Figure 2-21 Example Timing Diagram of Forward Registered Timing Mode



Forward Registered timing mode has one payload buffer. By default, DW_axi is ready to accept one transfer; therefore, `ready_source` asserts combinatorially on assertion of `valid_source`. In [Figure 2-21](#), the

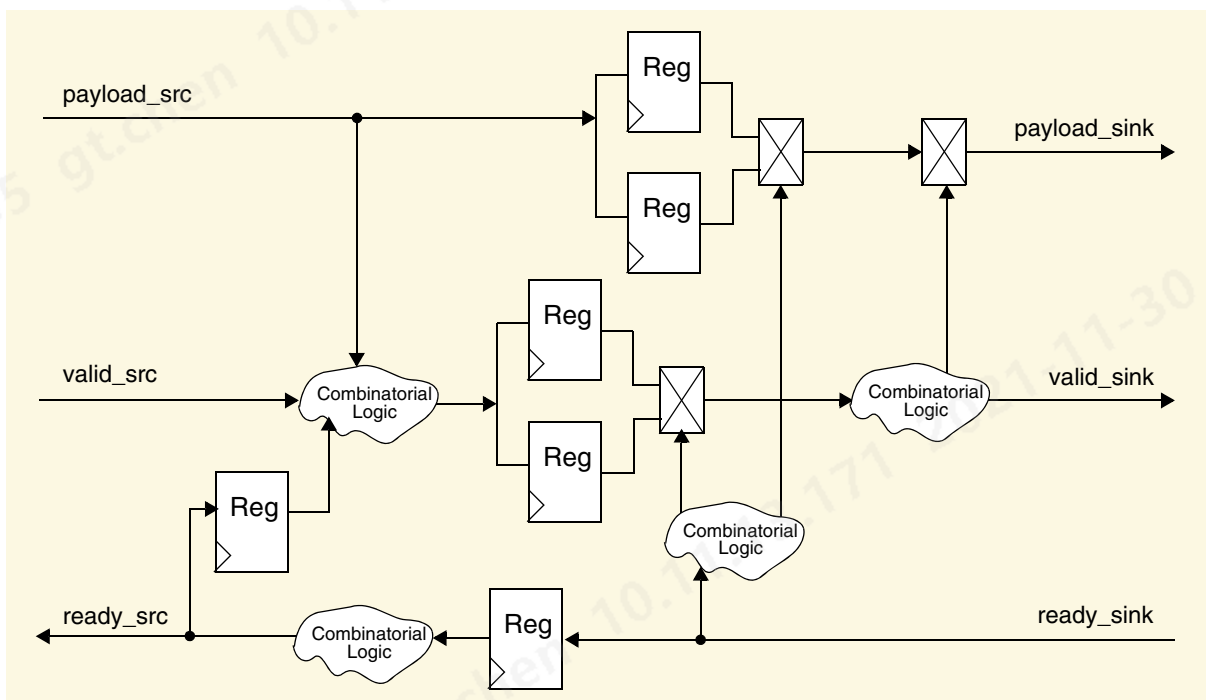
diagram shows that when the source asserts the valid signal, it is not until the next cycle that this valid assertion reaches the destination. In this example, the destination is not immediately ready to accept this payload so it keeps `ready_dest` deasserted. The payload buffer is not transferred out to the destination in this cycle, due to the deasserted `ready_dest`, so the `ready_source` is forced low combinationally from `ready_dest`. The single payload is now holding the first transfer, so `ready_source` stays low until this is accepted by the destination. Some time later, the source issues another valid but DW_axi cannot accept it until the destination asserts ready and the first transfer is transferred out from the payload buffer.

In the timing diagram, it is worth noting that when `ready_dest` is asserted, the `ready_source` is combinationaly asserted in the same cycle, because this backward control path is not registered in the Forward Registered timing mode. This allows the single payload buffer to pipeline transfers based on the value of `ready_dest`—transferring in the next valid transfer from the source in the same cycle that the current payload buffer is transferred out to the destination.

2.22.4 Fully Registered

Figure 2-22 shows the timing paths in a single DW_axi channel when the `AXI*_TMO` parameter for that channel has been configured with the Fully Registered option.

Figure 2-22 DW_axi Timing Paths for Fully Registered Timing Mode



There are no combinational paths through the DW_axi on any channels that are configured with fully registered pipeline mode.

This option has an area penalty over the Forward Registered option because two banks of registers are required for the payload of all payload sources. The two banks of payload registers are necessary to avoid a throughput penalty. The payload registers are used to effectively create a two-deep pipeline to store the AXI channel payload, which is described in the following example.

**Note**

An AXI channel should only be configured to the fully registered timing mode option if the backward control path, from `ready_sink` to `ready_source`, is the critical timing path in the subsystem. If the backward control path is not the critical timing path, the additional gates required for the Fully Registered timing mode option over the Forward Registered timing mode option have no positive effect on performance. For a detailed definition of *backward control path*, refer to “[DW_axi Terminology](#)” on page 24.

The write data or address channel operations are as follows:

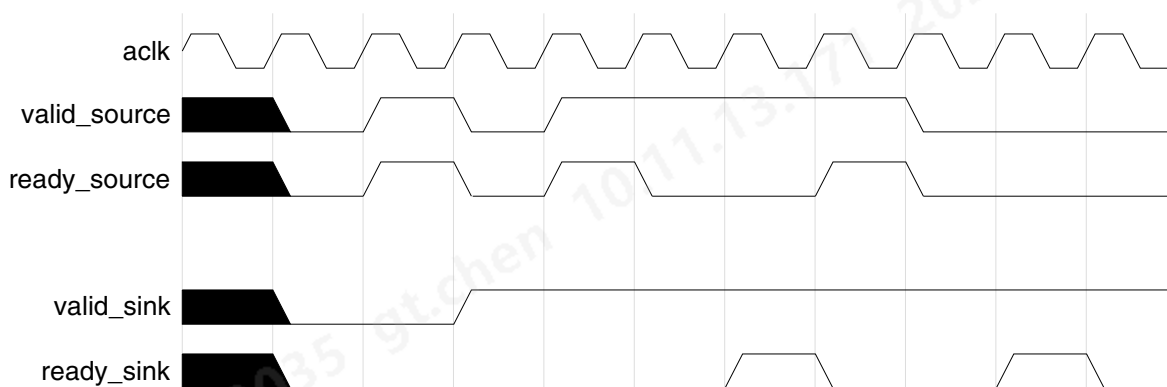
- If one of the two buffer stage payload banks is free, then the DW_axi stores the AXI payload from the master and returns the master ready signal high.
- If both buffer stage payload banks have stored payloads, then the DW_axi stalls the master ready signal if the slave ready signal is low, or if the master is not granted the AXI slave channel. If the slave ready signal is high and the master is granted the AXI slave channel, then the DW_axi stores the AXI payload from the master and returns the master ready signal high. Transfers are issued to the slave in the same order as received by the DW_axi.

The read data and write response operations are as follows:

- If one of the two buffer stage payload banks is free, then the DW_axi stores the AXI payload from the slave and returns the slave ready signal high.
- If both buffer stage payload banks have stored payloads, then the DW_axi stalls the slave ready signal if the master ready signal is low, or if the slave is not granted the AXI master channel. If the master ready signal is high and the slave is granted the AXI master channel, then the DW_axi stores the AXI payload from the slave and returns the slave ready signal high. Transfers are issued to the master in the same order as received by the DW_axi.

Figure 2-23 shows the timing diagram of the Fully Registered timing mode option.

Figure 2-23 Example Timing Diagram of Fully Registered Timing Mode



In the Fully Registered timing mode, there are two payload buffers. By default, DW_axi is ready to accept two transfers; therefore, `ready_source` asserts combinatorially on assertion of `valid_source`. In the example in Figure 2-23, the source drives three transfers before the destination asserts `ready`, so the first two transfers occupy the two payload buffers while DW_axi waits for the source to accept the first transfer. On the cycle after the destination asserts `ready`, `ready` is asserted on the source side. This illustrates how the two payload

buffers can become full. In this example, DW_axi accepts the third transfer while the second transfer is driven out to the destination.

In the Fully Registered timing mode, just like in Forward Registered, there is one cycle of latency added on the payload and valid signal paths. Additionally, there is one cycle of latency on the ready signal path (backward control path). This can be seen in [Figure 2-23](#), where ready_sink assertion results in ready_source assertion one cycle later. The ready_source signal is also asserted anytime there is an available payload buffer.

2.22.5 Pipeline Channel Arbiter

The Pipeline Channel Arbiter parameters – AXI*_PL_ARB – provide another choice in pipelining each channel of the DW_axi. Unlike the internal register slice inferred by the AXI*_TMO parameters, which pipeline between the master and slave port blocks, you can select the AXI*_PL_ARB parameters to add another pipelining stage after the channel arbiter.

When selected, this pipeline acts as another forward register-mode-only internal register slice; that is, the timing path has one buffer stage to ensure no throughput penalty. As such, if it is implemented on a channel with no other registered timing options selected, that channel has the same combinatorial and registered paths as detailed in the “[Forward Registered](#)” on page 73. If the channel already has the Fully Registered mode selected, then the combinatorial/registered path attributes of the Fully Registered mode apply to the channel.

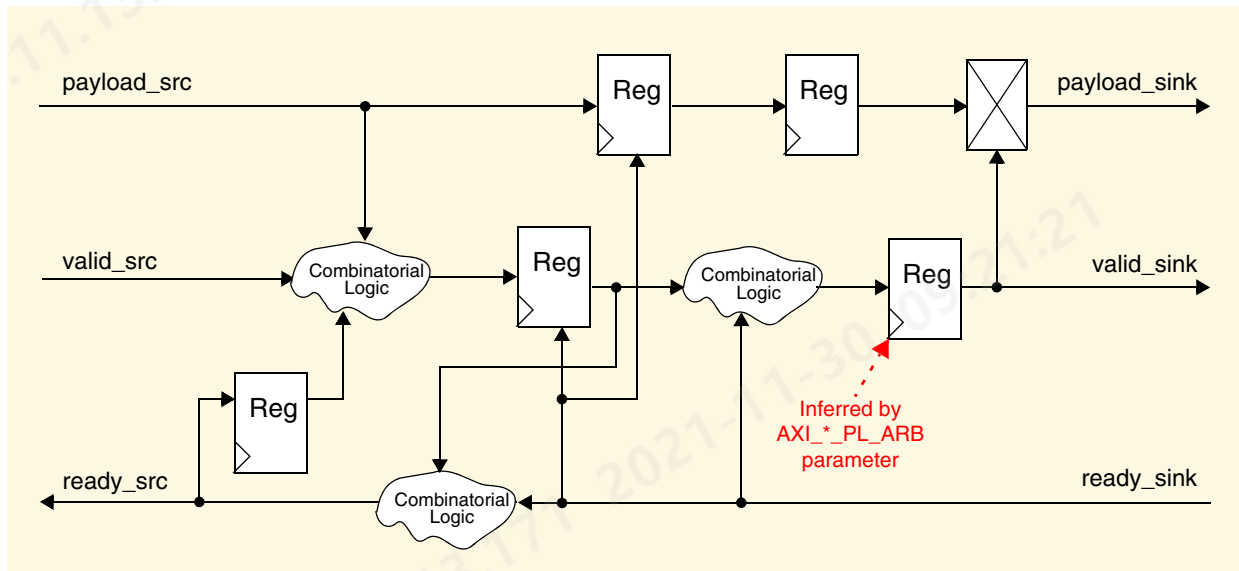


Note

If AXI_HAS_LOCKING == 1, then both the write and read address channels must use the same arbiter pipeline stage setting i.e. AXI_AR_PL_ARB == AXI_AW_PL_ARB.

You can select this pipelining stage standalone without selecting a full or forward register mode timing option, or it can be additional to a full or forward register mode timing option, giving a double pipelining of the channel. Note that each pipelining stage adds one cycle of latency through the channel.

[Figure 2-24](#) illustrates where in the critical path the pipelining stage is added with these parameters in a channel that already has a forward register timing option selected.

Figure 2-24 Timing Paths for Forward Registered Timing Mode with Pipeline Arbiter

For more information on the effect the channel arbiter pipeline has on shared layers, refer to [“Arbiter Pipeline Stage and Shared Layers”](#) on page 38.

The arbiter pipeline stage, combined with the requirement to implement write data ordering rules, introduces one cycle after a wlast beat where the next master to send a first write data beat to a slave cannot be determined. During this cycle, masters that have already begun interleaving to the slave—that is, if the interleaving depth at the slave is greater than 1—can access the slave. Normal operation then resumes one cycle later.

**Note**

Selecting the arbiter pipeline stage adds a potentially long path from the ready input of a channel sink—for example, aready_s1 for the read address channel at Slave 1—to a register internal to DW_axi. In testing, the arbiter pipeline stage has given positive results when used with an input delay of 20% on channel ready inputs. If input delays exceeding this are used, it is possible that the arbiter pipeline stage will have an overall negative input on the maximum frequency achievable with DW_axi. This effect can be countered by using a DW_axi_rs (register slice) module operating in reverse mode—in other words, registering the ready path only.

2.22.6 Multicycle Arbitration

You can use multicycle arbitration in order to reach high operating frequencies. Typically, the channel arbiters are part of the longest logic paths in the DW_axi, which becomes more pronounced for configurations with large numbers of masters/slaves, complex arbitration schemes, or fully dynamic arbitration priorities. Multi-cycle arbitration allows paths through the channel arbiters to be specified as multi-cycle paths, which eliminates them from the longest logic paths within the DW_axi.

You can use the AXI_[AR/AW/W]_MCA_NC_S(j) and AXI_[R/B]_MCA_NC_M(x) parameters to independently configure multi-cycle arbitration for every channel and for every master and slave port within the DW_axi. These parameters specify the number of arbitration cycles for each arbiter at each port in the DW_axi. You can configure these parameters under Pipelining Options > Multi-Cycle Arbitration in

the Specify Configuration activity of the coreConsultant GUI. The parameters are organized in table form for easy configuration.

**Note**

If a slave is connected to a single master, then multicycle arbitration is essentially disabled.

The default number of arbitration cycles for every arbiter is 1, which specifies that arbitration happens every cycle. You can select any value up to 16; a value greater than 1 infers logic for multi-cycle arbitration. This logic serves to hold arbiter inputs static for the configured number of arbitration cycles and sample the arbiter outputs at the appropriate time.

For every arbiter with multi-cycle arbitration selected, the automatically generated synthesis scripts set all appropriate paths through the arbiter as multi-cycle paths. You can export this synthesis script in SDC format from coreConsultant using the following steps:

1. Complete the Synthesize activity, setting the GenerateScriptsOnly parameter to 1 so that synthesis is not initiated. Click Apply; coreConsultant generates Design Compiler synthesis scripts.
2. At the coreConsultant command line, enter “write_sdc file.sdc”.

Enabling multi-cycle arbitration on a channel in the DW_axi does not alter the timing paths for the channel; it simply removes arbitration logic from the single-cycle logic paths.

Multi-cycle arbitration comes at the cost of a throughput penalty through the channel, which increases with the number of selected arbitration cycles. The worst-case arbitration latency due to multi-cycle arbitration is:

$$(2 * AXI_ [AR/AW/W/R/B]_MCA_NC_ [M(x)/S(j)] - 1$$

This latency occurs when a request is made to a channel arbiter one cycle after the start of the arbitration window. Therefore, it must wait for the rest of the current arbitration window before it is forwarded to the arbiter, and then its effect on the arbiter output is not seen until after another arbitration window has passed. For this reason, you should choose the lowest possible number of arbitration cycles that allow them to reach their performance goals.

If external arbiter priorities are enabled – AXI_HAS_EXT_PRIORITY=1 – and master or slave priorities are changing during runtime, you should enable the AXI_MCA_HLD_PRIOR parameter so that the DW_axi will hold master and slave priorities static for the configured number of arbitration cycles at each arbiter. Because multi-cycle arbitration can be configured differently at each port/channel, the priorities are registered separately at each port/channel where multi-cycle arbitration is enabled. If external priorities are being used, but priorities do not change during runtime, it is not required to enable this parameter. In this case, the priorities should be static for the number of cycles equal to the largest configured number of arbitration cycles before any valid is asserted to the DW_axi.

Multi-cycle arbitration cannot be selected on the write data channel of any slave port with a write interleaving depth of 1 – that is, AXI_WID_Sn = 1. In this case, there is no arbiter in the channel, so the parameter is redundant.

**Note**

If AXI_HAS_LOCKING=1, multicycle arbitration cannot be used on the address channels; that is, AXI_AR_MCA_NC_S(j) and AXI_AW_MCA_NC_S(j) must be equal to 1.

2.22.7 Write Address Channel to Write Data Channel Registering

By default, all write address channel (AW) to write data channel (W) paths are registered. Because of this register, if a master asserts $AWVALID_M(x)$ and $WVALID_M(x)$ in the same cycle, for a new write transaction $WVALID_S(j)$ of the addressed slave will not assert until $(PIPE_LATENCY + 1)$ cycles later. $PIPE_LATENCY$ is equal to the latency that exists on the channel due to channel pipelining options; for more information on registering, refer to “Forward Registered” on page 73, “Fully Registered” on page 75, and “Pipeline Channel Arbiter” on page 77.

By setting the $AXI_REG_AW_W_PATHS$ parameter to 0, the write address to write data channel paths will not be registered, and when a master asserts $AWVALID_M(x)$ and $WVALID_M(x)$ in the same cycle, $WVALID_S(j)$ will assert $PIPE_LATENCY$ cycles later. However, setting $AXI_REG_AW_W_PATHS$ to 0 will also reduce the operating frequency of the DW_axi instance, since longer logic paths will exist.



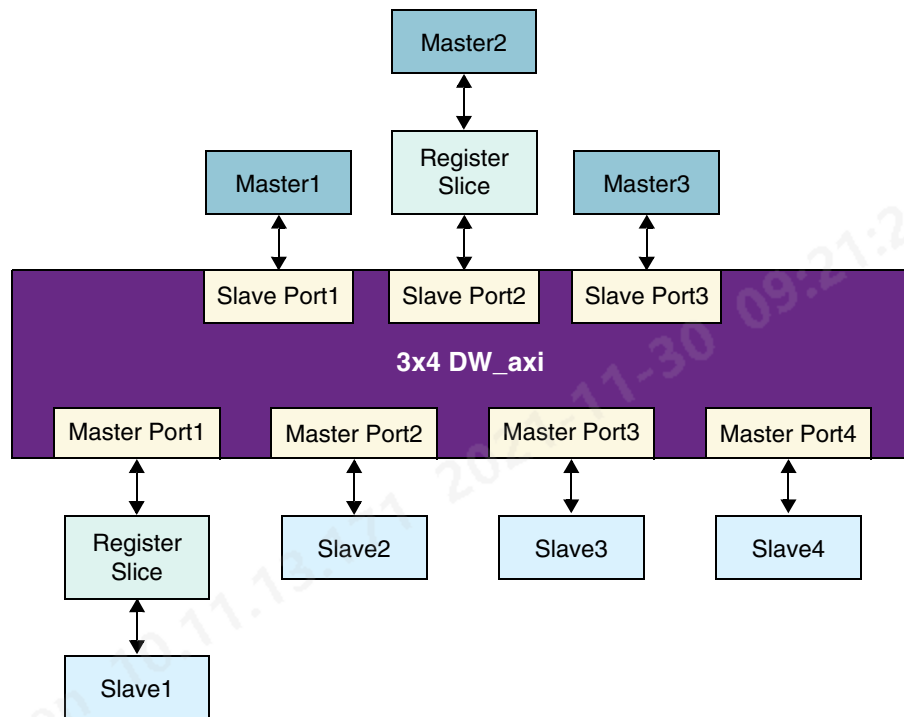
Note

In the following cases, setting $AXI_REG_AW_W_PATHS$ to 0 will not improve latency to a particular slave:

- If the write interleaving depth at a slave port is greater than 1 ($AXI_WID_S(j) > 1$) and the arbiter pipeline stage for the AW channel is not enabled
- If a slave's write data channel has a link from the shared write data channel and pipelining is not enabled on the shared AW layer ($AXI_AW_SHARED_PL=0$)
- If multicycle arbitration is enabled on the write data channel for a slave
- If slave is visible to multiple masters

2.22.8 External Register Slice

If you need to isolate long timing paths, then a register slice can be used at either the master or master port. Figure 2-25 shows the DW_axi where there is a register slice between Master1 and Master Port1 and Slave Port0 and Slave0.

Figure 2-25 Register Slice to Isolate Timing Paths

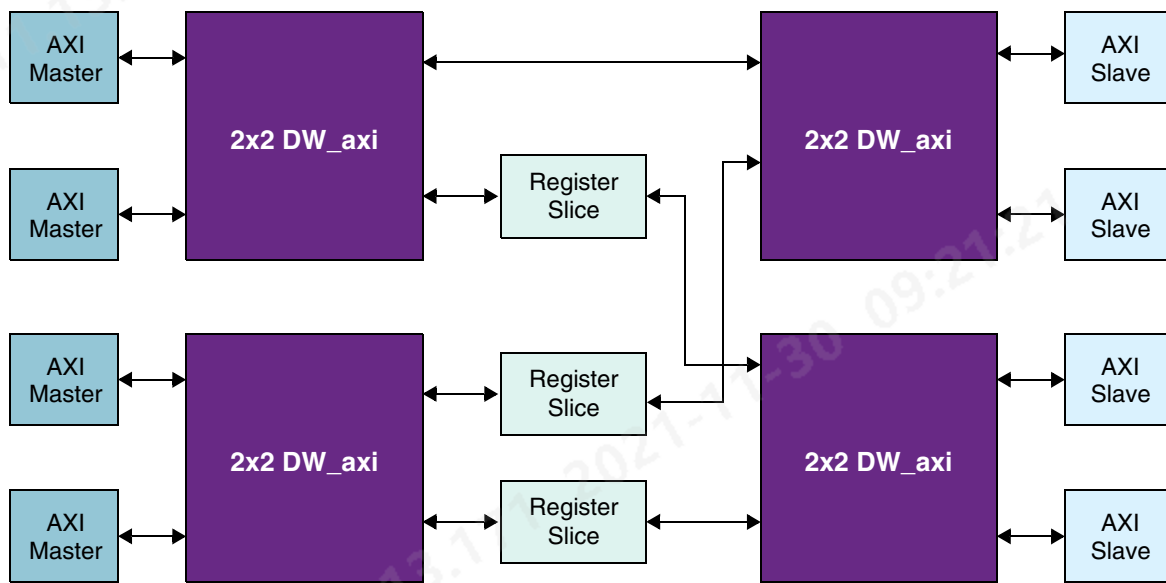
For the register slice, there are three modes of operation for the AXI channels, each of which can independently select a mode of operation for the register slice. For more details, refer to the [DesignWare DW_axi_rs Databook](#). The modes of operation are:

- Register the forward control path and the corresponding AXI channel payload
- Register the forward control path, the corresponding AXI channel payload, and the backward control path
- Bypass the register slice; do not register the forward control path, the backward control path, or the AXI channel payload path

Register slices can be instantiated on the periphery of the DW_axi in order to pipeline AXI channels, regardless of the timing mode selected for that channel within the DW_axi.

2.22.8.1 Cascading DW_axi Interconnects

In order to reduce the critical path, you can split the interconnect into multiple cascaded DW_axi interconnects with register slices between connecting ports, as illustrated in [Figure 2-26](#). In this example, a 4x4 interconnect is split into four 2x2 DW_axi interconnects.

Figure 2-26 Cascading of DW_axi Interconnects

2.23 Low-Power Interface

You can configure the DW_axi to include an AXI low-power handshaking interface. This interface allows the following:

- DW_axi informs the system low-power controller (LPC) when it has no outstanding transactions
- LPC requests the DW_axi to enter into a low-power state

Before issuing any APB transactions, ensure DW_axi is in not in a low power mode.

You can include this low-power interface in your design by setting the AXI_LOWPWR_HS_IF parameter to 1.

The low power handshaking interface includes the following signals:

- csysreq input – de-asserted by the LPC to initiate a low-power state; asserted by LPC to initiate an exit from a low-power state
- csysack output – de-asserted by DW_axi to acknowledge a request to enter a low-power state; asserted by DW_axi to acknowledge a request to exit a low-power state
- cactive output – de-asserted by DW_axi when the clock can be removed after the next positive edge; csysack must also be deasserted

The sequence of events for entering a low-power state is as follows:

1. The LPC requests the DW_axi to enter a low-power state by de-asserting the csysreq signal.
2. Since the DW_axi does not have a power-up or power-down sequence, it always acknowledges a csysreq signal on the next cycle by asserting or de-asserting the csysack signal.

When requested by the LPC, the low-power state depends on the value of the cactive signal at the time that the csysack signal is sampled by the LPC after the csysreq signal has been de-asserted.

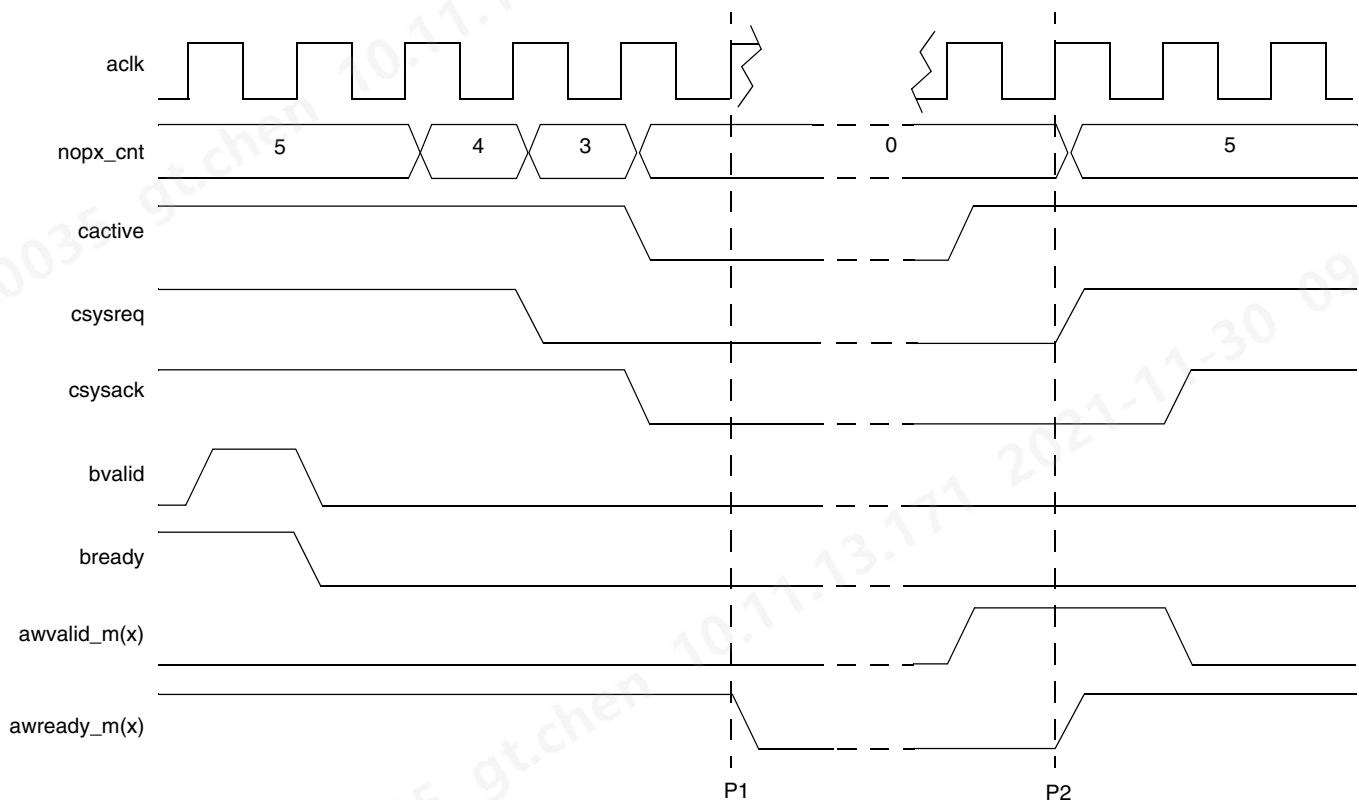
3. The cactive signal de-asserts when the DW_axi has no outstanding transactions. If you set the AXI_LOWPWR_NOPX_CNT parameter to "X" in coreConsultant, the cactive signal de-asserts after X cycles, during which there were no outstanding transactions, have elapsed. Note that transaction completion on DW_axi will be detected 1 cycle after the last transaction completes. Thus DW_axi will start counting down just one cycle after that.
4. DW_axi enters a low-power state when *all* of the cactive, csysreq, and csysack signals are low (0) when sampled at the positive edge of aclk; this is illustrated at P1 in Figure 2-27.

**Note**

Entry to a low-power state happens at any cycle in which the above condition is satisfied.

While in a low-power state, the awready_m, wready_m, and arready_m signals are held low, which prevents any new transaction from starting and thus allows the clock to be safely disabled; this is illustrated between points P1 and P2 of Figure 2-27.

Figure 2-27 Low-Power State Entry and Exit (AXI_LOWPWR_NOPX_CNT = 5)



The DW_axi exits a low-power state when the cactive signal goes high and is sampled at the positive edge of aclk; illustrated at P2 of Figure 2-27.

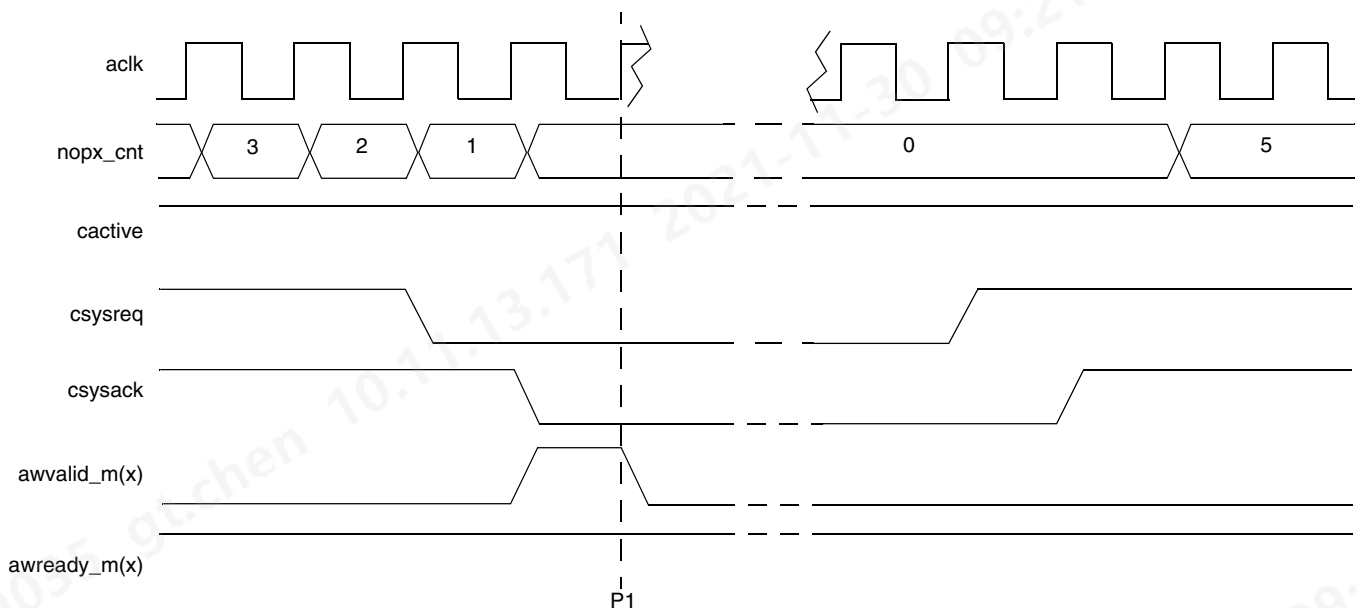
An exit from low-power can be initiated by:

- LPC asserting the csysreq signal, which causes the DW_axi to assert the cactive signal, illustrated in Figure 2-29.

- A master asserting the `awvalid_m(x)` or `arvalid(x)` signals of the DW_axi, which causes the DW_axi to assert the `cactive` signal, illustrated in Figure 2-27.

Figure 2-28 shows the DW_axi rejecting a request to enter a low-power state. At P1, the no-pending-transaction (`nopx_cnt`) counter has reached 0, but on this same cycle the `awvalid_m(x)` signal asserts, which keeps the `cactive` signal asserted. The LPC samples at P1 that the DW_axi has rejected the entry to low-power mode, and therefore must not remove the clock.

Figure 2-28 DW_axi Low-Power Interface Rejects Low-Power Entry



2.23.1 `cactive` Signal De-assertion

The `cactive` signal de-asserts `AXI_LOWPWR_NOPX_CNT + 1 aclk` cycles after the last pending transaction completes. If the `csysreq` signal de-asserts while the component is counting down to 0 from `AXI_LOWPWR_NOPX_CNT`, the `cactive` signal will de-assert on the next cycle.

After the positive edge of the `aclk` signal where the `cactive` signal and the `csysack` signal are sampled low, the low-power controller (LPC) may remove the clock(s) from the DW_axi (P1 in Figure 2-27). At this point the `bready` and `awready_m` signals will also be driven low.

2.23.2 `cactive` Signal Assertion

The `cactive` signal will assert combinatorially when the `awvalid_m(x)` or `arvalid_m(x)` signals are asserted, at which point it will then remain asserted until all outstanding transactions have completed and `AXI_LOWPWR_NOPX_CNT` cycles have elapsed.



Note

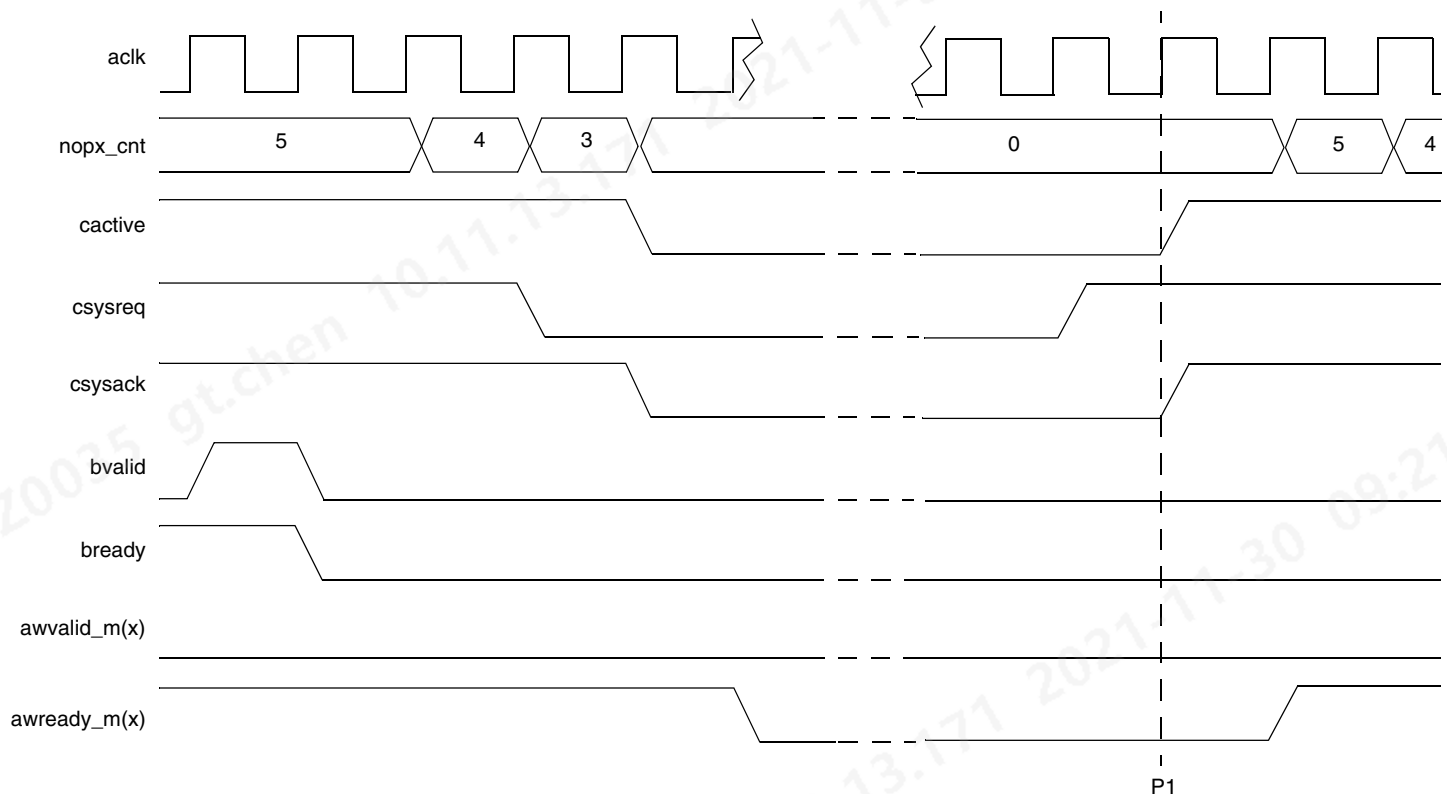
Early write data will not cause the `cactive` signal to assert or to remain asserted.

If the cactive signal is low and the csysreq signal is asserted, the DW_axi will assert the cactive signal at the same time as the csysack signal in order to complete the low-power exit handshake. After the clock edge where the cactive and csysack signals are sampled high on exit from low-power mode, the DW_axi will keep the cactive signal asserted for *AXI_LOWPWR_NOPX_CNT* cycles, after which it will either:

- De-assert until there are active transactions again.
- De-assert until another low-power exit handshake is completed.

This can be seen at P1 in Figure 2-29 where the cactive signal is asserted 1 clock cycle after the csysreq signal asserts, and the no-pending-transaction counter (nopx_cnt) starts to decrement again from the next cycle.

Figure 2-29 LPC Initiates Low-Power Exit



Note

When coming out of reset, if the awvalid_m(x) and arvalid_m(x) signals equal 0:

- cactive signal equals 0, if AXI_LOWPWR_NOPX_CNT parameter equals 0
- cactive signal equals 1, if AXI_LOWPWR_NOPX_CNT parameter is greater than 0 and will de-assert when *AXI_LOWPWR_NOPX_CNT* clock cycles have elapsed

2.24 QoS Controller

The QoS Controller in DW_axi is designed to meet the requirements of both bandwidth sensitive and latency sensitive masters, in a system with multiple masters. The QoS Controller provides a means by which an SoC architect can:

- Achieve better control over bandwidth
- Dynamically minimize average latency

Masters are classified as:

- Bandwidth sensitive – Master must get a required amount of bandwidth from a slave. If that bandwidth is not received, the overall performance of the system is degraded or results in data corruption.
- Latency sensitive – Masters require a fixed or minimum latency for the completion of transfers.
- Best-effort – Masters do not have stringent requirements of bandwidth and latency, so requirements are met if masters are served whenever possible based on “best effort”.

In order to meet the above requirements, the Quality of Service (QoS) controller does the following:

- Allows bandwidth sensitive masters to regulate incoming traffic on the master port based on the desired rate
- Allows latency sensitive masters to prioritize requests based on the QoS value

**Note**

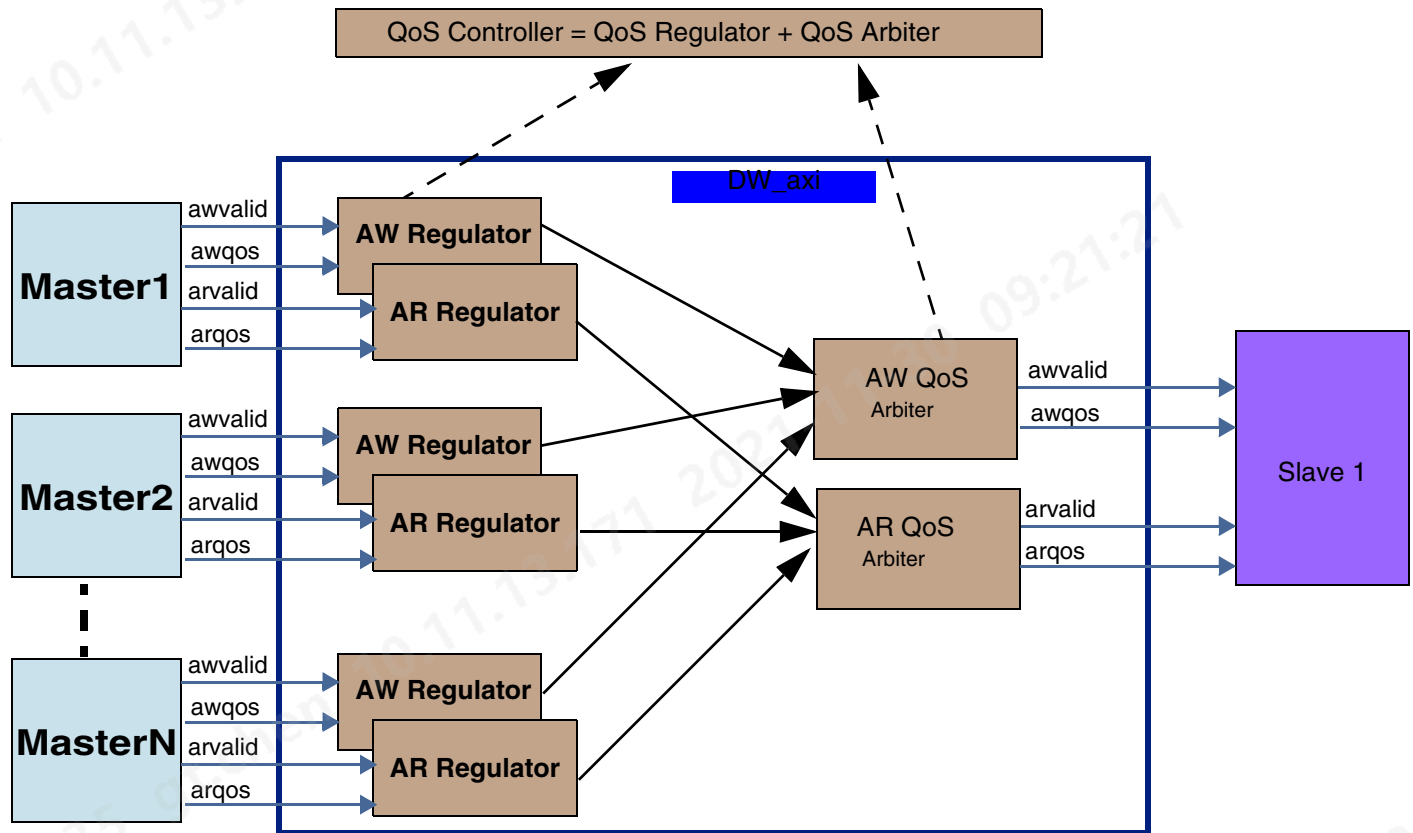
Quality of Service (QoS) functionality is source-only; that is, you must have a DWC-AMBA-Fabric-Source license in order to use the QoS features.

The QoS Controller in DW_axi is comprised of the following:

- The QoS Regulator – regulates the rate of the incoming traffic on a master port. The QoS Regulator can be configured on the write address and/or read address channel of each master port.
- QoS Arbiter – prioritizes requests based on the priority value. The QoS Arbiter can be configured on the write address and/or read address channel of each Slave port. QoS arbitration is described in [“QoS Arbitration Type \(Dynamic Priority\)”](#) on page 50.

[Figure 2-30](#) shows the components of a QoS Controller in a system with multiple masters accessing a common slave.

Figure 2-30 QoS Controller



2.24.1 QoS Regulator

The QoS Regulator considers the following attributes to regulate incoming requests:

- “[Transaction Rate](#)” on page 87
- “[Burstiness](#)” on page 88
- “[Peak Rate](#)” on page 88
- “[Slave Ready Dependency](#)” on page 89

For details regarding the registers used for programming the QoS Regulator, refer to QoS Internal Registers for Master Port.

For more information on programming the QoS Controller, refer to “[How to Program QoS Registers](#)” on page 97.

For details on integrating the QoS Controller, refer to “[Quality of Service \(QoS\) Integration](#)” on page 265.

2.24.1.1 Transaction Rate

The transaction rate of the QoS Regulator is the maximum rate at which incoming requests are accepted on the master port address channels. For example, if the transaction rate is programmed at 1/64, the QoS

Regulator accepts a maximum of one request in every 64 clocks. The transaction rate can be programmed by the QoS internal registers, as described in QoS Internal Registers for Master Port.

Internally, the QoS Regulator uses a token generator to generate tokens based on the programmed transaction rate. Tokens are consumed at the rate of one token per request. Tokens that are not consumed can be accumulated up to a maximum number, which is defined by the burstiness value.

2.24.1.2 Burstiness

The burstiness value determines the maximum number of tokens that can be accumulated in the absence of requests on the master port address channels. The burstiness value can be programmed by the QoS internal registers, as described in QoS Internal Registers for Master Port.

For example, if the burstiness value is programmed at 4 and the transaction rate is $1/64$, then in $64 \times n$ clock cycles, n tokens are generated. During the silent period of a master, when no requests are received, tokens are accumulated up to the maximum defined by the burstiness value, in this case 4. Any further tokens that are generated during the silent period are discarded.

If a master sends requests at a regular rate, then burstiness is not required and can be set at zero. For example, if a master sends a maximum of one request in every 64 clocks, then setting a transaction rate of $1/64$ and a burstiness value of zero ensures that the QoS Regulator accepts all requests without delay.

However, in the case of bursty traffic, there may be a silent period, followed by a series of transactions within a short period. For example, if the master is silent for 64×4 clocks and then sends 4 transactions within the next 64 clocks, then, if the burstiness value is programmed to zero, the QoS Regulator would accept the first request and then have to wait for 3×64 clock cycles to accept all the remaining requests. However, if the burstiness value is programmed to 4, the QoS Regulator can accept all the requests without delay, because 4 tokens would have already been accumulated during the silent period.

2.24.1.3 Peak Rate

Peak rate is the maximum rate at which accumulated tokens are consumed after a silent period. The peak rate can be programmed by the QoS internal registers, as described in QoS Internal Registers for Master Port.

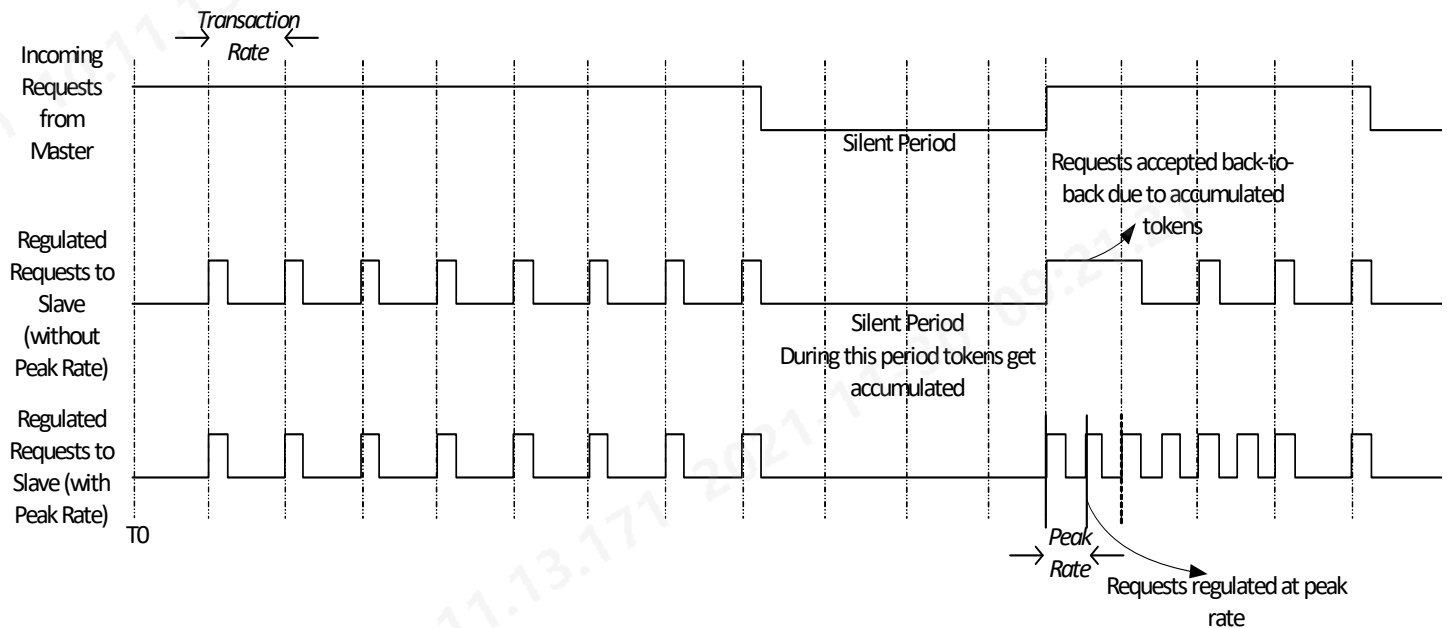
When the peak rate is not enabled, tokens are consumed as soon as requests are received from the master. For example, continuing the previous scenario, if the peak rate is set to $1/32$, then the QoS Regulator can accept a maximum of one request in every 32 clocks, and would have to wait for 3×32 clock cycles to accept all the requests in this scenario.

Peak rate should always be faster than the transaction rate. If the peak rate is programmed to be slower than the transaction rate, then the maximum transaction rate is never met.

Peak rate is useful only when burstiness is non-zero. When burstiness is zero, then requests are accepted by the QoS Regulator at transaction rate only. For example, if the transaction rate is $1/64$ and the peak rate is programmed as $3/64$, then burstiness should be programmed as 3 or higher, in order for the peak rate to be useful. If burstiness is 2 or lower, then sufficient tokens cannot be accumulated to meet the limit specified by the peak rate.

Peak rate is used to help in interleaving transactions from other masters, thereby reducing the latency of other masters.

Figure 2-31 shows how the transaction rate, burstiness, and peak rate interact to regulate traffic.

Figure 2-31 Transaction Rate, Burstiness, and Peak Rate in QoS Controller

2.24.1.4 Slave Ready Dependency

The QoS Regulator performance depends on the relationships between the following signals connected on the slave port:

- `arvalid_s(j)` and `arready_s(j)`
- `awvalid_s(j)` and `awready_s(j)`

A slave can assert `awready_s(j)/arready_s(j)` in either of the following situations:

- The `awready_s(j)/arready_s(j)` signal is asserted by default and is deasserted only when the slave has received the maximum number of outstanding transactions and cannot accept new requests.

In this case, you can enable slave ready dependency in the QoS Regulator. When this feature is enabled, token generation halts when the `awready_s(j)/arready_s(j)` signal is deasserted. If this feature is not enabled, then tokens continue to be generated and accumulated even when the `awready_s(j)/arready_s(j)` signal is deasserted. This can result in congestion when the slave becomes ready (the `awready_s(j)/arready_s(j)` signal is asserted).



Note

All slaves visible to a master must have the same behavior, where the `awready_s(j)/arready_s(j)` signal is asserted by default.

- The `awready_s(j)/arready_s(j)` signal is deasserted by default and is asserted only when the `awvalid/arvalid` signal is asserted.

In this case, the slave ready dependency feature should not be enabled. If this feature is enabled, token generation halts after the generation of one token and does not resume until the token is consumed. This behavior is similar to the case where burstiness is programmed as 1.

The behavior of this feature based on the default state of the `awready_s(j)/arready_s(j)` is shown in Table 2-5.

Table 2-5 Slave Ready Dependency

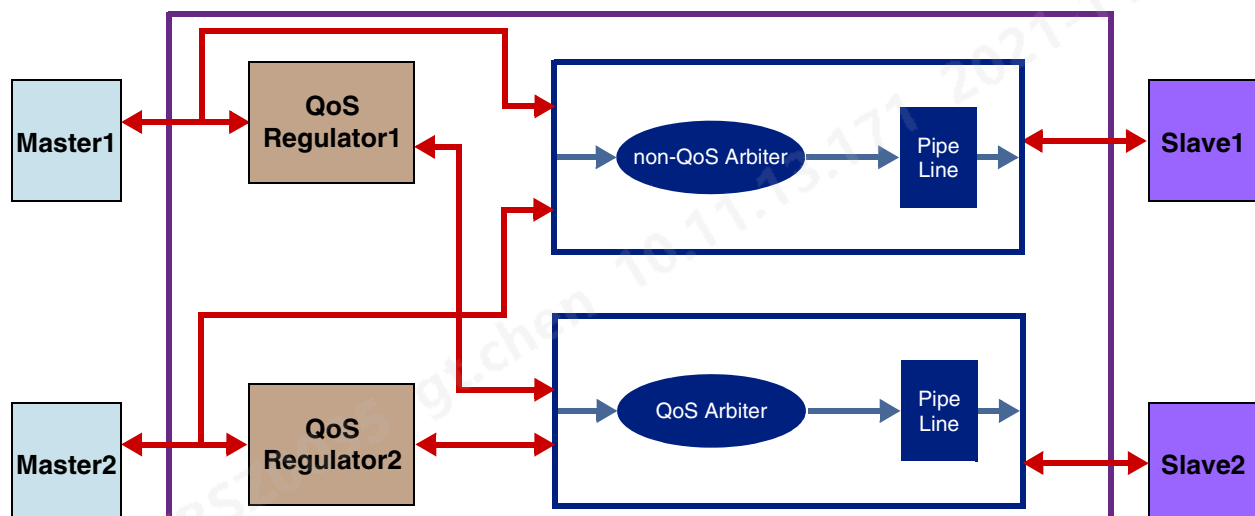
Default State of Slave	Slave Ready Dependency	Token Generation
<code>awready_s(j)/arready_s(j)</code> asserted	Enabled	Token generation halts when <code>awready_s(j)/arready_s(j)</code> is deasserted.
<code>awready_s(j)/arready_s(j)</code> asserted	Disabled	Token generation continues irrespective of the status of the <code>awready_s(j)/arready_s(j)</code> signal.
<code>awready_s(j)/arready_s(j)</code> deasserted	Enabled	Token generation halts after the generation of one token. Similar to <code>burstiness = 1</code> .
<code>awready_s(j)/arready_s(j)</code> deasserted	Disabled	Token generation continues irrespective of the status of the <code>awready_s(j)/arready_s(j)</code> signal.

Slave ready dependency can be programmed by the QoS internal registers, as described in QoS Internal Registers for Master Port.

2.24.2 QoS Architecture

Figure 2-32 shows Master1 and Master2 with read address channels on dedicated links with Slave1 and Slave2. QoS functionalities are enabled within the DW_axi because the Slave2 arbiter type is QoSArbiter; the Slave1 arbiter can be any other type. Requests for Slave1 access are not passed through the QoS Regulator, while requests for Slave2 are regulated by the QoS Regulator.

Figure 2-32 Dedicated Architecture – QoS and Non-QoS Arbiter



2.25 ACE-Lite Support

The DW_axi supports the ACE-Lite signaling interface. The ACE-Lite interface can be enabled by selecting the parameter `AXI_INTERFACE_TYPE = ACELITE`. The ACE-Lite interface includes the following signals.

- On the master port write address and read address channels:
 - `awsnoop_m(x)`, `awdomain_m(x)`, `awbar_m(x)`
 - `arsnoop_m(x)`, `ardomain_m(x)`, `arbar_m(x)`
- On the slave port write address and read address channels:
 - `awsnoop_s(j)`, `awdomain_s(j)`, `awbar_s(j)`
 - `arsnoop_s(j)`, `ardomain_s(j)`, `arbar_s(j)`

The snoop and domain signals are pass-through signals and are routed in same way as any other read or write address channel payload. These signals do not get modified within the interconnect and are simply passed to the slave ports.

Barrier transactions are currently not supported in DW_axi. The barrier input signals (`awbar_m(x)` and `arbar(x)`) are expected to be driven to zero.

**Note**

DW_axi does not perform any control operation on ACE-Lite signals.

3

Programming the DW_axi

The DW_axi can be programmed via software registers, which are described in more detail in “[Register Descriptions](#)” on page 165.

3.1 Software Registers

This section describes how to program the DW_axi software registers.

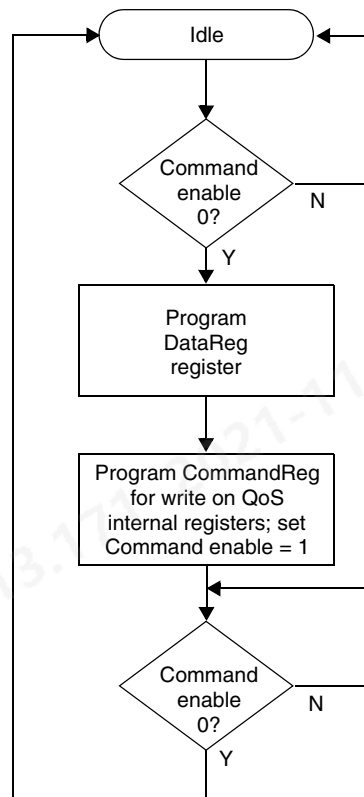
3.1.1 Programming Considerations

The following are considerations you should take into account when programming the DW_axi:

- The APB interface is used to configure the DW_axi registers.
- The DW_axi implements a set of registers for each master port channel. The required value must be written in the DataReg register before issuing a write command in the CommandReg register. The targeted register bits are updated with the contents of the DataReg register.
- Reading from a reserved location or reserved bits always returns 0.
- The software must poll the Command Enable bit of the CommandReg register after issuing a read command. The contents of the DataReg register are valid only when the Command Enable bit of the CommandReg register is zero. This imposes a restriction on back-to-back DataReg reads.

3.1.2 QoS Internal Register Write Operation

The sequence for performing writes to the internal registers is illustrated in [Figure 3-1](#).

Figure 3-1 QoS Internal Register Write Operation

1. The APB master updates the DataReg register with the intended content.
2. The APB master updates:
 - ❑ CommandReg register bits with intended command
 - ❑ Associated parameters
 - ❑ Write command on CommandReg[30]
3. The Command Enable bit must be set to 1. This signifies that the command in CommandReg is in progress. When the command completes, the Command Enable bit is automatically cleared to 0.
4. Data from DataReg is written in the targeted QoS internal register that is specified by the CommandReg bits for a selected channel and master port.

If the register does not exist, then an error bit – CommandReg[28] – is set and the write data are ignored. For example, if a configuration has three masters, the QoS internal registers do not exist for Master 4. If a write operation occurs for Master 4, it flags the error bit.

5. The Command Enable bit is reset to 0 by the interconnect upon the successful completion of a command or on an error status.

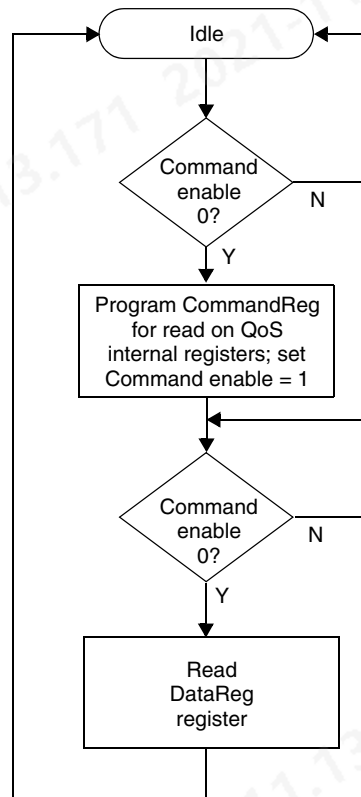
**Note**

- The APB master must write the data to the DataReg register before writing a command to the CommandReg register.
- Once the Command Enable bit is set, the CommandReg cannot be allowed to update until the command execution completes and the interconnect clears the Command Enable bit to 0.

3.1.3 QoS Internal Register Read Operation

The sequence for performing reads to the internal registers is illustrated in [Figure 3-2](#).

Figure 3-2 QoS Internal Register Read Operation



1. The APB master updates:
 - CommandReg register bits with intended command
 - Associated parameters
 - Read operation on CommandReg[30]
2. The Command Enable bit must be set to 1. This signifies that the command in CommandReg is in progress. When the command completes, the Command Enable bit is automatically cleared to 0.
3. The DataReg fields are updated by the targeted QoS internal register data bits. CommandReg specifies the following:
 - Targeted QoS internal register address offset

- Channel
- Master port number

If the register does not exist, then an error bit – CommandReg[28] – is set and the read data are ignored. For example, if a configuration has three masters, the QoS internal registers do not exist for Master 4. If a read operation occurs for Master 4, it flags the error bit.

4. When the command successfully completes, the DataReg is updated with read data for the QoS internal register that is programmed by the APB master. Additionally, the Command Enable bit is automatically cleared to 0.
5. The APB master reads the CommandReg in order to confirm if the Command Enable bit is reset to 0.
6. When the APB master confirms that the Command Enable bit is set to 0, it reads the DataReg to obtain the read data.

3.2 DW_axi Configuration for QoS

This section describes the usage model of the QoS feature in the DW_axi.



Note

Quality of Service (QoS) functionality is source-only; that is, you must have a DWC-AMBA-Fabric-Source license in order to use the QoS features.

In order to enable the QoS feature of the DW_axi, you must configure the DW_axi in coreConsultant or coreAssembler.

1. To enable the QoS in the DW_axi, configure the parameter AXI_HAS_QOS to 1.
2. Configure a slave port for the arbiter type by selecting the slave port arbiters as QoSArbiter on the read address channel and write address channel. Based on your requirements, you can configure any slave port arbiter as QoSArbiter.
3. Configure the master port for rate regulation:
 - The QoS regulator enables you to regulate the traffic from each master port on the read address channel or write address channel.
 - The option to select the QoS regulator on the master port is available only if the type of any slave port arbiter is defined as “QoSArbiter”. At this point, you can select any master port read address channel or write address channel that needs rate regulation.
4. Configure the master port for the QoS source.
 - The QoS value that defines the priority can be either internal or external, but not both.
 - External source – master port signals awqos_m(x)/arqos_m(x) are used as the source for the QoS value; internal registers are not available.
 - Internal source – internal registers are used as the source for the QoS value; master port signals awqos_m(x)/arqos_m(x) are not available.
 - In the case of an external source, the QoS value is decided by the master for each transaction; in the case of an internal source system, the software decides the value. The internal source system option is not available for AXI4 implementations.

3.3 How to Program QoS Registers

The DW_axi has a set of registers that you can use to program the QoS regulator. You use the APB interface to program the QoS internal registers. The following QoS attributes can be programmed:

- Transaction rate
- Burstiness
- Peak rate
- Slave ready dependency

For more information on the registers used for the QoS features, refer to “[Register Descriptions](#)” on page 165.

Transaction rate and peak rate values are fractions, but they must be programmed in the register as a binary representation. [Table 3-1](#) shows the binary coding for the fractions used in programming these values in the DataReg register.

Table 3-1 Register Programming Value Encoding

Binary Value	Fractional Equivalent
1000_0000_0000	1/2
0100_0000_0000	1/4
0010_0000_0000	1/8
0001_0000_0000	1/16
0000_1000_0000	1/32
0000_0100_0000	1/64
0000_0010_0000	1/128
0000_0001_0000	1/256
0000_0000_1000	1/512
0000_0000_0100	1/1024
0000_0000_0010	1/2048
0000_0000_0001	1/4096



Note

Programming the transaction rate as 0000_0000_0000 disables token generation and traffic is not regulated based on the tokens.

The burstiness value is programmed in the DataReg register. When the CommandReg [2:0] bits enable the Burstiness Regulator internal register, the DataReg register programs the burstiness value. Burstiness value is one more than the value defined in the DataReg register.

$$\text{Burstiness} = \text{DataReg}[23:16] + 1$$

For example, if the value programmed in the DataReg[23:16] bits is 3, then the burstiness value is 4.

Programming the peak rate is similar to programming the transaction rate. The binary value corresponding to the fractional value must be programmed in the DataReg[31:20] bits when CommandReg[2:0] bits are programmed to select the PEAK_RATE_XCT_RATE internal register.

**Note**

Programming the peak rate as 0000_0000_0000 disables the peak rate control logic and traffic is not regulated by the peak rate logic.

The sequences that follow show an example of how to program the QoS regulator on the write address channel of Master Port 2 for the following:

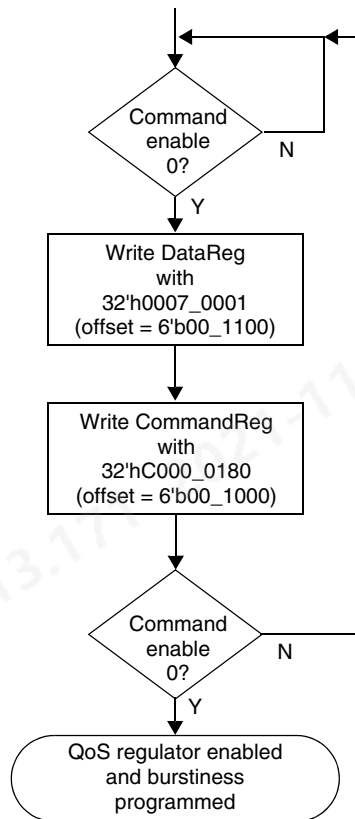
- Burstiness of 8
- Transaction rate of 1/64
- Peak rate of 1/32
- QOS_VALUE of 4'h3

3.3.1 Programming Sequence Examples

The following sections illustrate programming sequences for different QoS features.

3.3.1.1 Regulator Enable and Burstiness Programming

The flow diagram in [Figure 3-3](#) illustrates the sequence for enabling the QoS regulator and programming burstiness.

Figure 3-3 Flowchart for Regulator Enable and Burstiness Programming

Use the following steps to enable the QoS regulator and program burstiness:

1. Poll the CommandReg register by an APB read – the Command Enable bit must be read as 0; otherwise wait until it becomes 0.
2. Write the DataReg register with a value of 32'h0007_0001 (offset = 6'b00_1100). The BURSTINESS_REGULATOR_EN internal register is set for:
 - BURSTINESS_REGULATOR_EN[23:16] + 1 – burstiness
 - BURSTINESS_REGULATOR_EN[0] – regulator enable
3. Write the CommandReg register with value of 32'hC000_0180 (offset = 6'b00_1000).
 - CommandReg[31] – command enable
 - CommandReg[30] – write operation
 - CommandReg[11:8] – master port 2
 - CommandReg[7] – write address channel
 - CommandReg[2:0] – address offset for BURSTINESS_REGULATOR_EN register

**Note**

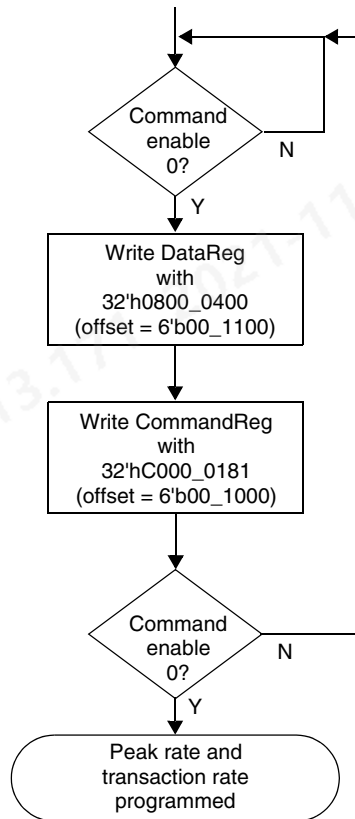
Selecting the QoS regulator in coreConsultant enables the regulator module at the master port. By default the regulator is disabled; that is, traffic is not regulated. You must enable the regulator through programming.

4. This transaction completes when the Command Enable bit is cleared by the interconnect.

3.3.1.2 Programming Peak Rate and Transaction Rate

The flow diagram in [Figure 3-4](#) illustrates the sequence for programming the peak rate and transaction rate.

Figure 3-4 Flowchart for Programming Peak Rate and Transaction Rate



Use the following steps to program the peak rate and transaction rate:

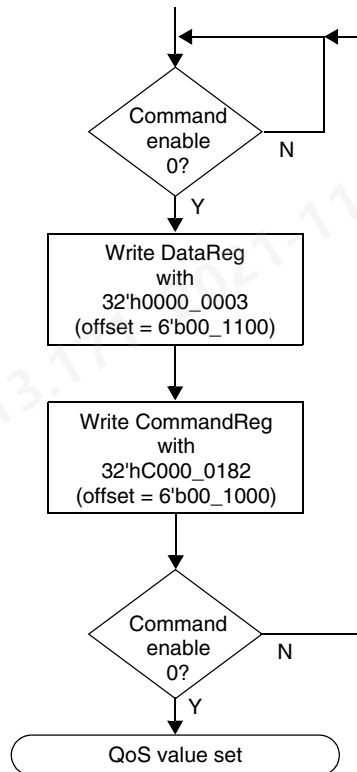
1. Poll the CommandReg register by an APB read – the Command Enable bit must be read as 0; otherwise wait until it becomes 0.
2. Write the DataReg register with a value of 32'h0800_0400 (offset = 6'b00_1100). The PEAK_RATE_XCT_RATE internal register is set for:
 - PEAK_RATE_XCT_RATE[31:20] – peak rate
 - PEAK_RATE_XCT_RATE[15:4] – transaction rate
3. Write the CommandReg register (Offset = 6'b00_1000) with 32'hC000_0181.
 - CommandReg[31] – command enable
 - CommandReg[30] – write operation
 - CommandReg[11:8] – master port 2
 - CommandReg[7] – write address channel
 - CommandReg[2:0] – address offset for PEAK_RATE_XACT_RATE register

4. This transaction completes when the Command Enable bit is cleared by the interconnect.

3.3.1.3 Programming QoS Value

The flow diagram in [Figure 3-5](#) illustrates the sequence for programming the QoS value.

Figure 3-5 Flowchart for Programming QoS Value



Use the following steps to program the QoS value:

1. Poll the CommandReg register by an APB read – the Command Enable bit must be read as 0; otherwise wait until it becomes 0.
2. Write the DataReg register with a value of 32'h0000_0003 (offset = 6'b00_1100). The QOS_VALUE internal register is set for:
 - QOS_VALUE[3:0] – QoS value
3. Write the CommandReg register with a value of 32'hC000_0182 (offset = 6'b00_1000).
 - CommandReg[31] – command enable
 - CommandReg[30] – write operation
 - CommandReg[11:8] – master port 2
 - CommandReg[7] – write address channel
 - CommandReg[2:0] – address offset for QOS_VALUE register

**Note**

QoS value is valid only when configured for an internal source.

4. This transaction completes when the Command Enable bit is cleared by the interconnect.

4

Parameter Descriptions

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

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

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

- Top Level Parameters on [page 105](#)
- AXI Source Code Configuration on [page 110](#)
- AXI Deadlock Notification Configuration on [page 111](#)
- Pipelining Options / Channel Pipelining Stages on [page 112](#)
- Pipelining Options / AW to W Registering on [page 116](#)
- Pipelining Options / Shared Layer Pipelining on [page 117](#)
- Pipelining Options / Multi-Cycle Arbitration on [page 119](#)
- Mult-Tile/Bi-directional Command / System Master Numbers on [page 125](#)
- Mult-Tile/Bi-directional Command / Interconnecting Slave Ports on [page 126](#)
- Mult-Tile/Bi-directional Command / Interconnecting Master Port configuration on [page 127](#)
- Arbitration Options / Arbitration Type on [page 128](#)
- Arbitration Options / Master and Slaves Priority on [page 138](#)
- Sideband Signals on [page 140](#)
- User Signals on [page 142](#)
- Slave Visibility on [page 144](#)
- Architecture Options / Architecture Options on [page 145](#)
- Slave Address Map on [page 150](#)

- Master Port Configuration on [page 155](#)
- Slave Port Configuration on [page 158](#)
- QoS Options / Register Interface Configuration on [page 161](#)
- QoS Options / QoS Signals on [page 162](#)
- QoS Options / QoS Regulator Module Configuration on [page 163](#)

4.1 Top Level Parameters

Table 4-1 Top Level Parameters

Label	Description
System Parameters	
Select AXI Interface Type	<p>The parameter selects the type of AXI interface.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ AXI3 (0) ■ AXI4 (1) ■ ACE-Lite (2) <p>Default Value: AXI3</p> <p>Enabled: DWC-AMBA-Fabric-Source license required for AXI4 and ACE-Lite</p> <p>Parameter Name: AXI_INTERFACE_TYPE</p>
AXI Data Bus Width	<p>This is the bit width of the data bus, which applied to all interfaces.</p> <p>Values: 8, 16, 32, 64, 128, 256, 512</p> <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_DW</p>
AXI Address Bus Width	<p>This is the bit width of the address bus, which is applied to all interfaces.</p> <p>Values: 32, ..., 64</p> <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_AW</p>
Number of AXI Masters	<p>This is the number of AXI masters connecting to DW_axi. A master port is instantiated for each external AXI master.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 2 Masters</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_NUM_MASTERS</p>
Number of AXI Slaves	<p>This is the number of AXI slaves connecting to DW_axi. A slave port is instantiated for each external AXI slave.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 2 Slaves</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_NUM_SLAVES</p>

Table 4-1 Top Level Parameters (Continued)

Label	Description
AXI ID Width of Masters	<p>This is the ID bus width of all five channels connected to an external master. All masters have the same ID width for all five AXI channels. If a master's ID width for any channel is less than the configured value, then the unused bits should be connected externally to 0. For a single master system this parameter will default to 0.</p> <p>Values: 1, ..., 12 Default Value: 4 Enabled: Always Parameter Name: AXI_MIDW</p>
AXI ID Width of Slaves	<p>Read-only parameter. This is the ID bus width of all five AXI channels connected to an external slave. It is a function of the AXI ID Width of a Masters (AXI_MIDW) and the number of AXI masters (AXI_NUM_SYS_MASTERS).</p> $AXI_SIDW = AXI_MIDW + \text{ceil}(\log_2 (AXI_NUM_SYS_MASTERS))$ <p>This parameter is calculated automatically, and the same width is applied to all slaves. A slave must use all of these bits because they are a concatenation of the ID bus from the master and the master number from which the transfer originated. All slaves have the same ID width for all five AXI channels. For more information about this feature, see the "ID Bus" section in the DW_axi Databook.</p> <p>Values: 1, ..., 20 Default Value: 5 Enabled: Always Parameter Name: AXI_SIDW</p>
AXI Burst Length Width	<p>This is the width of the burst length signal for both read and write address channels on both the master and slave ports. The AXI protocol specifies this as a 4-bit value, but this width is configurable to 8 bits wide in order to support longer bursts up to 256 data beats.</p> <p>Values: 4, ..., 8 Default Value: 4 Enabled: Always Parameter Name: AXI_BLW</p>
Enable TrustZone Feature?	<p>When this is set to True, it enables the TrustZone support. An input port exists for each external AXI slave, indicating whether the slave is secure. When set to False, the TrustZone signals are not included on the I/O, and TrustZone functionality is not supported. For more information about this feature, see the "Trustzone Support" section in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_HAS_TZ_SUPPORT</p>

Table 4-1 Top Level Parameters (Continued)

Label	Description
Enable Remap Mode?	<p>When this is set to True, the remap feature is supported in which two address maps are included in each read and write address decoder. Additionally, an external remap_n signal is included on the I/O, which controls the selection of the address maps. If this is set to False, then remap_n is not included on the I/O. For more information, see the "Remap Operation" section in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_REMAP_EN</p>
Use External Decoders?	<p>When set to True, DW_axi uses external read and write decoders for each master port. These address decoders can have different memory maps and multiple remap options. When set to False, DW_axi uses a internal read and write address decoders for each master port. These address decoders are fixed to a maximum of two remap options and all read/write address decoders have the same memory map. For more information, see the "External Decoder" section in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_XDCDR</p>
Support Locked Transactions ?	<p>When set to True, AXI locked transactions are supported. This allows all masters to lock a slave for exclusive access for a locked sequence. Parameter is disabled for single master systems. Note: Disabling this parameter does not prevent the DW_axi from forwarding locking transactions; it just removes the logic required for the DW_axi to implement locking rules.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: ((AXI_INITIAL_LOCKDOWN == 0) && (AXI_NUM_MASTERS > 1)) && (AXI_INTERFACE_TYPE == 0) && (AXI_HAS_QOS == 0)</p> <p>Parameter Name: AXI_HAS_LOCKING</p>

Table 4-1 Top Level Parameters (Continued)

Label	Description
Enable QoS features ?	<p>If set to True, it enables QoS support. This allows the setting of QoS Arbiter and QoS Regulator related Configuration.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: DWC-AMBA-Fabric-Source License required for enabling this feature</p> <p>Parameter Name: AXI_HAS_QOS</p>
Interconnect Tiling Options	
Enable Bi-directional Command Support feature?	<p>When this is set to True, it enables the Bi-Directional Command support. This enables the setting of Master System numbers, and the creation of interconnecting Masters and Slaves.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: AXI_NUM_MASTERS > 1</p> <p>Parameter Name: AXI_HAS_BICMD</p>
Number of AXI System Masters	<p>This is the number of AXI masters in the entire system. (System in this instance being all connected DW_axi) Excluding interconnecting Master ports</p> <p>Values: 1, ..., 64</p> <p>Default Value: AXI_NUM_MASTERS</p> <p>Enabled: AXI_HAS_BICMD == 1</p> <p>Parameter Name: AXI_NUM_SYS_MASTERS</p>
Number of Interconnecting Master Ports	<p>This is the number of Master ports on the interconnect that are configured as Interconnecting Master Ports. If AXI_HAS_BICMD = 1, AXI_NUM_ICM cannot be 0.</p> <p>Values: 0, ..., 4</p> <p>Default Value: 0</p> <p>Enabled: AXI_HAS_BICMD == 1</p> <p>Parameter Name: AXI_NUM_ICM</p>

Table 4-1 Top Level Parameters (Continued)

Label	Description
Enable Multi Tile Deadlock Avoidance ?	<p>When this is set to True, it enables the AXI_ACC_NON_LCL_SLV_Sx parameters, which are used to mark certain slave ports as accessing slaves through other DW_axi instances. This information is then used to prevent deadlock in multi tile systems.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: AXI_NUM_SLAVES > 1</p> <p>Parameter Name: AXI_EN_MULTI_TILE_DLOCK_AVOID</p>
Low power interface configuration	
Include Low-Power Interface?	<p>When set to True, the low-power handshaking interface (csysreq, csysack, and cactive signals) and associated control logic is implemented. If false, no support for low-power handshaking interface is provided.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_LOWPWR_HS_IF</p>
No outstanding transactions counter?	<p>This parameter sets the number of AXI clock cycles to wait before cactive signal de-asserts, when there are no pending transactions.</p> <p>Note that if csysreq de-asserts while waiting this number of cycles, cactive will immediately de-assert. If a new transaction is initiated during the wait period, the counting will be halted, cactive will not de-assert, and the counting will be re-initiated when there are no pending transactions.</p> <p>Values: 0x0, ..., 0xffffffff</p> <p>Default Value: 0x0</p> <p>Enabled: AXI_LOWPWR_HS_IF == 1</p> <p>Parameter Name: AXI_LOWPWR_NOPX_CNT</p>

4.2 AXI Source Code Configuration Parameters

Table 4-2 AXI Source Code Configuration Parameters

Label	Description
AXI Source Code Configuration	
Use DesignWare Foundation Synthesis Library	<p>Use DesignWare Foundation parts by default for optimal Synthesis QoR. This parameter can be set to false (0) if you have an RTL source license, in which case you may use source code for DesignWare Foundation Parts without the need for a DesignWare Foundation license. RTL source users, who also have a DesignWare Foundation key, may choose to retain the Foundation parts.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: True if DesignWare License is available; False if no DesignWare License is available</p> <p>Enabled: Parameter is enabled if customer has both Source and DesignWare licenses</p> <p>Parameter Name: USE_FOUNDATION</p>

4.3 AXI Deadlock Notification Configuration Parameters

Table 4-3 AXI Deadlock Notification Configuration Parameters

Label	Description
AXI Deadlock Notification Configuration	
Include Deadlock Notification?	<p>Enables deadlock notification functionality and creates the relevant I/O pins. An interrupt is generated if no read or write transactions complete within the beginning and end of a deadlock evaluation period.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: DWC-AMBA-Fabric-Source License required for enabling this feature</p> <p>Parameter Name: AXI_DLOCK_NOTIFY_EN</p>
Duration of Timeout Period	<p>Number of aclk cycles in each deadlock evaluation period.</p> <p>Values: 15, ..., 4294967295</p> <p>Default Value: 15</p> <p>Enabled: AXI_DLOCK_NOTIFY_EN == 1</p> <p>Parameter Name: AXI_DLOCK_TIMEOUT</p>

4.4 Pipelining Options / Channel Pipelining Stages Parameters

Table 4-4 Pipelining Options / Channel Pipelining Stages Parameters

Label	Description
Internal Register Slice Mode	
Read Address Channels Internal Register Slice Mode	<p>This selects the mode of the internal register slice for the read address channels. The internal register slice resides between the master and slave port within DW_axi.</p> <ul style="list-style-type: none"> ■ Combinatorial: Pass through, all paths unregistered. ■ Forward Registered: Source to sink paths registered, sink to source paths unregistered. ■ Fully Registered: All paths registered. <p>Values:</p> <ul style="list-style-type: none"> ■ Combinatorial (0) ■ Forward Registered (1) ■ Fully Registered (2) <p>Default Value: Combinatorial Enabled: AXI_INITIAL_LOCKDOWN == 0 Parameter Name: AXI_AR_TMO</p>
Write Address Channels Internal Register Slice Mode	<p>This selects the mode of the internal register slice for the write address channels. The internal register slice resides between the master and slave port within DW_axi.</p> <ul style="list-style-type: none"> ■ Combinatorial: Pass through, all paths unregistered. ■ Forward Registered: Source-to-sink paths registered, sink-to-source paths unregistered. ■ Fully Registered: All paths registered. <p>Values:</p> <ul style="list-style-type: none"> ■ Combinatorial (0) ■ Forward Registered (1) ■ Fully Registered (2) <p>Default Value: Combinatorial Enabled: AXI_INITIAL_LOCKDOWN == 0 Parameter Name: AXI_AW_TMO</p>

Table 4-4 Pipelining Options / Channel Pipelining Stages Parameters (Continued)

Label	Description
Read Data Channels Internal Register Slice Mode	<p>This selects the mode of the internal register slice for the read data channels. The internal register slice resides between the master and slave port within DW_axi.</p> <ul style="list-style-type: none"> Combinatorial: Pass through, all paths unregistered. Forward Registered: Source-to-sink paths registered, sink-to-source paths unregistered. Fully Registered: All paths registered. <p>Values:</p> <ul style="list-style-type: none"> Combinatorial (0) Forward Registered (1) Fully Registered (2) <p>Default Value: Combinatorial Enabled: AXI_INITIAL_LOCKDOWN == 0 Parameter Name: AXI_R_TMO</p>
Write Data Channels Internal Register Slice Mode	<p>This selects the mode of the internal register slice for the write data channels. The internal register slice resides between the master and slave port within DW_axi.</p> <ul style="list-style-type: none"> Combinatorial: Pass through, all paths unregistered. Forward Registered: Source to sink paths registered, sink to source paths unregistered. Fully Registered: All paths registered. <p>Values:</p> <ul style="list-style-type: none"> Combinatorial (0) Forward Registered (1) Fully Registered (2) <p>Default Value: Combinatorial Enabled: AXI_INITIAL_LOCKDOWN == 0 Parameter Name: AXI_W_TMO</p>

Table 4-4 Pipelining Options / Channel Pipelining Stages Parameters (Continued)

Label	Description
Burst Response Channels Internal Register Slice Mode	<p>This selects the mode of the internal register slice for the burst response channels. The internal register slice resides between the master and slave port within DW_axi.</p> <ul style="list-style-type: none"> Combinatorial: Pass through, all paths unregistered. Forward Registered: Source to sink paths registered, sink to source paths unregistered. Fully Registered: All paths registered. <p>Values:</p> <ul style="list-style-type: none"> Combinatorial (0) Forward Registered (1) Fully Registered (2) <p>Default Value: Combinatorial Enabled: AXI_INITIAL_LOCKDOWN == 0 Parameter Name: AXI_B_TMO</p>
Pipeline Stage After Channel Arbiter	
Add Pipeline Stage After Read Address Channel Arbiters ?	<p>This selects whether or not to add a pipeline stage after the arbiter in the read address channels. If enabled a single buffer stage is added to the channel.</p> <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_AR_PL_ARB</p>
Add Pipeline Stage After Write Address Channel Arbiters ?	<p>This selects whether or not to add a pipeline stage after the arbiter in the write address channels. If enabled a single buffer stage is added to the channel.</p> <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_AW_PL_ARB</p>
Add Pipeline Stage After Read Data Channel Arbiters ?	<p>This selects whether or not to add a pipeline stage after the arbiter in the read data channels. If enabled, a single buffer stage is added to the channel.</p> <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_R_PL_ARB</p>

Table 4-4 Pipelining Options / Channel Pipelining Stages Parameters (Continued)

Label	Description
Add Pipeline Stage After Write Data Channel Arbiters ?	<p>This selects whether or not to add a pipeline stage after the arbiter in the write data channels. If enabled, a single buffer stage is added to the channel.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_W_PL_ARB</p>
Add Pipeline Stage After Burst Response Channel Arbiters ?	<p>This selects whether or not to add a pipeline stage after the arbiter in the burst response channels. If enabled a single buffer stage is added to the channel.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_B_PL_ARB</p>

4.5 Pipelining Options / AW to W Registering Parameters

Table 4-5 Pipelining Options / AW to W Registering Parameters

Label	Description
AW to W Registering	
Register AW to W Paths	<p>All write address channel (AW) to write data channel (W) paths are registered. Where AWVALID_M(x) and WVALID_M(x) from a master assert in the same cycle for a new write transaction, the DW_axi interconnect will assert WVALID_S(j) on the next cycle, not the current one (assuming no pipeline stages exist in the channel). By setting this parameter to 0, the AW to W paths will no longer be registered and the DW_axi interconnect can assert WVALID_S(j) to a slave in the same cycle that it asserts AWVALID_S(j).</p> <p>However, setting this parameter to 0 will also reduce the operating frequency of the DW_axi instance as longer logic paths will exist.</p> <p>NOTE: In the following cases, setting AXI_REG_AW_W_PATHS to 0 will not improve latency to a particular slave:</p> <ul style="list-style-type: none"> ■ If the write interleaving depth at a slave port is > 1 (AXI_WID_S(j) > 1) and the arbiter pipeline stage for the AW channel is not enabled. ■ If a slaves write data channel has a link from the shared write data channel and pipelining is not enabled on the shared AW layer (AXI_AW_SHARED_PL=0). ■ If multi-cycle arbitration is enabled on the write data channel for a slave. ■ If slave is visible to multiple masters. <p>Values: 0, 1 Default Value: 1 Enabled: Always Parameter Name: AXI_REG_AW_W_PATHS</p>

4.6 Pipelining Options / Shared Layer Pipelining Parameters

Table 4-6 Pipelining Options / Shared Layer Pipelining Parameters

Label	Description
Shared Layer Pipelining	
Add Pipeline Stage In Shared Read Address Channel ?	<p>This parameter selects whether to add a pipeline stage to the shared read address channel. Due to the additional multiplexing in a shared channel, the combinatorial paths are longer than a dedicated channel. Enabling this parameter instantiates a forward registered pipeline stage in the shared read address channel block that shortens the critical paths, allowing high clock frequencies to be reached.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if the shared read address channel is present.</p> <p>Parameter Name: AXI_AR_SHARED_PL</p>
Add Pipeline Stage In Shared Write Address Channel ?	<p>This selects whether to add a pipeline stage to the shared write address channel. Due to the additional multiplexing in a shared channel, the combinatorial paths are longer than a dedicated channel. Enabling this parameter instantiates a forward-registered pipeline stage in the shared write address channel block that shortens the critical paths, allowing high clock frequencies to be reached.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if the shared write address channel is present.</p> <p>Parameter Name: AXI_AW_SHARED_PL</p>
Add Pipeline Stage In Shared Read Data Channel ?	<p>This selects whether to add a pipeline stage to the shared read data channel. Due to the additional multiplexing in a shared channel, the combinatorial paths are longer than a dedicated channel. Enabling this parameter instantiates a forward registered pipeline stage in the shared read data channel block that shortens the critical paths, allowing high clock frequencies to be reached.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if the shared read data channel is present.</p> <p>Parameter Name: AXI_R_SHARED_PL</p>

Table 4-6 Pipelining Options / Shared Layer Pipelining Parameters (Continued)

Label	Description
Add Pipeline Stage In Shared Write Data Channel ?	<p>This selects whether to add a pipeline stage to the shared write data channel. Due to the additional multiplexing in a shared channel, the combinatorial paths are longer than a dedicated channel. Enabling this parameter instantiates a forward registered pipeline stage in the shared write data channel block that shortens the critical paths, allowing high clock frequencies to be reached.</p> <p>Enabling this parameter also adds a single pipeline register (without buffer) which does not add latency but allows high clock frequencies to be reached. This register comes at the expense of a single cycle after (wlast & wvalid & wready) which the master and slave completing a write transaction on the shared write data channel may not access each other on the write data channel. For more information, see the "Shared Write Data Channel Pipelining" section in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if the shared write data channel is present.</p> <p>Parameter Name: AXI_W_SHARED_PL</p>
Add Pipeline Stage In Shared Burst Response Channel ?	<p>This selects whether to add a pipeline stage to the shared burst response channel. Due to the additional multiplexing in a shared channel, the combinatorial paths are longer than a dedicated channel. Enabling this parameter instantiates a forward registered pipeline stage in the shared burst response channel block that shortens the critical paths, allowing high clock frequencies to be reached.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if the shared write response channel is present.</p> <p>Parameter Name: AXI_B_SHARED_PL</p>

4.7 Pipelining Options / Multi-Cycle Arbitration Parameters

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters

Label	Description
External Priorities and Multi-Cycle Arbitration	
Register External Priorities for Multi Cycle Arbitration ?	<p>If external arbiter priorities and multi-cycle arbitration are being used, enable this parameter to force DW_axi to register the priority signals for any arbiter where multicycle arbitration has been enabled. If the external priorities are static during run time, then this option does not have to be enabled. However, if the external priorities are changing during run time, this option must be enabled for correct operation of the DW_axi interconnect.</p> <p>Because multi cycle arbitration can be configured differently at each arbiter, enabling this option infers registers for the priority signals at every arbiter that has an arbitration cycles value larger than 1. For more information about this feature, see the "Multicycle Arbitration" section, and for more information on arbitration types, see the "Arbitration Types" section in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if AXI_[AR/AW/W]_MCA_NC_S(j) or AXI_[R/B]_MCA_NC_M(x) and AXI_HAS_EXT_PRIORITY are enabled.</p> <p>Parameter Name: AXI_MCA_HLD_PRIOR</p>
Shared Slave Port	
Number of Arbitration Cycles for the shared Read Address Channel.	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the shared Read Address channel.</p> <p>If a value larger than 1 is selected, then multicycle arbitration is enabled in the shared Read Address channel.</p> <p>Enabling this feature instantiates logic around the arbiter in the shared Read Address channel such that paths through the arbiter can be specified as multicycle paths. This can help to allow the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first come first served or fair among equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_AR_SHARED_MCA_NC</p>

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters (Continued)

Label	Description
Number of Arbitration Cycles for the shared Write Address Channel.	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the shared Write Address channel.</p> <p>If a value larger than 1 is selected, then multicycle arbitration is enabled in the shared Write Address channel.</p> <p>Enabling this feature instantiates logic around the arbiter in the shared Write Address channel such that paths through the arbiter can be specified as multicycle paths. This can help to allow the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first come first served or fair among equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>Values: 1, ..., 16 Default Value: 1 Enabled: Always Parameter Name: AXI_AW_SHARED_MCA_NC</p>
Number of Arbitration Cycles for the shared Write Data Channel.	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the shared Write Data channel.</p> <p>If a value larger than 1 is selected, then multicycle arbitration is enabled in the shared Write Data channel.</p> <p>Enabling this feature instantiates logic around the arbiter in the shared Write Data channel such that paths through the arbiter can be specified as multicycle paths. This can help to allow the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first come first served or fair among equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>Values: 1, ..., 16 Default Value: 1 Enabled: Always Parameter Name: AXI_W_SHARED_MCA_NC</p>

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters (Continued)

Label	Description
Slave Port j	
Number of Arbitration Cycles for Read Address Channel in Slave Port j (for j = 0; j <= AXI_NUM_SLAVES)	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the Read Address channel of slave port j.</p> <p>If a value larger than 1 is selected, then multi cycle arbitration is enabled in the Read Address channel for slave port j.</p> <p>Enabling this feature instantiates logic around the arbiter in the Read Address channel of slave port j such that paths through the arbiter can be specified as multi cycle paths. This helps the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first-come-first-served or fair-among-equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>For more information, see the "Multicycle Arbitration" section and for information on arbitration types, see the "Arbitration Types" section in the DW_axi Databook.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: If AXI_HAS_LOCKING=1, multicycle arbitration cannot be used on the address channels; that is, AXI_AR_MCA_NC_S(j) must be equal to 0.</p> <p>Parameter Name: AXI_AR_MCA_NC_S(j)</p>
Number of Arbitration Cycles for Write Address Channel in Slave Port j (for j = 0; j <= AXI_NUM_SLAVES)	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the Write Address channel of slave port j.</p> <p>If a value larger than 1 is selected, then multi cycle arbitration is enabled in the Write Address channel for slave port j.</p> <p>Enabling this feature instantiates logic around the arbiter in the Write Address channel of slave port j such that paths through the arbiter can be specified as multi cycle paths. This helps the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first-come-first-served or fair-among-equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>For more information, see the "Multicycle Arbitration" section and for information on arbitration types, see the "Arbitration Types" section in the DW_axi Databook.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: If AXI_HAS_LOCKING=1, multicycle arbitration cannot be used on the address channels; that is, AXI_AW_MCA_NC_S(j) must be equal to 0.</p> <p>Parameter Name: AXI_AW_MCA_NC_S(j)</p>

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters (Continued)

Label	Description
Number of Arbitration Cycles for Write Data Channel in Slave Port j (for j = 0; j <= AXI_NUM_SLAVES)	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the Write Data channel of slave port j.</p> <p>If a value larger than 1 is selected, then multi cycle arbitration is enabled in the Write Data channel for slave port j.</p> <p>Enabling this feature instantiates logic around the arbiter in the Write Data channel of slave port j such that paths through the arbiter can be specified as multi cycle paths. This helps the DW_axi to be synthesized to high frequencies by allowing the channel arbiters to be removed from the worst logic paths.</p> <p>This option allows complex arbitration schemes such as first-come-first-served or fair-among-equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>For more information, see the "Multicycle Arbitration" section and for information on arbitration types, see the "Arbitration Types" section in the DW_axi Databook.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: If AXI_WID_S(j) = 1, then AXI_W_MCA_NC_S(j) is disabled. If there is a write interleaving depth of 1 at a slave port write data channel, there is no arbiter inferred at that point. Logic that applies AXI write data ordering rules allows only one master at a time to send write data to the slave, in address order.</p> <p>Parameter Name: AXI_W_MCA_NC_S(j)</p>
Shared Master Port	
Number of Arbitration Cycles for the shared Read Data Channel	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the shared Read Data channel.</p> <p>If a value larger than 1 is selected then multicycle arbitration is enabled in the shared Read Data channel.</p> <p>Enabling this feature instantiates logic around the arbiter in the shared Read Data channel such that paths through the arbiter can be specified as multicycle paths. This can help to ease timing congestion in the configuration by easing the timing requirements through large arbiters.</p> <p>This option allows complex arbitration schemes such as first-come-first-served or fair-among-equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_R_SHARED_MCA_NC</p>

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters (Continued)

Label	Description
Number of Arbitration Cycles for the shared Burst Response Channel	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the shared Burst Response channel.</p> <p>If a value larger than 1 is selected then multicycle arbitration is enabled in the shared Burst Response channel.</p> <p>Enabling this feature instantiates logic around the arbiter in the shared Burst Response channel such that paths through the arbiter can be specified as multicycle paths. This can help to ease timing congestion in the configuration by easing the timing requirements through large arbiters.</p> <p>This option allows complex arbitration schemes such as first-come-first-served or fair-among-equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>Values: 1, ..., 16 Default Value: 1 Enabled: Always Parameter Name: AXI_B_SHARED_MCA_NC</p>
Master Port x	
Number of Arbitration Cycles for Read Data Channel in Master Port x (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the Read Data channel of master port x.</p> <p>If a value larger than 1 is selected, then multicycle arbitration will be enabled in the Read Data channel for master port x.</p> <p>Enabling this feature instantiates logic around the arbiter in the Read Data channel of master port x such that paths through the arbiter can be specified as multicycle paths. This can help to ease timing congestion in the configuration by easing the timing requirements through large arbiters.</p> <p>This option allows complex arbitration schemes such as first come first served or fair among equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>For more information, see section "Multicycle Arbitration" and for more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values: 1, ..., 16 Default Value: 1 Enabled: Always Parameter Name: AXI_R_MCA_NC_M(x)</p>

Table 4-7 Pipelining Options / Multi-Cycle Arbitration Parameters (Continued)

Label	Description
Number of Arbitration Cycles for Burst Response Channel in Master Port x (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the number of cycles over which arbitration is performed for the arbiter in the Burst Response channel of master port x.</p> <p>If a value larger than 1 is selected, then multicycle arbitration will be enabled in the Burst Response channel for master port x.</p> <p>Enabling this feature instantiates logic around the arbiter in the Burst Response channel of master port x such that paths through the arbiter can be specified as multicycle paths. This can help to ease timing congestion in the configuration by easing the timing requirements through large arbiters.</p> <p>This option allows complex arbitration schemes such as first come first served or fair among equals to be used with large client numbers without negatively affecting the maximum clock frequency of the configuration.</p> <p>For more information, see section "Multicycle Arbitration" and for more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values: 1, ..., 16</p> <p>Default Value: 1</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_B_MCA_NC_M(x)</p>

4.8 Mult-Tile/Bi-directional Command / System Master Numbers Parameters

Table 4-8 Mult-Tile/Bi-directional Command / System Master Numbers Parameters

Label	Description
System Number	
Master Numbers (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$)	<p>Assigns to each direct master port a unique system number. For more information, see section "System Masters" in the DW_axi Databook.</p> <p>Values: 1, ..., AXI_NUM_SYS_MASTERS</p> <p>Default Value: Value of AXI_NUM_SYS_MASTERS parameter</p> <p>Enabled: AXI_HAS_BICMD parameter must be enabled</p> <p>Parameter Name: AXI_SYS_NUM_FOR_M(x)</p>

4.9 Mult-Tile/Bi-directional Command / Interconnecting Slave Ports Parameters

Table 4-9 Mult-Tile/Bi-directional Command / Interconnecting Slave Ports Parameters

Label	Description
Connected to ICM of Different DW_axi ?	
Does S(j) Access Non Local System Slaves? (for j = 1; j <= AXI_NUM_SLAVES)	<p>Used to prevent a deadlock scenario in multi-tile DW_axi systems. Set to 1 for every slave port which will access a non local slave, i.e. the slave port is not connected directly to a system slave, but is connected to the master port of a different DW_axi instance.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Enabled if ((AXI_HAS_BICMD = 1) OR (AXI_EN_MULTI_TILE_DLOCK_AVOID = 1)) AND (AXI_NUM_SLAVES >= j)</p> <p>Parameter Name: AXI_ACC_NON_LCL_SLV_S(j)</p>

4.10 Mult-Tile/Bi-directional Command / Interconnecting Master Port configuration Parameters

Table 4-10 Mult-Tile/Bi-directional Command / Interconnecting Master Port configuration Parameters

Label	Description
Number of System Masters	
Number of System Masters that can access the interconnecting Master port x (for x = 1; x <= AXI_NUM_ICM)	<p>Configures interconnecting master ports to allow non-local system masters to forward their transactions through the port. This parameter is not required if there is only one interconnecting master port. For more information, see section "Interconnecting Masters" in the DW_axi Databook.</p> <p>Values: 1, ..., 8</p> <p>Default Value: 1</p> <p>Enabled: Determined by AXI_HAS_BICMD parameter</p> <p>Parameter Name: AXI_NUM_MST_THRU_ICM(x)</p>
1st Non-Local System number	
Nth Non-Local System number (for x,n = 1; x,n <= AXI_NUM_ICM,AXI_NUM_MST_THRU_ICM(x))	<p>Set a System Master number that can forward transactions through interconnecting Master Port (x) in Normal mode</p> <p>Values: 1, ..., AXI_NUM_SYS_MASTERS</p> <p>Default Value: 1</p> <p>Enabled: Determined by AXI_HAS_BICMD parameter</p> <p>Parameter Name: AXI_ALLOW_MST(n)_ICM(x)</p>

4.11 Arbitration Options / Arbitration Type Parameters

Table 4-11 Arbitration Options / Arbitration Type Parameters

Label	Description
Slave Port j	
Arbiter Type for Read Address channel in Slave Port (j) (for j = 1; j <= AXI_NUM_SLAVES)	<p>Selects the type of arbiter to be used in the Read Address channel at slave port j.</p> <ul style="list-style-type: none"> 0 - Priority; Highest priority wins and the lowest numbered master among masters with the same priority wins. 1 - First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. 2 - Fair Among Equals: Two tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle by cycle basis. 3 - User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. 4 - QOSArbiter: Instantiates a dynamic priority arbiter for QoS support. Arbitration at Slave Port. Priority of request is determined by corresponding awqos/arqos signal values. <p>For more information about arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Dependencies: This parameter is enabled if (AXI_NUM_SLAVES >= j). And also it has following dependencies:</p> <ul style="list-style-type: none"> AXI_HAS_QOS = 1 The QoS Arbiter is available for selection on the Read Address channels. AXI_HAS_LOCKING = 1 The external (user-defined) arbiter cannot be used on Read Address channels. <p>Values:</p> <ul style="list-style-type: none"> Priority (0) First Come First Serve (1) Fair Among Equals (2) User Defined (3) QOSArbiter (4) <p>Default Value: Priority</p> <p>Enabled: See Description</p> <p>Parameter Name: AXI_AR_ARB_TYPE_S(j)</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Arbiter Type for Write Address channel in Slave Port (j) (for j = 1; j <= AXI_NUM_SLAVES)	<p>Selects the type of arbiter to be used in the Write Address channel at slave port j.</p> <ul style="list-style-type: none"> ■ 0 - Priority; Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ 1 - First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ 2 - Fair Among Equals: Two tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle by cycle basis. ■ 3 - User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. ■ 4 - QOSArbiter: Instantiates a dynamic priority arbiter for QoS support. Arbitration at Slave Port. Priority of request is determined by corresponding awqos/arqos signal values. <p>For more information about arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Dependencies: This parameter is enabled if (AXI_NUM_SLAVES >= j). And also it has following dependencies:</p> <ul style="list-style-type: none"> ■ AXI_HAS_QOS = 1 The QoS Arbiter is available for selection on the Write Address channels. ■ AXI_HAS_LOCKING = 1 The external (user-defined) arbiter cannot be used on Write Address channels. <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) ■ QOSArbiter (4) <p>Default Value: Priority</p> <p>Enabled: See Description</p> <p>Parameter Name: AXI_AW_ARB_TYPE_S(j)</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Arbiter Type for Write Data channel in Slave Port (j) (for j = 1; j <= AXI_NUM_SLAVES)	<p>Selects the type of arbiter to be used in the Write Data channel at slave port j.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they are requested, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_NUM_SLAVES >= j</p> <p>Parameter Name: AXI_W_ARB_TYPE_S(j)</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Master Port x	
Arbiter Type for Read Data channel in Master Port (x) (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the type of arbiter to be used in the Read Data channel at master port (x).</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered slave among slaves with the same priority wins. ■ First Come First Serve: Slaves granted in the order that they request, longest waiting slave has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between slaves of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_NUM_MASTERS >= x</p> <p>Parameter Name: AXI_R_ARB_TYPE_M(x)</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Arbiter Type for Burst Response channel in Master Port (x) (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the type of arbiter to be used in the Burst Response channel at master port (x).</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered slave among slaves with the same priority wins. ■ First Come First Serve: Slaves granted in the order that they request, longest waiting slave has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between slaves of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_NUM_MASTERS >= x</p> <p>Parameter Name: AXI_B_ARB_TYPE_M(x)</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Write Address Shared Layer	
Shared Layer Arbiters, Arbiter Type for Shared Write Address Channel	<p>Selects the type of arbiter to be used in the shared Write Address channel.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. ■ QOSArbiter: Instantiates a dynamic priority arbiter for QoS support. Priority of request is determined by the corresponding awqos/arqos signal value. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Dependencies:</p> <ul style="list-style-type: none"> ■ AXI_HAS_QOS = 1 The QoS Arbiter is available for selection on the Write Address channels. ■ AXI_HAS_LOCKING = 1 The external (user-defined) arbiter cannot be used on Write Address channels. <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) ■ QOSArbiter (4) <p>Default Value: Priority</p> <p>Enabled: See Description</p> <p>Parameter Name: AXI_AW_SHARED_ARB_TYPE</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Write Data Shared Layer	
Shared Layer Arbiters, Arbiter Type for Shared Write Data Channel	<p>Selects the type of arbiter to be used in the shared Write Data channel.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_W_HAS_SHARED_LAYER == 1</p> <p>Parameter Name: AXI_W_SHARED_ARB_TYPE</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Burst Response Shared Layer	
Shared Layer Arbiters, Arbiter Type for Shared Burst Response Channel	<p>Selects the type of arbiter to be used in the shared Burst Response channel.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_B_HAS_SHARED_LAYER == 1</p> <p>Parameter Name: AXI_B_SHARED_ARB_TYPE</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Read Address Shared Layer	
Shared Layer Arbiters, Arbiter Type for Shared Read Address Channel	<p>Selects the type of arbiter to be used in the shared Read Address channel.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. ■ QOSArbiter: Instantiates a dynamic priority arbiter for QoS support. Priority of request is determined by the corresponding awqos/arqos signal value. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Dependencies:</p> <ul style="list-style-type: none"> ■ AXI_HAS_QOS = 1 The QoS Arbiter is available for selection on the Read Address channels. ■ AXI_HAS_LOCKING = 1 The external (user-defined) arbiter cannot be used on Read Address channels. <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) ■ QOSArbiter (4) <p>Default Value: Priority</p> <p>Enabled: See Description</p> <p>Parameter Name: AXI_AR_SHARED_ARB_TYPE</p>

Table 4-11 Arbitration Options / Arbitration Type Parameters (Continued)

Label	Description
Read Data Shared Layer	
Shared Layer Arbiters, Arbiter Type for Shared Read Data Channel	<p>Selects the type of arbiter to be used in the shared Read Data channel.</p> <ul style="list-style-type: none"> ■ Priority: Highest priority wins and the lowest numbered master among masters with the same priority wins. ■ First Come First Serve: Masters granted in the order that they request, longest waiting master has the highest priority. ■ Fair Among Equals: Two-tier arbitration. First tier is dynamic priority, second tier shares grants equally between masters of the same highest requesting priority on a cycle-by-cycle basis. ■ User Defined: Instantiates a plain text arbitration module which the user can edit to their own requirements within the guidelines outlined in the DW_axi Databook. By default, this plain text module instantiates a dynamic priority arbiter. <p>For more information on arbitration types, see section "Arbitration Types" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Priority (0) ■ First Come First Serve (1) ■ Fair Among Equals (2) ■ User Defined (3) <p>Default Value: Priority</p> <p>Enabled: AXI_R_HAS_SHARED_LAYER == 1</p> <p>Parameter Name: AXI_R_SHARED_ARB_TYPE</p>

4.12 Arbitration Options / Master and Slaves Priority Parameters

Table 4-12 Arbitration Options / Master and Slaves Priority Parameters

Label	Description
External Priority	
Enable External Priority Support?	<p>/When this parameter is set to True, an input port for each AXI master and slave port (mst_priority_m* and slv_priority_s*, respectively) is included on the I/O and the arbitration priorities can be programmed during operation of the component. When set to False, the the External Priority signals are not included on the I/O and AXI master and slave port priorities are fixed. For more information on external priority signals, see section "External Priorities" in the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_EXT_PRIORITY</p>
Shared Layer Priority	
Priority for Shared Layer Master	<p>Sets the priority of the shared layer read address, write address, and write data channels when they request access at a dedicated layer. Shared layer master priority indicates that on these channels the shared layer is requesting on behalf of the masters. For more information on external priority signals, see section "External Priorities" in the DW_axi Databook.</p> <p>Values: 1, ..., AXI_NUM_MASTERS</p> <p>Default Value: 1</p> <p>Enabled: AXI_SHARED_LAYER_MASTER_PRIORITY_EN_VAL == 1</p> <p>Parameter Name: AXI_SHARED_LAYER_MASTER_PRIORITY</p>
Priority for Shared Layer Slave	<p>Sets the priority of the shared layer read data and burst response channels when they request access at a dedicated layer. Shared layer slave priority indicates that, on these channels, the shared layer is requesting on behalf of slaves. For more information on external priority signals, see section "External Priorities" in the DW_axi Databook.</p> <p>Values: 1, ..., AXI_NUM_SLAVES</p> <p>Default Value: 1</p> <p>Enabled: AXI_SHARED_LAYER_SLAVE_PRIORITY_EN_VAL == 1</p> <p>Parameter Name: AXI_SHARED_LAYER_SLAVE_PRIORITY</p>

Table 4-12 Arbitration Options / Master and Slaves Priority Parameters (Continued)

Label	Description
Priority	
Priority for Master (x) (for x = 1; x <= AXI_NUM_MASTERS)	<p>Arbitration priority associated with Master(x). The same priority is used for write address, read address, and write data channel arbitrations. The larger numbers represent higher priority assignments, so priority n has a higher priority than n-1.</p> <p>Values: 0, ..., [::DW_axi::calcMaxValue AXI_NUM_MASTERS]</p> <p>Default Value: 0</p> <p>Enabled: These parameters are not available when AXI_HAS_EXT_PRIORITY is set to True.</p> <p>Parameter Name: AXI_PRIORITY_M(x)</p>
Priority for Slave(j) (for j = 1; j <= AXI_NUM_SLAVES)	<p>Arbitration priority associated with the Slave (j). The same priority is used for write response and read data arbitration. The larger numbers represent higher priority assignments, so priority n has a higher priority than n-1. Arbitration priority value 0 is reserved for the default slave.</p> <p>Values: 1, ..., AXI_NUM_SLAVES</p> <p>Default Value: 1</p> <p>Enabled: These parameters are not available when AXI_HAS_EXT_PRIORITY is set to True.</p> <p>Parameter Name: AXI_PRIORITY_S(j)</p>

4.13 Sideband Signals Parameters

Table 4-13 Sideband Signals Parameters

Label	Description
HasSideBand	
Include Sideband/User Bus for Write Address Channels?	<p>If set to True, then all master and slave write address channels have an associated sideband/user bus. The write address channel sideband/user bus is routed in the same way as the write address channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_HAS_AWSB</p>
Included Sideband Bus for Write Data Channels?	<p>If set to True, then all master and slave write data channels have an associated sideband/user bus. The write data channel sideband bus is routed in the same way as the write data channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_HAS_WSB</p>
Include Sideband Bus for Write Response Channels?	<p>If set to True, then all master and slave write response channels have an associated sideband/user bus. The write response channel sideband/user bus is routed in the same way as the write response channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_HAS_BSB</p>
Included Sideband Bus for Read Address Channels?	<p>If set to True, then all master and slave read address channels have an associated sideband/user bus. The read address channel sideband/user bus is routed in the same way as the read address channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: Always Parameter Name: AXI_HAS_ARSB</p>

Table 4-13 Sideband Signals Parameters (Continued)

Label	Description
Include Sideband Bus for Read Data Channels?	<p>If set to True, then all master and slave read data channels have an associated sideband/user bus. The read data channel sideband/user bus is routed in the same way as the read data channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_RSB</p>
Width	
Width of the Write Address Channel Sideband bus	<p>When the AXI_HAS_AWSB parameter is set to True, you can set the write address channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_AWSB == 1</p> <p>Parameter Name: AXI_AW_SBW</p>
Width of the Write Data Channel Sideband bus	<p>When the AXI_HAS_WSB parameter is set to True, you can set the write data channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_WSB == 1</p> <p>Parameter Name: AXI_W_SBW</p>
Width of the Write Response Channel Sideband bus	<p>When the AXI_HAS_SBW parameter is set to True, you can set the write response channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_BSB == 1</p> <p>Parameter Name: AXI_B_SBW</p>
Width of the Read Address Channel Sideband bus	<p>When the AXI_HAS_ARSB parameter is set to True, you can set the read address channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_ARSB == 1</p> <p>Parameter Name: AXI_AR_SBW</p>
Width of the Read Data Channel Sideband bus	<p>When the AXI_HAS_RSB parameter is set to True, you can set the read data channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_RSB == 1</p> <p>Parameter Name: AXI_R_SBW</p>

4.14 User Signals Parameters

Table 4-14 User Signals Parameters

Label	Description
HasSideBand	
Include Sideband/User Bus for Write Address Channels?	<p>If set to True, then all master and slave write address channels have an associated sideband/user bus. The write address channel sideband/user bus is routed in the same way as the write address channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_AWSB</p>
Included Sideband Bus for Write Data Channels?	<p>If set to True, then all master and slave write data channels have an associated sideband/user bus. The write data channel sideband bus is routed in the same way as the write data channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_WSB</p>
Include Sideband Bus for Write Response Channels?	<p>If set to True, then all master and slave write response channels have an associated sideband/user bus. The write response channel sideband/user bus is routed in the same way as the write response channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_BSB</p>
Included Sideband Bus for Read Address Channels?	<p>If set to True, then all master and slave read address channels have an associated sideband/user bus. The read address channel sideband/user bus is routed in the same way as the read address channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_ARSB</p>

Table 4-14 User Signals Parameters (Continued)

Label	Description
Include Sideband Bus for Read Data Channels?	<p>If set to True, then all master and slave read data channels have an associated sideband/user bus. The read data channel sideband/user bus is routed in the same way as the read data channel payload.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: Always</p> <p>Parameter Name: AXI_HAS_RSB</p>
Width	
Width of the Write Address Channel Sideband bus	<p>When the AXI_HAS_AWSB parameter is set to True, you can set the write address channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_AWSB == 1</p> <p>Parameter Name: AXI_AW_SBW</p>
Width of the Write Data Channel Sideband bus	<p>When the AXI_HAS_WSB parameter is set to True, you can set the write data channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_WSB == 1</p> <p>Parameter Name: AXI_W_SBW</p>
Width of the Write Response Channel Sideband bus	<p>When the AXI_HAS_SBW parameter is set to True, you can set the write response channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_BSB == 1</p> <p>Parameter Name: AXI_B_SBW</p>
Width of the Read Address Channel Sideband bus	<p>When the AXI_HAS_ARSB parameter is set to True, you can set the read address channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_ARSB == 1</p> <p>Parameter Name: AXI_AR_SBW</p>
Width of the Read Data Channel Sideband bus	<p>When the AXI_HAS_RSB parameter is set to True, you can set the read data channel sideband/user bus width.</p> <p>Values: 1, ..., AXI_MAX_SBW</p> <p>Default Value: 1</p> <p>Enabled: AXI_HAS_RSB == 1</p> <p>Parameter Name: AXI_R_SBW</p>

4.15 Slave Visibility Parameters

Table 4-15 Slave Visibility Parameters

Label	Description
Master 1	
Normal Mode Visibility of Slave j by Master x (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>This selects whether Slave j is visible by Master x in Normal Mode.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: ((AXI_NUM_SLAVES >= j) AND (AXI_NUM_MASTERS >= x))</p> <p>Enabled: ((AXI_NUM_SLAVES >= j) AND (AXI_NUM_MASTERS >= x))</p> <p>Parameter Name: AXI_NV_S(j)_BY_M(x)</p>
Boot Mode Visibility of Slave j by Master x (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>This selects whether Slave j is visible by Master x in Boot Mode.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: ((AXI_NUM_SLAVES >= j) AND (AXI_NUM_MASTERS >= x) AND (AXI_REMAP_EN == 1))</p> <p>Enabled: ((AXI_NUM_SLAVES >= j) AND (AXI_NUM_MASTERS >= x) AND (AXI_REMAP_EN == 1))</p> <p>Parameter Name: AXI_BV_S(j)_BY_M(x)</p>

4.16 Architecture Options / Architecture Options Parameters

Table 4-16 Architecture Options / Architecture Options Parameters

Label	Description
AW Channel Quick Configure	
Share AW Channel For All Links ?	<p>Choose to make the write address channel shared to all master slave links by default. This parameter is enabled if the presence of a DWC AMBA Fabric package license is detected. This parameter is disabled if locking sequences are enabled (AXI_HAS_LOCKING == 1).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_ALL_AW_LAYER_SHARED</p>
Slave Port j	
Write Address channel Architecture (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>Connects the write address channel between Master (x) and Slave (j) using either</p> <ul style="list-style-type: none"> ■ Dedicated layer : Recommended if the link has a high bandwidth OR low latency requirement ■ Shared layer: Recommended if the links has low bandwidth AND relaxed latency requirement <p>For more information about dedicated and shared layers, refer to "Address and Data bus architectures" section.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Dedicated (0) ■ Shared (1) <p>Default Value: AXI_ALL_AW_LAYER_SHARED AND AXI_VV_S(j)_BY_M(x)</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_AW_LAYER_S(j)_M(x)</p>

Table 4-16 Architecture Options / Architecture Options Parameters (Continued)

Label	Description
W Channel Quick Configure	
Share W Channel For All Links ?	<p>Choose to make the write data channel shared to all master slave links by default. This parameter is enabled if the presence of a DWC AMBA Fabric package license is detected. This parameter is disabled if locking sequences are enabled (AXI_HAS_LOCKING == 1).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_ALL_W_LAYER_SHARED</p>
Slave Port j	
Write Data channel Architecture (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>Connects the write data channel between Master (x) and Slave (j) using either</p> <ul style="list-style-type: none"> ■ Dedicated layer : Recommended if the link has a high bandwidth OR low latency requirement ■ Shared layer: Recommended if the links has low bandwidth AND relaxed latency requirement <p>For more information about dedicated and shared layers, refer to "Address and Data bus architectures" section.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Dedicated (0) ■ Shared (1) <p>Default Value: AXI_ALL_W_LAYER_SHARED AND AXI_VV_S(j)_BY_M(x)</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_W_LAYER_S(j)_M(x)</p>
B Channel Quick Configure	
Share B Channel For All Links ?	<p>This parameter makes the burst response channel shared to all master slave links by default. This parameter is enabled if the presence of a DWC AMBA Fabric package license is detected. This parameter is disabled if locking sequences are enabled (AXI_HAS_LOCKING == 1).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_ALL_B_LAYER_SHARED</p>

Table 4-16 Architecture Options / Architecture Options Parameters (Continued)

Label	Description
Master Port x	
Burst Response channel Architecture (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>Connects the burst response channel between Master (x) and Slave (j) using either</p> <ul style="list-style-type: none"> ■ Dedicated layer : Recommended if the link has a high bandwidth OR low latency requirement ■ Shared layer: Recommended if the links has low bandwidth AND relaxed latency requirement <p>For more information about dedicated and shared layers, refer to "Address and Data bus architectures" section.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Dedicated (0) ■ Shared (1) <p>Default Value: AXI_ALL_B_LAYER_SHARED AND AXI_VV_S(j)_BY_M(x)</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_B_LAYER_S(j)_M(x)</p>
AR Channel Quick Configure	
Share AR Channel For All Links ?	<p>Choose to make the read address channel shared to all master slave links by default. This parameter is enabled if the presence of a DWC AMBA Fabric package license is detected. This parameter is disabled if locking sequences are enabled (AXI_HAS_LOCKING == 1).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_ALL_AR_LAYER_SHARED</p>

Table 4-16 Architecture Options / Architecture Options Parameters (Continued)

Label	Description
Slave Port j	
Read Address channel Architecture (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>Connects the read address channel between Master (x) and Slave (j) using either</p> <ul style="list-style-type: none"> ■ Dedicated layer : Recommended if the link has a high bandwidth OR low latency requirement ■ Shared layer: Recommended if the links has low bandwidth AND relaxed latency requirement <p>For more information about dedicated and shared layers, refer to "Address and Data bus architectures" section.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Dedicated (0) ■ Shared (1) <p>Default Value: AXI_ALL_AR_LAYER_SHARED AND AXI_VV_S(j)_BY_M(x)</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_AR_LAYER_S(j)_M(x)</p>
R Channel Quick Configure	
Share R Channel For All Links ?	<p>This parameter makes the read data channel shared to all master slave links by default. This parameter is enabled if the presence of a DWC AMBA Fabric package license is detected. This parameter is disabled if locking sequences are enabled (AXI_HAS_LOCKING == 1).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_ALL_R_LAYER_SHARED</p>

Table 4-16 Architecture Options / Architecture Options Parameters (Continued)

Label	Description
Master Port x	
Read Data channel Architecture (for j,x = 1; j,x <= AXI_NUM_SLAVES,AXI_NUM_MASTERS)	<p>Connects the read data channel between Master (x) and Slave (j) using either</p> <ul style="list-style-type: none"> ■ Dedicated layer : Recommended if the link has a high bandwidth OR low latency requirement ■ Shared layer: Recommended if the links has low bandwidth AND relaxed latency requirement <p>For more information about dedicated and shared layers, refer to "Address and Data bus architectures" section.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Dedicated (0) ■ Shared (1) <p>Default Value: AXI_ALL_R_LAYER_SHARED AND AXI_VV_S(j)_BY_M(x)</p> <p>Enabled: This parameter is enabled if DWC-AMBA-Fabric-Source License is detected and is disabled if AXI_HAS_LOCKING=1</p> <p>Parameter Name: AXI_R_LAYER_S(j)_M(x)</p>

4.17 Slave Address Map Parameters

Table 4-17 Slave Address Map Parameters

Label	Description
Include REGION signals on Slave Port?	
Include Region Signals on Slave Port (j)? (for j = 1; j <= AXI_NUM_SLAVES)	<p>If set to true, Write Address channels and Read address channels for Slave Port (j) have region signal on the I/O; that is awregion_s(j)/arregion_s(j).</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: (AXI_INTERFACE_TYPE = AXI4/ACELITE) AND (AXI_NUM_SLAVES >= j)</p> <p>Parameter Name: AXI_HAS_REGIONS_S(j)</p>
Slave 1	
Number of Regions in Normal for Slave j (for j = 1; j <= AXI_NUM_SLAVES)	<p>This parameter specifies the number of address regions for Slave (j) in Normal mode. This parameter is active if any master has visibility of this slave in Normal mode. If no master has visibility of this slave in Normal mode, then this parameter is inactive.</p> <p>Values: 1, ..., 8</p> <p>Default Value: 1</p> <p>Enabled: This parameter is active when AXI_NV_S(j)_BY_ANY_M is set to True and AXI_REMAP_EN is set to True</p> <p>Parameter Name: AXI_NUM_RN_S(j)</p>

Table 4-17 Slave Address Map Parameters (Continued)

Label	Description
Normal Start Address for Slave(j) (Region i) (for i,j = 1; i,j <= 8, AXI_NUM_SLAVES)	<p>This parameter specifies the Start address region for a slave (j) in Normal mode. The regions can be separate or contiguous blocks. If the parameter AXI_NV_S(j)_BY_ANY_M is set to False, then this parameter is invalid for all of i (where i=1 to 8). Otherwise, this parameter is valid for all: i <= AXI_NUM_RN_S(j)</p> <p>For Region 1 to 8, the default value is:</p> <ul style="list-style-type: none"> ■ Region 1: 0x2000000 to 0x2600000 ■ Region 2: 0x3000000 to 0x2700000 ■ Region 3: 0x40100000 to 0x4000000 ■ Region 4: 0x50100000 to 0x5000000 ■ Region 5: 0x60100000 to 0x6000000 ■ Region 6: 0x70100000 to 0x7000000 ■ Region 7: 0x80100000 to 0x8000000 ■ Region 8: 0x90100000 to 0x9000000 <p><i>Note:</i> Default values for regions 1 to 8 are mentioned for AXI bus width (AXI_AW) of 32 bits</p> <p>Values: 0x0, ..., 0xffffffff</p> <p>Default Value: See Description</p> <p>Enabled: There must be at least one slave in the system and at least one master must have access to Slave 1; additionally, AXI_HAS_XCDR=0</p> <p>Parameter Name: AXI_R(i)_NSA_S(j)</p>

Table 4-17 Slave Address Map Parameters (Continued)

Label	Description
Normal End Address for Slave(j) (Region i) (for i, j = 1; i, j ≤ 8, AXI_NUM_SLAVES)	<p>This parameter specifies the End address region for a slave (j) in normal mode. The regions can be separate or contiguous blocks. If the parameter AXI_NV_S(j)_BY_ANY_M is set to false, then this parameter is invalid for all of i (where i=1 to 8). Otherwise, this parameter is valid for all: i ≤ AXI_NUM_RB_S(j)</p> <p>For Region 1 to 8, the default value is:</p> <ul style="list-style-type: none"> ■ Region 1: 0x200ffff to 0x260ffff ■ Region 2: 0x300ffff to 0x270ffff ■ Region 3: 0x4010ffff to 0x400ffff ■ Region 4: 0x5010ffff to 0x500ffff ■ Region 5: 0x6010ffff to 0x600ffff ■ Region 6: 0x7010ffff to 0x700ffff ■ Region 7: 0x8010ffff to 0x800ffff ■ Region 8: 0x9010ffff to 0x900ffff <p>Note: Default values for regions 1 to 8 are mentioned for AXI bus width (AXI_AW) of 32 bits</p> <p>Values: 0x0, ..., 0xffffffff</p> <p>Default Value: See Description</p> <p>Enabled: There must be at least one slave in the system and at least one master must have access to Slave 1; additionally, AXI_HAS_XCDR=0</p> <p>Parameter Name: AXI_R(i)_NEA_S(j)</p>
Number of Regions in Boot for Slave j (for j = 1; j ≤ AXI_NUM_SLAVES)	<p>This parameter specifies the number of address regions for Slave (j) in Boot mode. This parameter is active if any master has visibility of this slave in Boot mode. If no master has visibility of this slave in Boot mode, then this parameter is inactive.</p> <p>Values: 1, ..., 8</p> <p>Default Value: 1</p> <p>Enabled: This parameter is active when AXI_BV_S(j)_BY_ANY_M is set to True and AXI_REMAP_EN is set to True</p> <p>Parameter Name: AXI_NUM_RB_S(j)</p>

Table 4-17 Slave Address Map Parameters (Continued)

Label	Description
Boot Start Address for Slave(j) (Region i) (for i,j = 1; i,j <= 8, AXI_NUM_SLAVES)	<p>This parameter specifies the Start address region for a slave (j) in Boot mode. The regions can be separate or contiguous blocks. If the parameter AXI_NV_S(j)_BY_ANY_M is set to False, then this parameter is invalid for all of i (where i=1 to 8). Otherwise, this parameter is valid for all: i <= AXI_NUM_RN_S(j)</p> <p>For Region 1 to 8, the default value is:</p> <ul style="list-style-type: none"> ■ Region 1: 0x2000000 to 0x2600000 ■ Region 2: 0x3000000 to 0x2700000 ■ Region 3: 0x40100000 to 0x4000000 ■ Region 4: 0x50100000 to 0x5000000 ■ Region 5: 0x60100000 to 0x6000000 ■ Region 6: 0x70100000 to 0x7000000 ■ Region 7: 0x80100000 to 0x8000000 ■ Region 8: 0x90100000 to 0x9000000 <p><i>Note:</i> Default values for regions 1 to 8 are mentioned for AXI bus width (AXI_AW) of 32 bits</p> <p>Values: 0x0, ..., 0xffffffff</p> <p>Default Value: See Description</p> <p>Enabled: There must be at least one slave in the system and at least one master must have access to Slave 1; additionally, AXI_HAS_XCDR=0</p> <p>Parameter Name: AXI_R(i)_BSA_S(j)</p>

Table 4-17 Slave Address Map Parameters (Continued)

Label	Description
Boot End Address for Slave(j) (Region i) (for i,j = 1; i,j <= 8, AXI_NUM_SLAVES)	<p>This parameter specifies the End address region for a slave (j) in normal mode. The regions can be separate or contiguous blocks. If the parameter AXI_NV_S(j)_BY_ANY_M is set to false, then this parameter is invalid for all of i (where i=1 to 8). Otherwise, this parameter is valid for all: i <= AXI_NUM_RB_S(j)</p> <p>For Region 1 to 8, the default value is:</p> <ul style="list-style-type: none"> ■ Region 1: 0x200ffff to 0x260ffff ■ Region 2: 0x300ffff to 0x270ffff ■ Region 3: 0x4010ffff to 0x400ffff ■ Region 4: 0x5010ffff to 0x500ffff ■ Region 5: 0x6010ffff to 0x600ffff ■ Region 6: 0x7010ffff to 0x700ffff ■ Region 7: 0x8010ffff to 0x800ffff ■ Region 8: 0x9010ffff to 0x900ffff <p>Note: Default values for regions 1 to 8 are mentioned for AXI bus width (AXI_AW) of 32 bits</p> <p>Values: 0x0, ..., 0xffffffff</p> <p>Default Value: See Description</p> <p>Enabled: There must be at least one slave in the system and at least one master must have access to Slave 1; additionally, AXI_HAS_XCDR=0</p> <p>Parameter Name: AXI_R(i)_BEA_S(j)</p>

4.18 Master Port Configuration Parameters

Table 4-18 Master Port Configuration Parameters

Label	Description
Read Accept Per Unique ID Limit	
Read Accept Limit per ID for Master x (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$)	<p>This sets the maximum number of outstanding read commands that an external Master x can generate for a specific ARID value before DW_axi stalls the master's read address channel. In other words, this is the maximum number of read commands that Master port x can accept for a particular ARID value. This limits the hardware required to implement out-of-order deadlock prevention.</p> <p>$\text{AXI_MAX_RCA_ID_M}(x) \leq \text{AXI_MAX_FARC_S}(j)$ of all slaves visible to that master. Otherwise, DW_axi supports a master having more outstanding transactions than what any of its slaves are capable of accepting.</p> <p>Values: 1, ..., 128</p> <p>Default Value: 4</p> <p>Enabled: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Parameter Name: $\text{AXI_MAX_RCA_ID_M}(x)$</p>
Active Unique Read ID Accept Limit	
Read ID Accept Limit per ID for Master x (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$)	<p>This sets the maximum number of outstanding read commands that an external Master x can generate with unique ARID values before DW_axi stalls the master's read address channel. In other words, this is the maximum number of unique values of ARID that Master port x can accept for outstanding read commands at any one time. This limits the hardware required to implement out-of-order deadlock prevention. Also, this limits the number of client inputs to the read data channel arbiter in the master port as a maximum of $\text{AXI_MAX_URIDA_M}(x)$ slaves can be requesting a master x's read data channel simultaneously.</p> <p>For non-ICM ports, this must be less than or equal to 2 to the power of AXI_MIDW. For ICM ports, this must be less than 64, but to avoid redundancy, you must take care to set it to a value less than or equal to the number of unique ID values that can reach this ICM port from all system masters that are visible to this port.</p> <p>Note: When two system masters send the same master ID to an ICM port, the ID values from each is unique at the ICM port, because the system master number will be pre-pended on by that point).</p> <p>Values: 1, ..., AXI_MAX_UIDA</p> <p>Default Value: 5 commands</p> <p>Enabled: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Parameter Name: $\text{AXI_MAX_URIDA_M}(x)$</p>

Table 4-18 Master Port Configuration Parameters (Continued)

Label	Description
Write Accept Per Unique ID Limit	
Write Accept Limit per ID for Master x (for x = 1; x <= AXI_NUM_MASTERS)	<p>This sets the maximum number of outstanding write commands that an external Master x can generate for a specific AWID value before DW_axi stalls the master's write address channel. In other words, this is the maximum number of write commands that Master port x can accept for a particular AWID value. This limits the hardware required to implement out-of-order deadlock prevention.</p> <p>AXI_MAX_WCA_ID_M(x) <= AXI_MAX_FAWC_S(j) of all slaves visible to that master. Otherwise, DW_axi supports a master having more outstanding transactions than any of its slaves are capable of accepting.</p> <p>Values: 1, ..., 128</p> <p>Default Value: 4</p> <p>Enabled: AXI_NUM_MASTERS >= x</p> <p>Parameter Name: AXI_MAX_WCA_ID_M(x)</p>
Active Unique Write ID Accept Limit	
Write ID Accept Limit per ID for Master x (for x = 1; x <= AXI_NUM_MASTERS)	<p>This sets the maximum number of outstanding write commands that an external Master x can generate with unique AWID values before DW_axi stalls the Master's write address channel. In other words, this is the maximum number of unique values of AWID that the Master port x can accept for outstanding write commands at any one time. This limits the hardware required to implement out-of-order deadlock prevention. This also limits the number of client inputs to the write response channel arbiter in the master port as a maximum of AXI_MAX_UWIDA_M(x) slaves can be requesting a Master x's write response channel simultaneously.</p> <p>For non-ICM ports, this must be less than or equal to 2 to the power of AXI_MIDW. For ICM ports, this must be less than 64, but to avoid redundancy, you must take care to set it to a value less than or equal to the number of unique ID values that can reach this ICM port from all system masters that are visible to this port.</p> <p>Note: When two system masters send the same master ID to an ICM port, the ID values from each is unique at the ICM port, because the system master number will be pre-pended on by that point).</p> <p>Values: 1, ..., AXI_MAX_UIDA</p> <p>Default Value: 5 commands</p> <p>Enabled: AXI_NUM_MASTERS >= x</p> <p>Parameter Name: AXI_MAX_UWIDA_M(x)</p>

Table 4-18 Master Port Configuration Parameters (Continued)

Label	Description
Read Data Interleaving	
Limit Interleave Depth of Master(x) read data channel to x? (for x = 1; x <= AXI_NUM_MASTERS)	<p>This parameter enforces an interleave depth of 1 on the read data channel of Master Port(x). If set to False (0), the interleaved depth of the read data channel of Master Port(x) is AXI_MAX_URIDA_M(x).</p> <p>This parameter is disabled if Master(x) does not have a dedicated read data channel; this is because the shared read data channel does not support read data interleave limiting.</p> <p>NOTE: There are potential bus deadlock conditions associated with setting this parameter to True. For more details, see section "Read Data Interleaving" of the DW_axi Databook.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: AXI_NUM_MASTERS > x and AXI_M(x)_ON_R_SHARED_ONLY_VAL = 0</p> <p>Parameter Name: AXI_RI_LIMIT_M(x)</p>

4.19 Slave Port Configuration Parameters

Table 4-19 Slave Port Configuration Parameters

Label	Description
Write Interleaving Depth	
Write Interleave Depth for Slave (j) (for x,j = 1; x,j <= 8, AXI_NUM_SLAVES)	<p>This sets the write interleave depth for Slave j. The data from multiple masters can be interleaved to Slave j, if the slave supports write data interleaving.</p> <p>This must be set less than or equal to the write data interleave depth of the attached slave.</p> <p>If AXI_NUM_MASTERS = 1 OR if the number of masters visible to this slave = 1 (in both boot and normal modes), then this parameter is inactive. Otherwise, it must be less than or equal to AXI_NUM_MASTERS. When this parameter is inactive, DW_axi does not restrict the write data interleaving to a maximum value.</p> <p>Also, AXI_WID_S(j) <= AXI_MAX_FAWC_S(j); otherwise, DW_axi is able to interleave from more sources than it can accept transactions from, so there would be redundant write interleaving slots.</p> <p>This parameter is disabled in the following scenarios:</p> <ul style="list-style-type: none"> Slave j does not have a dedicated write data channel because the shared write data channel supports only an interleave depth of 1 to slaves. AXI_INTERFACE_TYPE is selected and set to AXI4 or ACE-Lite, nor AXI3; in this case, the parameter value is set to 1. <p>Values: 1, ..., 8</p> <p>Default Value: 1</p> <p>Enabled: AXI_NUM_MASTERS, AXI_NV_S(j)_BY_M(x), AXI_BV_S(j)_BY_M(x), AXI_MAX_FAWC_S(j), AXI_INTERFACE_TYPE</p> <p>Parameter Name: AXI_WID_S(j)</p>

Table 4-19 Slave Port Configuration Parameters (Continued)

Label	Description
Write Forwarding Limit	
Maximum Number of Forwarded Write Commands for Slave (j) (for x,j = 1; x,j <= 8,AXI_NUM_SLAVES)	<p>This sets the maximum number of outstanding write commands that can be forwarded to Slave 1. In other words, it is the maximum number of outstanding write commands that Slave port 1 can generate. This setting limits the hardware required to track write interleaving rules. It also limits hardware required by a slave port to ensure that it does not generate a locked command until all outstanding read/write commands previously issued are complete.</p> <p>If AXI_NUM_MASTERS = 1 OR if the number of masters visible to this slave = 1, then this parameter is inactive and the maximum number of outstanding write commands that can be forwarded to Slave 1 is not limited by DW_axi.</p> <p>For all visible master ports, AXI_MAX_FAWC_S(j) must be less than or equal to min(Sum(AXI_MAX_WCA_ID_M(x) * AXI_MAX_UWIDA_M(x)), EXT_OSW_MAX_S(j)) of all visible master ports.</p> <p>Where, EXT_OSW_MAX_S(j) is maximum outstanding write transactions supported by the external slave connected to the slave port(j) (EXT_OSW_MAX_S(j) is parameter associated with connected slave, it is not a DW_axi parameter).</p> <p>To save logic in the DW_axi instance, the user must set this parameter to a value less than the maximum limit outlined above, but not greater than the acceptance capability of the attached slave on this channel.</p> <p>Values: 1, ..., 256</p> <p>Default Value: 4</p> <p>Enabled: (AXI_NUM_SLAVES >= j) AND (AXI_NMV_S(j) > 1)</p> <p>Parameter Name: AXI_MAX_FAWC_S(j)</p>

Table 4-19 Slave Port Configuration Parameters (Continued)

Label	Description
Read Forwarding Limit	
Maximum Number of Forwarded Read Commands for Slave (j) (for x,j = 1; x,j <= 8,AXI_NUM_SLAVES)	<p>This specifies the maximum number of outstanding read commands that can be forwarded to Slave 1. In other words, it is the maximum number of outstanding read commands that Slave port 1 can generate. This parameter limits hardware required by the slave port to ensure that it does not generate a locked command until all the outstanding read/write commands previously issued are complete. If AXI_NUM_MASTERS = 1 OR if the number of masters visible to this slave = 1, then this parameter is inactive and the maximum number of outstanding read commands that can be forwarded to Slave 1 is not limited by DW_axi. AXI_MAX_FARC_S(j) must be <= min(Sum(AXI_MAX_RCA_ID_M(x) * AXI_MAX_URIDA_M(x)), EXT_OSR_MAX_S(j)) of all the visible master ports. Where, EXT_OSR_MAX_S(j) is maximum outstanding read transactions supported by the external slave connected to the slave port j. (EXT_OSR_MAX_S(j) is parameter associated with connected slave , It is not a DW_axi parameter)</p> <p>To save logic in the DW_axi instance the user should set this parameter to a value less than the maximum limit outlined above, but not greater than the acceptance capability of the attached slave on this channel.</p> <p>Values: 1, ..., 256</p> <p>Default Value: 4</p> <p>Enabled: (AXI_NUM_SLAVES >= j) AND (AXI_NMV_S(j) > 1)</p> <p>Parameter Name: AXI_MAX_FARC_S(j)</p>

4.20 QoS Options / Register Interface Configuration Parameters

Table 4-20 QoS Options / Register Interface Configuration Parameters

Label	Description
APB Data Bus width Configuration	
Select APB Read and Write Data Bus Width	<p>Selects APB read and write data bus widths.</p> <p>Values: 8, 16, 32</p> <p>Default Value: 32</p> <p>Enabled: 0</p> <p>Parameter Name: APB_DATA_WIDTH</p>
Base Address Configuration	
Base Address for APB registers on AXI interconnect	<p>Determines base address for DW_axi APB register programming. Bits[5:0] are used for the APB register offset and should always be set as all zeros.</p> <p>Values: 0x00000, ..., 0xfffffc0</p> <p>Default Value: 0x00400</p> <p>Enabled: AXI_HAS_QOS == 1</p> <p>Parameter Name: AXI_IC_REG_BASE_ADDR</p>
Synchronization Depth Configuration	
Dual clock mode Synchronization Depth	<p>This parameter determines the number of synchronizer buffer stages for APB-to-AXI clock domain crossing signals.</p> <p>Values: 2, 3, 4</p> <p>Default Value: 2</p> <p>Enabled: AXI_HAS_QOS == 1</p> <p>Parameter Name: AXI_NUM_SYNC_FF</p>

4.21 QoS Options / QoS Signals Parameters

Table 4-21 QoS Options / QoS Signals Parameters

Label	Description
QoS port on Write Address Channel - awqos_m	
Include QoS Ports on Write Address? (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the QoS signals' driver source at each master port Read Address channels.</p> <ul style="list-style-type: none"> 1: External mode: AWQOS_M(x) port exists on interconnect top level for master port(x) Read Address channel. Value on the port is driven from external source. 0: Internal mode: AWQOS_M(x) port does not exist on interconnect top level for master port(x) Read Address channel. The signal value is driven from APB register interface, which supports a dedicated register for dynamic programming of the signal. Any transition on these signals is synchronized to the AXI clock. <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: False</p> <p>Enabled: (AXI_NUM_MASTERS >= x) AND (AXI_HAS_QOS=1) AND (AXI_INTERFACE_TYPE=0)</p> <p>Parameter Name: AXI_HAS_AWQOS_EXT_M(x)</p>
QoS port on Read Address Channel - arqos_m	
Include QoS Ports on Read Address? (for x = 1; x <= AXI_NUM_MASTERS)	<p>Selects the QoS signals' driver source at each master port Read Address channels.</p> <ul style="list-style-type: none"> 1: External mode: ARQOS_M(x) port exists on interconnect top level for master port(x) Read Address channel. Value on the port is driven from external source. 0: Internal mode: ARQOS_M(x) port does not exist on interconnect top level for master port(x) Read Address channel. The signal value is driven from APB register interface, which supports a dedicated register for dynamic programming of the signal. Any transition on these signals is synchronized to the AXI clock. <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: False</p> <p>Enabled: (AXI_NUM_MASTERS >= x) AND (AXI_HAS_QOS=1) AND (AXI_INTERFACE_TYPE=0)</p> <p>Parameter Name: AXI_HAS_ARQOS_EXT_M(x)</p>

4.22 QoS Options / QoS Regulator Module Configuration Parameters

Table 4-22 QoS Options / QoS Regulator Module Configuration Parameters

Label	Description
Write Address Channel QoS Regulator Module	
Include QoS Regulator for Write Address Channel for Master (x)? (for x = 1; x <= AXI_NUM_MASTERS)	<p>Provides option to include or exclude QoS regulator logic on MasterPort(x) Read Address Channel.</p> <ul style="list-style-type: none"> 0: Exclude QoS regulator logic on xth Master Port Read Address channel. 1: Include QoS regulator logic on xth Master Port Read Address channel. <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: False</p> <p>Enabled: (AXI_NUM_MASTERS >= x) AND (AXI_HAS_QOS=1)</p> <p>Parameter Name: AXI_AW_HAS_QOS_REGULATOR_M(x)</p>
Read Address Channel QoS Regulator Module	
Include QoS Regulator for Read Address Channel for Master (x)? (for x = 1; x <= AXI_NUM_MASTERS)	<p>Provides option to include or exclude QoS regulator logic on MasterPort(x) Read Address Channel.</p> <ul style="list-style-type: none"> 0: Exclude QoS regulator logic on xth Master Port Read Address channel. 1: Include QoS regulator logic on xth Master Port Read Address channel. <p>Values:</p> <ul style="list-style-type: none"> false (0) true (1) <p>Default Value: False</p> <p>Enabled: (AXI_NUM_MASTERS >= x) AND (AXI_HAS_QOS=1)</p> <p>Parameter Name: AXI_AR_HAS_QOS_REGULATOR_M(x)</p>

5

Register Descriptions

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

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

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

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

Exists Expressions

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

Offset

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

Memory Access Attributes

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

Table 5-1 Possible Read and Write Behaviors

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

Table 5-2 Memory Access Examples

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

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

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

Component Banks/Blocks

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

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

Address Block	Description
axi_address_block on page 169	<p>DW_axi General Memory Block.</p> <p>Note: The offset of the registers (0x400, 0x404, 0x408 and 0x40c) are derived from the AXI_IC_REG_BASE_ADDR parameter value (default value of 0x400). For register address decoding the DW_axi, considers only lower 6 bits of the address.</p> <p>Exists: Always</p>

5.1 axi_memory_map/axi_address_block Registers

DW_axi General Memory Block.

Note: The offset of the registers (0x400, 0x404, 0x408 and 0x40c) are derived from the AXI_IC_REG_BASE_ADDR parameter value (default value of 0x400). For register address decoding the DW_axi, considers only lower 6 bits of the address.

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

Table 5-5 Registers for Address Block: axi_memory_map/axi_address_block

Register	Offset	Description
AXI_VERSION_ID_REG on page 170	* Varies	DW_axi Component Version ID register
AXI_HW_CFG_REG on page 171	* Varies	DW_axi Hardware Configuration register
AXI_CMD_REG on page 174	* Varies	DW_axi Command Enable Register
AXI_DATA_REG on page 180	* Varies	DW_axi Data Register

5.1.1 AXI_VERSION_ID_REG

- **Name:** DW_axi Component Version ID register
- **Description:** This register represents the DW_axi_component version.
- **Size:** 32 bits
- **Offset:** AXI_IC_REG_BASE_ADDR
- **Exists:** AXI_HAS_QOS

AXI_VERID_FIELDS 31:0

Table 5-6 Fields for Register: AXI_VERSION_ID_REG

Bits	Name	Memory Access	Description
31:0	AXI_VERID_FIELDS	R	DW_AXI Version ID Number. ASCII value of each number in component version followed by *. For example, 33_30_30_2A represents the version 3.00*. Value After Reset: DW_AXI_VERSION_ID Exists: Always

5.1.2 AXI_HW_CFG_REG

- **Name:** DW_axi Hardware Configuration register
- **Description:** This register provides the configuration details of the DW_axi.
- **Size:** 32 bits
- **Offset:** 0x4 + AXI_IC_REG_BASE_ADDR
- **Exists:** AXI_HAS_QOS

AXI_HW_CFG_REG_RSVD_31to25	31:25
AXI_NUM_SLAVES	24:20
AXI_HW_CFG_REG_RSVD_19to17	19:17
AXI_NUM_MASTERS	16:12
AXI_HW_CFG_REG_RSVD_11to9	11:9
AXI_LOW_POWER_IF	8
AXI_BI_DIR_CMD_EN	7
AXI_REMAP_EN	6
AXI_DECODER_TYPE	5
AXI_HWCFG_TRUST_ZONE_EN	4
AXI_HWCFG_LOCK_EN	3
AXI_HWCFG_AXI4_SUPPORT	2
AXI_HWCFG_APB3_SUPPORT	1
AXI_HWCFG_QOS_SUPPORT	0

Table 5-7 Fields for Register: AXI_HW_CFG_REG

Bits	Name	Memory Access	Description
31:25	AXI_HW_CFG_REG_RSVD_31to25	R	Reserved Value After Reset: 0x0 Exists: Always
24:20	AXI_NUM_SLAVES	R	Number of AXI Slaves. Value After Reset: AXI_NUM_SLAVES Exists: Always
19:17	AXI_HW_CFG_REG_RSVD_19to17	R	Reserved Value After Reset: 0x0 Exists: Always
16:12	AXI_NUM_MASTERS	R	Number of AXI masters. Value After Reset: AXI_NUM_MASTERS Exists: Always

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

Bits	Name	Memory Access	Description
11:9	AXI_HW_CFG_REG_RSVD_11to9	R	Reserved Value After Reset: 0x0 Exists: Always
8	AXI_LOW_POWER_IF	R	Include/exclude low-power interface. Values: <ul style="list-style-type: none"> 0x1 (REMAP_SUPPORTED): DW_axi- Low-power interface is included. 0x0 (REMAP_NOT_SUPPORTED): DW_axi- Low-power interface is not included. Value After Reset: AXI_LOWPWR_HS_IF Exists: Always
7	AXI_BI_DIR_CMD_EN	R	Support for Enable Bi-directional Command. Values: <ul style="list-style-type: none"> 0x1 (REMAP_SUPPORTED): DW_axi supports the bi-directional command. 0x0 (REMAP_NOT_SUPPORTED): DW_axi does not support the bi-directional command. Value After Reset: AXI_HAS_BICMD Exists: Always
6	AXI_REMAP_EN	R	Support for Enable Remap mode. Values: <ul style="list-style-type: none"> 0x1 (REMAP_SUPPORTED): DW_axi supports Enable Remap mode. 0x0 (REMAP_NOT_SUPPORTED): DW_axi does not support Enable Remap mode. Value After Reset: AXI_REMAP_EN Exists: Always
5	AXI_DECODER_TYPE	R	Use of External/Internal Decoder. Values: <ul style="list-style-type: none"> 0x1 (EXT_DECODER): DW_axi uses external decoder. 0x0 (INT_DECODER): DW_axi does not use the external decoder. Value After Reset: AXI_HAS_XDCDR Exists: Always

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

Bits	Name	Memory Access	Description
4	AXI_HWCFG_TRUST_ZONE_EN	R	Support for Enable Trust Zone. Values: <ul style="list-style-type: none"> 0x1 (TZEN_SUPPORTED): DW_axi-Trust Zone features are Enabled. 0x0 (TZEN_NOT_SUPPORTED): DW_axi-Trust Zone features are Disabled. Value After Reset: AXI_HAS_TZ_SUPPORT Exists: Always
3	AXI_HWCFG_LOCK_EN	R	Support for Enable Lock transactions. Values: <ul style="list-style-type: none"> 0x1 (LOCKEN_SUPPORTED): DW_axi-Locking feature is Enabled. 0x0 (LOCKEN_NOT_SUPPORTED): DW_axi-Locking feature is Disabled. Value After Reset: AXI_HAS_LOCKING Exists: Always
2	AXI_HWCFG_AXI4_SUPPORT	R	Support for AXI4 features. Values: <ul style="list-style-type: none"> 0x1 (AXI4_SUPPORTED): DW_axi supports AXI4 features. 0x0 (AXI4_NOT_SUPPORTED): DW_axi does not support AXI4 features. Value After Reset: "(AXI_INTERFACE_TYPE==0) ? 0:1" Exists: Always
1	AXI_HWCFG_APB3_SUPPORT	R	Has APB3 support. Values: <ul style="list-style-type: none"> 0x1 (APB3_SUPPORTED): DW_axi has APB3 support. 0x0 (APB3_NOT_SUPPORTED): DW_axi does not have APB3 support. Value After Reset: AXI_HAS_APB3 Exists: Always
0	AXI_HWCFG_QOS_SUPPORT	R	QOS Support. Values: <ul style="list-style-type: none"> 0x1 (QOS_ENABLED): DW_axi QOS support is enabled. 0x0 (QOS_DISABLED): DW_axi QOS support is disabled. Value After Reset: AXI_HAS_QOS Exists: Always

5.1.3 AXI_CMD_REG

- **Name:** DW_axi Command Enable Register
- **Description:** By programming this register, the QoS internal registers are either updated or the data from the QoS internal registers is read back.
- **Size:** 32 bits
- **Offset:** $0x8 + \text{AXI_IC_REG_BASE_ADDR}$
- **Exists:** AXI_HAS_QOS

AXI_CMD_EN	31
AXI_RD_WR_CMD	30
AXI_SOFT_RESET_BIT	29
AXI_ERR_BIT	28
AXI_CMD_REG_RSVD_27to12	27:12
AXI_MASTER_PORT	11:8
AXI_RD_WR_CHAN	7
AXI_CMD_REG_RSVD_6to3	6:3
AXI_CMD	2:0

Table 5-8 Fields for Register: AXI_CMD_REG

Bits	Name	Memory Access	Description
31	AXI_CMD_EN	R/W	<p>Command enable. This bit must be set to 1 to execute the command in this register. In case the bit is not set to 1, the command in the register will not be executed. Once set to 1, this bit can be reset to 0 only by the interconnect logic after performing the command in the register. In case, APB master tries to write another command in the register that will be ignored without any error message. Similarly, APB master will not be allowed to clear this bit. The access type is R_W0 (Read and Write on 0).</p> <ul style="list-style-type: none"> ■ 1: The command in the register is valid and it is not yet completed by the interconnect. The command and data registers are locked for any updates till this bit is reset to 0. Any write to this bit by APB master when it is already set to 1 will be ignored without any error. ■ 0: The command register is ready to accept new commands. The data register holds the data of the last successful transaction depending on whether it was for a read or write. <p>It is recommended to write the data in the data register first before writing the command in the command register for write transaction. As soon as command enable bit is set to 1 and it is for the write access, the data register and command registers can't be updated by the APB master. Similarly, it is recommended that APB master should read the data register to get the read-data of the command mentioned in the command register only once command enable bit is reset to 0 by the interconnect.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x1 (EXEC_CMD): DW_AXI: executes a read/write command on QoS registers. ■ 0x0 (ACCEPT_CMD): DW_AXI: is ready to accept commands. <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Volatile: true</p>

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

Bits	Name	Memory Access	Description
30	AXI_RD_WR_CMD	R/W	<p>Read/write bit. This bit decodes if the type of command for QoS internal registers offset mentioned in CMD [2:0]. It is valid only if Command enable bit is set to 1. Also, it is associated with the master port mentioned in bit[11:8] of this register and the command is for that master port only. The types of command controlled by this bits are:</p> <ul style="list-style-type: none"> 0: Read command 1: Write command <p>It is valid when command enable bit is set to 1 by the APB master.</p> <p>Values:</p> <ul style="list-style-type: none"> 0x1 (WR_CMD): DW_AXI: Write Command for the QoS internal registers. 0x0 (RD_CMD): DW_AXI: Read Command for the QoS internal registers. <p>Value After Reset: 0x0</p> <p>Exists: Always</p>
29	AXI_SOFT_RESET_BIT	R/W	<p>Internal Register reset. When this bit is set to 1 by the APB master, it resets all registers in the interconnect to their reset values. It is valid when command enable is set to 1 by the APB master. If this bit is 1 along with the command enable bit then it will execute Internal Registers Reset command irrespective of other bits value.</p> <p>Values:</p> <ul style="list-style-type: none"> 0x1 (RESET_INT_REG): DW_AXI: If CommandReg Enable bit is also set, then the internal registers are reset. 0x0 (ACTIVE_INT_REG): DW_AXI: The internal registers are out of reset. <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Volatile: true</p>

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

Bits	Name	Memory Access	Description
28	AXI_ERR_BIT	R	<p>Error Bit. This bit is a read-only bit as it is set to 1 only by the interconnect in case of error scenarios. It is valid only once command enable is reset to 0 by the interconnect. It will be cleared to 0 on next write to the command register by APB master. It will be set under the following conditions:</p> <ul style="list-style-type: none"> ■ On any QoS internal register read or write command issued for invalid master (master number index). ■ On any QoS internal register read or write command issued for undefined commands. ■ If trying to read or write command register fields (BURSTINESS_REGULATOR_EN, PEAK_RATE_XCT_RATE, SLV_READY) for a master (channel) whose regulator AXI_AW_HAS_QOS_REGULATOR_M(j) parameter is disabled. ■ If trying to read or write command register fields (QOS_VALUE) for a master (channel) whose AXI_HAS_AWQOS_EXT_M (j) parameter is enabled. <p>Values:</p> <ul style="list-style-type: none"> ■ 0x1 (ERR_BIT_SET): DW_AXI: The Error Bit is set. ■ 0x0 (ERR_BIT_NOT_SET): DW_AXI: The Error Bit is not set. <p>Value After Reset: 0x0</p> <p>Exists: Always</p>
27:12	AXI_CMD_REG_RSVD_27to12	R	<p>Reserved</p> <p>Value After Reset: 0x0</p> <p>Exists: Always</p>

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

Bits	Name	Memory Access	Description
11:8	AXI_MASTER_PORT	R/W	<p>Master Port ID. Identifies the master port for which command (bit[2:0]) is configured.</p> <p>It is readable and writeable by the APB master to following values:</p> <ul style="list-style-type: none"> ■ 4'h0: master port 1 ■ 4'h1: master port 2 ■ 4'h2: master port 3 ■ ■ 4'hF: master port 16 ■ Master port number = bit[11:8] + 1 <p>It is valid when command enable bit is set to 1 by the APB master.</p> <p>Value After Reset: 0x0</p> <p>Exists: Always</p>
7	AXI_RD_WR_CHAN	R/W	<p>Read Address Channel /Write Address Channel. Identifies that the command is for read address channel or write address channel of the master port (bit[11:8]).</p> <ul style="list-style-type: none"> ■ 0: the command is for read address channel. ■ 1: the command is for write address channel. <p>It is valid when command enable bit is set to 1 by the APB master</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x1 (CMD_READ_ADDR_CH): DW_AXI: The command is for read address channel. ■ 0x0 (CMD_WRITE_ADDR_CH): DW_AXI: The command is for write address channel. <p>Value After Reset: 0x0</p> <p>Exists: Always</p>
6:3	AXI_CMD_REG_RSVD_6to3	R	<p>Reserved</p> <p>Value After Reset: 0x0</p> <p>Exists: Always</p>

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

Bits	Name	Memory Access	Description
2:0	AXI_CMD	R/W	<p>Command selection for internal registers. Following are the address offsets of the QoS internal registers dedicated to read address channel or write address channel for the master port number programmed at CommandReg[11:8]. CommandReg[7] determines the read address channel or write address channel. QoS internal registers address offsets are:</p> <ul style="list-style-type: none"> ■ 3'b000: BURSTINESS_REGULATOR_EN ■ 3'b001: PEAK_RATE_XCT_RATE ■ 3'b010: QOS_VALUE ■ 3'b011: SLV_READY <p>It is valid when command enable bit is set to 1 by the APB master.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (BURSTINESS_REGULATOR_EN) ■ 0x1 (PEAK_RATE_XCT_RATE) ■ 0x2 (QOS_VALUE) ■ 0x3 (SLV_READY) <p>Value After Reset: 0x0</p> <p>Exists: Always</p>

5.1.4 AXI_DATA_REG

- **Name:** DW_axi Data Register
- **Description:** This register is programmed through the APB interface.
DataReg[31]-DataReg[00] provides:
 - Write data when the QoS internal registers are set for write operations. Data register bits are written in the targeted QoS internal register when a QoS internal register write command is executed.
 - Read data when the QoS internal registers are set for read operations. The DataReg register is updated with QoS internal register data when a QoS internal register read command is executed.
- **Size:** 32 bits
- **Offset:** 0xc + AXI_IC_REG_BASE_ADDR
- **Exists:** AXI_HAS_QOS

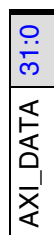


Table 5-9 Fields for Register: AXI_DATA_REG

Bits	Name	Memory Access	Description
31:0	AXI_DATA	R/W	Data to be written into the QoS internal register or the data read from the QoS internal register. Value After Reset: 0x0 Exists: Always

6

Signal Descriptions

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

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

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

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

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

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

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

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

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

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

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

Attributes used with Synchronous To

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

The I/O signals are grouped as follows:

- Clocks and Resets on [page 183](#)
- APB Interface Signals on [page 185](#)
- Master Port x Read Address Channel on [page 188](#)
- Master Port x Write Address Channel on [page 194](#)
- Master Port x Write Data Channel on [page 199](#)
- Master Port x Write Response Channel on [page 202](#)
- Master Port x Read Data Channel on [page 205](#)
- AXI Low Power Interface on [page 208](#)
- Deadlock Notification on [page 209](#)
- Slave Port j Write Address Channel on [page 211](#)
- Slave Port j Write Data Channel on [page 217](#)
- Slave Port j Write Response Channel on [page 220](#)
- Slave Port j Read Address Channel on [page 222](#)
- Slave Port j Read Data Channel on [page 229](#)
- Default Slave Write Address Channel on [page 232](#)
- Default Slave Write Data Channel on [page 235](#)
- Default Slave Write Response Channel on [page 237](#)
- Default Slave Read Address Channel on [page 238](#)
- Default Slave Read Data Channel on [page 241](#)

6.1 Clocks and Resets Signals

aclk -
 aresetn -
 mst_priority_shared -
 slv_priority_shared -
 remap_n -

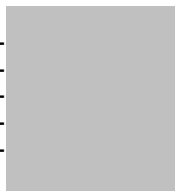


Table 6-1 Clocks and Resets Signals

Port Name	I/O	Description
aclk	I	<p>AXI Clock Signal. All input signals are sampled on the rising edge of aclk. All output signals must occur after the rising edge of aclk. There must be no combinatorial paths between input and output signals on both master and slave interfaces.</p> <p>Exists: Always</p> <p>Synchronous To: None</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
aresetn	I	<p>AXI Reset Signal. The bus reset signal is active low and is used to reset the system and the bus. During reset, the following must occur:</p> <ul style="list-style-type: none"> ■ A master interface must drive arvalid, awvalid, and wvalid low. ■ A slave interface must drive rvalid and bvalid low. <p>Asynchronous assertion, synchronous de-assertion. The reset must be deasserted synchronously after the rising edge of aclk. DW_axi does not contain logic to perform this synchronization, so it must be provided externally.</p> <p>Exists: Always</p> <p>Synchronous To: None</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: Low</p>

Table 6-1 Clocks and Resets Signals (Continued)

Port Name	I/O	Description
mst_priority_shared[(AXI_MST_PRIORITY_W-1):0]	I	<p>Arbitration priority associated with the shared master port when it requests on behalf of slaves to a dedicated master port. This priority can be changed during operation of the component. Legal values are all bits zero (0) to all 1s, with all bits zeroes (0) considered the lowest priority and all bits one (1) the highest priority.</p> <p>Exists: (AXI_SHARED_LAYER_MASTER_PRIORITY_EN_VAL == 1) && (AXI_HAS_EXT_PRIORITY == 1)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
slv_priority_shared[(AXI_SLV_PRIORITY_W-1):0]	I	<p>Arbitration priority associated with the shared slave port when it requests on behalf of masters to a dedicated slave port. This priority can be changed during operation of the component. Legal values are all bits zero (0) to all bits one (1), with all zeroes (0) considered the lowest priority and all 1s the highest priority.</p> <p>Exists: (AXI_SHARED_LAYER_SLAVE_PRIORITY_EN_VAL == 1) && (AXI_HAS_EXT_PRIORITY == 1)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
remap_n	I	<p>Selection of memory map. When there are two memory maps (for normal and boot modes), they are selected by remap_n (AXI_REMAP_EN = 1). Boot mode is selected when remap_n is low; normal mode is selected when remap_n is high. When there is only one memory map, remap_n is not included in the top-level I/O. It is always '1' (high) when there is only one memory map.</p> <p>Exists: (AXI_REMAP_EN==1)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: Low</p>

6.2 APB Interface Signals

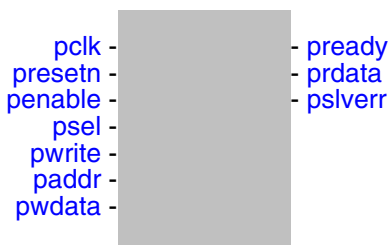


Table 6-2 APB Interface Signals

Port Name	I/O	Description
pclk	I	<p>APB Clock. The rising edge of pclk times all transfers the APB interface.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: None</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
presetn	I	<p>APB reset signal is active low. This signal is connected directly to the system bus reset signal. Asynchronous assertion, synchronous de-assertion. The reset must be de-asserted synchronously after the rising edge of pclk. The DW_axi does not contain logic to perform this synchronization, so it must be provided externally.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: None</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: Low</p>
penable	I	<p>APB enable. Active high signal that indicates the second and subsequent cycles of an APB transfer.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>


Table 6-2 APB Interface Signals (Continued)

Port Name	I/O	Description
psel	I	<p>APB select. The APB bridge unit generates this signal to each peripheral bus slave. The signal indicates that the slave device is selected and data transfer is required. Active high.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
pwrite	I	<p>APB write access when high, and an APB read access when low.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
paddr[31:0]	I	<p>APB address bus. This signal is 32 bits wide and is driven by the peripheral bus bridge unit.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
pwwdata[(APB_DATA_WIDTH-1):0]	I	<p>APB write data. This bus is driven by the peripheral bus bridge unit during write cycles when pwrite is high.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
pready	O	<p>APB ready. Active high signal that indicates the slave is ready.</p> <p>Exists: (AXI_HAS_QOS == 1) && (AXI_HAS_APB3 == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

Table 6-2 APB Interface Signals (Continued)

Port Name	I/O	Description
prdata[(APB_DATA_WIDTH-1):0]	O	<p>APB read data. The selected slave drives this bus during read cycles when pwrite is low.</p> <p>Exists: (AXI_HAS_QOS == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: Yes</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
pslverr	O	<p>APB slave error.</p> <p>Exists: (AXI_HAS_QOS == 1) && (AXI_HAS_APB3 == 1)</p> <p>Synchronous To: pclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.3 Master Port x Read Address Channel (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$) Signals



xdcdr_slv_num_rd_mx -
 arvalid_mx -
 arid_mx -
 araddr_mx -
 arlen_mx -
 arsize_mx -
 arburst_mx -
 arlock_mx -
 arcache_mx -
 arprot_mx -
 arsideband_mx -
 aruser_mx -
 arqos_mx -
 arsnoop_mx -
 ardomain_mx -
 arbar_mx -
 mst_priority_mx -

arready_mx

Table 6-3 Master Port x Read Address Channel (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$) Signals

Port Name	I/O	Description
xdcdr_slv_num_rd_mx[(AXI_LOG2_NSP 1-1):0]	I	Output of master external read decoder. Exists: (AXI_NUM_MASTERS \geq x) && (AXI_HAS_XDCR = 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
arvalid_mx	I	Master read address valid. This signal indicates that a valid read address and control information are available. <ul style="list-style-type: none"> 1: Address and control information available 0: Address and control information unavailable The address and control information remain stable until the address acknowledge signal, arready, goes high. Exists: AXI_NUM_MASTERS \geq x Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

Table 6-3 Master Port x Read Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
arid_mx[(AXI_IDW_M1-1):0]	I	<p>Master read address ID. This signal is the identification tag for the read address group of signals.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
araddr_mx[(AXI_AW-1):0]	I	<p>Master read address. The read address bus gives the address of the first transfer in a read burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arlen_mx[(AXI_BLW-1):0]	I	<p>Master burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The AXI protocol specifies this as a 4 bit value, but this width can be configured to 8 bits wide to support longer bursts up to 256 data beats.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arsize_mx[2:0]	I	<p>Master burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-3 Master Port x Read Address Channel (for $x = 1$; $x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
arburst_mx[1:0]	I	<p>Master burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AR_TMO} == 1) \parallel (\text{AXI_AR_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arlock_mx[(((AXI_INTERFACE_TYPE > 0) ? 1:2)-1):0]	I	<p>Master lock type. This signal provides additional information about the atomic characteristics of the transfer. The signal width and functionality changes between the AXI3 and AXI4/ACE-Lite interface types; for more information, refer to the 'Locked Transfers' section in the DW_axi Databook.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AR_TMO} == 1) \parallel (\text{AXI_AR_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arcache_mx[3:0]	I	<p>Master read cache type. This signal indicates the bufferable, cacheable/modifiable, write-through, write-back, and allocatable attributes of the transaction.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AR_TMO} == 1) \parallel (\text{AXI_AR_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arprot_mx[2:0]	I	<p>Master protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AR_TMO} == 1) \parallel (\text{AXI_AR_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-3 Master Port x Read Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
arsideband_mx[(AXI_AR_SBW-1):0]	I	<p>Optional. Master AXI3 sideband bus for read address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named arsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named aruser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_ARSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
aruser_mx[(AXI_AR_SBW-1):0]	I	<p>Optional. Master AXI4 or ACE-Lite user bus for read address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named arsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named aruser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_ARSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arqos_mx[3:0]	I	<p>Master Quality of Service identifier, conveying priority information associated with each transaction at read address channel of master port(x).</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (((AXI_HAS_QOS = 1) && (AXI_INTERFACE_TYPE > 0)) (AXI_HAS_ARQOS_EXT_M(x) = 1))</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-3 Master Port x Read Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
arready_mx	O	<p>Master read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.</p> <ul style="list-style-type: none"> 1: Slave ready 0: Slave not ready <p>Default State (after Power On Reset): When AXI_AR_TMO is not Combinatorial or AXI_AR_PL_ARB is true, then the default value is low. When AXI_AR_TMO is Combinatorial and AXI_AR_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
arsnoop_mx[3:0]	I	<p>This signal indicates the transaction type for shareable master read transactions.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
ardomain_mx[1:0]	I	<p>This signal indicates the shareability domain of a master read transaction.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arbar_mx[1:0]	I	<p>This signal indicates a master read barrier transaction.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AR_TMO == 1) (AXI_AR_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-3 Master Port x Read Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
mst_priority_mx[(AXI_MST_PRIORITY_W-1):0]	I	<p>Master arbitration priority associated with master port x. This priority can be changed during operation of the component. Legal values this input are all bits zero (0) up to all bits 1, with all zeros (0s) being considered lowest priority and all bits one (1) highest priority.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_EXT_PRIORITY = 1)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

6.4 Master Port x Write Address Channel (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$) Signals

xdcdr_slv_num_wr_mx -
 awvalid_mx -
 awaddr_mx -
 awid_mx -
 awlen_mx -
 awsize_mx -
 awburst_mx -
 awlock_mx -
 awcache_mx -
 awprot_mx -
 awsideband_mx -
 awuser_mx -
 awqos_mx -
 awsnoop_mx -
 awdomain_mx -
 awbar_mx -

awready_mx

Table 6-4 Master Port x Write Address Channel (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$) Signals

Port Name	I/O	Description
xdcdr_slv_num_wr_mx[(AXI_LOG2_NS P1-1):0]	I	Output of external master write decoder. Exists: (AXI_NUM_MASTERS $\geq x$) && (AXI_HAS_XDCR = 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
awvalid_mx	I	Master write address valid. This signal indicates that a valid write address and control information are available. <ul style="list-style-type: none"> 1: Address and control information available 0: Address and control information unavailable. The address and control information remain stable until the address acknowledge signal, awready, goes high. Exists: AXI_NUM_MASTERS $\geq x$ Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

Table 6-4 Master Port x Write Address Channel (for $x = 1$; $x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
awaddr_mx[(AXI_AW-1):0]	I	<p>Master write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awid_mx[(AXI_IDW_M1-1):0]	I	<p>Master write address ID. This signal is the identification tag for the write address group of signals.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awlen_mx[(AXI_BLW-1):0]	I	<p>Master burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The AXI protocol specifies this as a 4 bit value, but this width can be configured to 8 bits wide to support longer bursts up to 256 data beats.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awsiz_mx[2:0]	I	<p>Master burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-4 Master Port x Write Address Channel (for $x = 1; x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
awburst_mx[1:0]	I	<p>Master burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AW_TMO} == 1) \parallel (\text{AXI_AW_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awlock_mx[(((AXI_INTERFACE_TYPE > 0)?1:2)-1):0]	I	<p>Master lock type. This signal provides additional information about the atomic characteristics of the transfer. The signal width and functionality changes between the AXI3 and AXI4/ACE-Lite interface types. For more information, refer to 'Locked Transfers' section in the DW_axi Databook.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AW_TMO} == 1) \parallel (\text{AXI_AW_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awcache_mx[3:0]	I	<p>Master cache type. This signal indicates the bufferable, cacheable/modifiable, write-through, write-back, and allocatable attributes of the transaction.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AW_TMO} == 1) \parallel (\text{AXI_AW_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awprot_mx[2:0]	I	<p>Master protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.</p> <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_AW_TMO} == 1) \parallel (\text{AXI_AW_TMO} == 2))$? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p>

Table 6-4 Master Port x Write Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
awsideband_mx[(AXI_AW_SBW-1):0]	I	<p>Optional. Master AXI3 sideband bus for write address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named awsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named awuser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_AWSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p>
awuser_mx[(AXI_AW_SBW-1):0]	I	<p>Optional. Master AXI4/ACE-Lite user bus for write address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named awsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named awuser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_AWSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awqos_mx[3:0]	I	<p>Master Quality of Service identifier, conveying priority information associated with each transaction at Write Address channel of master port(x).</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (((AXI_HAS_QOS = 1) && (AXI_INTERFACE_TYPE > 0)) (AXI_HAS_AWQOS_EXT_M(x) = 1))</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-4 Master Port x Write Address Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
awready_mx	O	<p>Master write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.</p> <ul style="list-style-type: none"> 1: Slave ready 0: Slave not ready <p>Default State (after Power On Reset): When AXI_AW_TMO is not Combinatorial or AXI_AW_PL_ARB is true, then the default value is low. When AXI_AW_TMO is Combinatorial and AXI_AW_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
awsnoop_mx[2:0]	I	<p>This signal indicates the transaction type for shareable master write transactions.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awdomain_mx[1:0]	I	<p>This signal indicates the shareability domain of a master write transaction.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awbar_mx[1:0]	I	<p>This signal indicates a master write barrier transaction.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_TMO == 1) (AXI_AW_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

6.5 Master Port x Write Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals



Table 6-5 Master Port x Write Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals

Port Name	I/O	Description
wvalid_mx	I	<p>Master write valid. This signal indicates that valid write data and strobes are available.</p> <ul style="list-style-type: none"> 1: Write data and strobes available 0: Write data and strobes unavailable <p>Exists: AXI_NUM_MASTERS >= x Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High</p>
wid_mx[(AXI_IDW_M1-1):0]	I	<p>Master write ID tag. This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.</p> <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_INTERFACE_TYPE = AXI3) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A</p>
wdata_mx[(AXI_DW-1):0]	I	<p>Master write data.</p> <p>Exists: AXI_NUM_MASTERS >= x Synchronous To: aclk Registered: ((AXI_W_TMO == 1) (AXI_W_TMO == 2)) ? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A</p>

Table 6-5 Master Port x Write Data Channel (for $x = 1$; $x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
wstrb_mx[((AXI_DW/8)-1):0]	I	<p>Master write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, wstrb_m(x) corresponds to wdata_m[(8 * x) + 7:(8 * x)].</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_W_TMO == 1) (AXI_W_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
wlast_mx	I	<p>Master write last. This signal indicates the last transfer in a write burst.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
wsideband_mx[(AXI_W_SBW-1):0]	I	<p>Optional. Master AXI3 sideband bus for write data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named wsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named wuser_m. <p>Exists: (AXI_NUM_MASTERS $\geq x$) && (AXI_HAS_WSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_W_TMO == 1) (AXI_W_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
wuser_mx[(AXI_W_SBW-1):0]	I	<p>Optional. Master AXI4/ACE-Lite user bus for write data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named wsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named wuser_m. <p>Exists: (AXI_NUM_MASTERS $\geq x$) && (AXI_HAS_WSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_W_TMO == 1) (AXI_W_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-5 Master Port x Write Data Channel (for $x = 1$; $x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
wready_mx	O	<p>Master write ready. This signal indicates that the slave can accept the write data.</p> <ul style="list-style-type: none"> 1: Slave ready 0: Slave not ready <p>Default State (after Power On Reset): When AXI_W_TMO is not Combinatorial or AXI_W_PL_ARB is true, then the default value is low. When AXI_W_TMO is Combinatorial and AXI_W_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_MASTERS $\geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.6 Master Port x Write Response Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals

bready_mx - 
 - bvalid_mx
 - bid_mx
 - bresp_mx
 - bsideband_mx
 - buser_mx

Table 6-6 Master Port x Write Response Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals

Port Name	I/O	Description
bvalid_mx	O	<p>Master write response valid. This signal indicates that the valid write response data is available.</p> <ul style="list-style-type: none"> 1: Write response available 0: Write response unavailable <p>Default State (after Power On Reset): When AXI_B_TMO is not Combinatorial or AXI_B_PL_ARB is true, then the default value is low. When AXI_B_TMO is Combinatorial and AXI_B_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_B_LAYER_M(x)_S0 to AXI_B_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_B_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
bid_mx[(AXI_IDW_M1-1):0]	O	<p>Master response ID tag. This signal is the ID tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_B_LAYER_M(x)_S0 to AXI_B_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_B_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-6 Master Port x Write Response Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
bresp_mx[1:0]	O	<p>Master write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_B_LAYER_M(x)_S0 to AXI_B_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_B_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
bsideband_mx[(AXI_B_SBW-1):0]	O	<p>Optional. Master AXI3 sideband bus for write response channel. The signal name changes between the AXI3 and the AXI4/ACE Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named bsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named buser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_BSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_B_LAYER_M(x)_S0 to AXI_B_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_B_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
buser_mx[(AXI_B_SBW-1):0]	O	<p>Optional. Master AXI4 or ACE-Lite user bus for write response channel. The signal name changes between the AXI3 and the AXI4/ACE Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named bsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named buser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_BSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_B_LAYER_M(x)_S0 to AXI_B_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_B_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-6 Master Port x Write Response Channel (for $x = 1$; $x \leq \text{AXI_NUM_MASTERS}$) Signals (Continued)

Port Name	I/O	Description
bready_mx	I	<p>Master response ready. This signal indicates that the master can accept the response information.</p> <ul style="list-style-type: none">■ 1: Master ready■ 0: Master not ready <p>Exists: $\text{AXI_NUM_MASTERS} \geq x$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.7 Master Port x Read Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals



Table 6-7 Master Port x Read Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals

Port Name	I/O	Description
rvalid_mx	O	<p>Master read valid. This signal indicates that the required read data is available and the read transfer can complete.</p> <ul style="list-style-type: none"> 1: read data available 0: read data not available <p>Default State (after Power On Reset): When AXI_R_TMO is not Combinatorial or AXI_R_PL_ARB is true, then the default value is low. When AXI_R_TMO is Combinatorial and AXI_R_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
rid_mx[(AXI_IDW_M1-1):0]	O	<p>Master read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-7 Master Port x Read Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
rdata_mx[(AXI_DW-1):0]	O	<p>Master read data.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
rresp_mx[1:0]	O	<p>Master read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
rlast_mx	O	<p>Master read last. This signal indicates the last transfer in a read burst.</p> <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
rsideband_mx[(AXI_R_SBW-1):0]	O	<p>Optional. Master AXI3 sideband bus for read data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named rsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named ruser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_RSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-7 Master Port x Read Data Channel (for x = 1; x <= AXI_NUM_MASTERS) Signals (Continued)

Port Name	I/O	Description
ruser_mx[(AXI_R_SBW-1):0]	O	<p>Optional. Master AXI4/ACE-Lite user bus for read data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named rsideband_m. When the AXI4/ACE-Lite interface is enabled, this signal is named ruser_m. <p>Exists: (AXI_NUM_MASTERS >= x) && (AXI_HAS_RSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_R_LAYER_M(x)_S0 to AXI_R_LAYER_M(x)_S{AXI_NUM_SLAVES} = 1) and (AXI_R_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
rready_mx	I	<p>Master read ready. This signal indicates that the master can accept the read data and response information.</p> <ul style="list-style-type: none"> 1: Master ready 0: Master no ready <p>Exists: AXI_NUM_MASTERS >= x</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

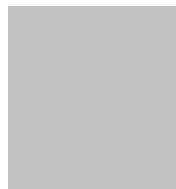
6.8 AXI Low Power Interface Signals

csysreq -  - csysack
- cactive

Table 6-8 AXI Low Power Interface Signals

Port Name	I/O	Description
csysreq	I	<p>System low-power request from system clock controller.</p> <ul style="list-style-type: none"> De-asserted by system low power controller (LPC) to initiate entry into a low power state. Asserted to initiate exit from a low power state. <p>Exists: (AXI_LOWPWR_HS_IF == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High</p>
csysack	O	<p>Low-power request acknowledgment.</p> <ul style="list-style-type: none"> De-asserted by DW_axi to acknowledge request to enter low-power state Asserted by DW_axi to acknowledge request to exit low-power state <p>Exists: (AXI_LOWPWR_HS_IF == 1) Synchronous To: aclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High</p>
cactive	O	<p>Clock active request. De-asserted by DW_axi to tell system low-power controller (LPC) that clock can be removed.</p> <ul style="list-style-type: none"> 1: peripheral clock required. 0: peripheral clock not required. <p>Note: Clock should be removed on positive edge after cactive and csysack signals are sampled low (0). Exists: (AXI_LOWPWR_HS_IF == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High</p>

6.9 Deadlock Notification Signals



- dlock_mst
- dlock_slv
- dlock_id
- dlock_wr
- dlock_irq

Table 6-9 **Deadlock Notification Signals**

Port Name	I/O	Description
dlock_mst[(AXI_LOG2_LCL_NM-1):0]	O	Lowest numbered deadlocked master. Exists: (AXI_DLOCK_NOTIFY_EN == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dlock_slv[(AXI_LOG2_NSP1-1):0]	O	Slave with which dlock_mst is deadlocked. Exists: (AXI_DLOCK_NOTIFY_EN == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dlock_id[(AXI_MIDW-1):0]	O	AXI ID of deadlocked transaction. Exists: (AXI_DLOCK_NOTIFY_EN == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dlock_wr	O	Asserted if deadlocked transaction is a write, deasserted if it is a read. Exists: (AXI_DLOCK_NOTIFY_EN == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

Table 6-9 Deadlock Notification Signals (Continued)

Port Name	I/O	Description
dlock_irq	O	Deadlock notification. Exists: (AXI_DLOCK_NOTIFY_EN == 1) Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

6.10 Slave Port j Write Address Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

awready_sj -

- awvalid_sj
- awaddr_sj
- awid_sj
- awlen_sj
- awsize_sj
- awburst_sj
- awlock_sj
- awcache_sj
- awprot_sj
- awsideband_sj
- awuser_sj
- awqos_sj
- awsnoop_sj
- awdomain_sj
- awbar_sj
- awregion_sj

Table 6-10 Slave Port j Write Address Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

Port Name	I/O	Description
awvalid_sj	O	<p>Slave write address valid. This signal indicates that a valid write address and control information are available.</p> <ul style="list-style-type: none"> ■ 1: Address and control information available. ■ 0: Address and control information unavailable. <p>The address and control information remain stable until the address acknowledge signal, awready, goes high.</p> <p>Default State (after Power On Reset): When AXI_AW_TMO is not Combinatorial or AXI_AW_PL_ARB is true, then the default value is low. When AXI_AW_TMO is Combinatorial and AXI_AW_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_SLAVES $\geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

Table 6-10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
awaddr_sj[(AXI_AW-1):0]	O	<p>Slave write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awid_sj[(AXI_SIDW-1):0]	O	<p>Slave write address ID. This signal is the identification tag for the write address group of signals.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes for non-QoS if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awlen_sj[(AXI_BLW-1):0]	O	<p>Slave burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The AXI protocol specifies this as a 4 bit value, but this width can be configured to 8 bits wide to support longer bursts up to 256 data beats.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awsiz_sj[2:0]	O	<p>Slave burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
awburst_sj[1:0]	O	<p>Slave burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awlock_sj[(((AXI_INTERFACE_TYPE>0)?1:2)-1):0]	O	<p>Slave lock type. This signal provides additional information about the atomic characteristics of the transfer. The signal width and functionality changes between the AXI3 and AXI4/ACE-Lite interface types; for more information, refer to the 'Locked Transfers' section in the DW_axi Databook.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awcache_sj[3:0]	O	<p>Slave cache type. This signal indicates the bufferable, cacheable/modifiable, write-through, write-back, and allocatable attributes of the transaction.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awprot_sj[2:0]	O	<p>Slave protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
awsideband_sj[(AXI_AW_SBW-1):0]	O	<p>Optional. Slave AXI3 sideband bus for write address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named awsideband_s. When the AXI4 or ACE-Lite interface is enabled, this signal is named awuser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_AWSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awuser_sj[(AXI_AW_SBW-1):0]	O	<p>Slave AXI4/ACE-Lite user bus for write address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named awsideband_s. When the AXI4 or ACE-Lite interface is enabled, this signal is named awuser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_AWSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
awqos_sj[3:0]	O	<p>Slave Quality of Service identifier, conveying priority information associated with each transaction at Write Address channel of slave port(j).</p> <p>Note:: If the DW_axi is configured with QoS enabled and the system design requires the QoS value to be propagated to the slave, the sideband signals in the other DesignWare Library AXI components can be used to propagate this value. If the QoS value does not need to be propagated, this output can be ignored.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_QOS = 1) && (AXI_INTERFACE_TYPE > 0)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awready_sj	I	<p>Slave write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.</p> <ul style="list-style-type: none"> ■ 1: Slave ready ■ 0: Slave not ready <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
awsnoop_sj[2:0]	O	<p>This signal indicates the transaction type for shareable slave write transactions.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-10 Slave Port j Write Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
awdomain_sj[1:0]	O	<p>This signal indicates the shareability domain of a slave write transaction.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awbar_sj[1:0]	O	<p>This signal indicates a slave write barrier transaction.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
awregion_sj[3:0]	O	<p>Slave region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AW_LAYER_S(j)_M1 to AXI_AW_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AW_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

6.11 Slave Port j Write Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals

wready_sj -

- wvalid_sj
- wid_sj
- wdata_sj
- wstrb_sj
- wlast_sj
- wsideband_sj
- wuser_sj

Table 6-11 Slave Port j Write Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals

Port Name	I/O	Description
wvalid_sj	O	<p>Slave write valid. This signal indicates that valid write data and strobes are available.</p> <ul style="list-style-type: none"> ■ 1: Write data and strobes available ■ 0: Write data and strobes unavailable <p>Default State (after Power On Reset): When AXI_W_TMO is not Combinatorial or AXI_W_PL_ARB is true, then the default value is low. When AXI_W_TMO is Combinatorial and AXI_W_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
wid_sj[(AXI_SIDW-1):0]	O	<p>Slave write ID tag. This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes for non-QoS if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-11 Slave Port j Write Data Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals (Continued)

Port Name	I/O	Description
wdata_sj[(AXI_DW-1):0]	O	<p>Slave write data.</p> <p>Exists: AXI_NUM_SLAVES $\geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
wstrb_sj[((AXI_DW/8)-1):0]	O	<p>Slave write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, wstrb_m(x) corresponds to wdata_m[(8 * x) + 7:(8 * x)].</p> <p>Exists: AXI_NUM_SLAVES $\geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
wlast_sj	O	<p>Slave write last. This signal indicates the last transfer in a write burst.</p> <p>Exists: AXI_NUM_SLAVES $\geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
wsideband_sj[(AXI_W_SBW-1):0]	O	<p>Optional. Slave AXI3 sideband bus for write data channel. The signal name changes between the AXI3 and the AXI4 or ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named wsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named wuser_s. <p>Exists: (AXI_NUM_SLAVES $\geq j$) && (AXI_HAS_WSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-11 Slave Port j Write Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
wuser_sj[(AXI_W_SBW-1):0]	O	<p>Optional. Slave AXI4/ACE-Lite user bus for write data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named wsideband_s. When the AXI4 or ACE-Lite interface is enabled, this signal is named wuser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_WSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_W_LAYER_S(j)_M1 to AXI_W_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_W_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
wready_sj	I	<p>Slave write ready. This signal indicates that the slave can accept the write data.</p> <ul style="list-style-type: none"> 1: Slave ready 0: Slave not ready <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.12 Slave Port j Write Response Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

bvalid_sj -
 bid_sj -
 bresp_sj -
 bsideband_sj -
 buser_sj -

Table 6-12 Slave Port j Write Response Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

Port Name	I/O	Description
bvalid_sj	I	<p>Slave write response valid. This signal indicates that valid write response data is available.</p> <ul style="list-style-type: none"> 1: Write response available 0: Write response unavailable <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
bid_sj[(AXI_SIDW-1):0]	I	<p>Slave response ID tag. This signal is the ID tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.</p> <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
bresp_sj[1:0]	I	<p>Slave write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.</p> <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: $((\text{AXI_B_TMO} == 1) \parallel (\text{AXI_B_TMO} == 2)) ? \text{Yes} : \text{No}$</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-12 Slave Port j Write Response Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
bsideband_sj[(AXI_B_SBW-1):0]	I	<p>Optional. Slave AXI3 sideband bus for write response channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named bsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named buser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_BSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_B_TMO == 1) (AXI_B_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
buser_sj[(AXI_B_SBW-1):0]	I	<p>Optional. Slave AXI4/ACE-Lite user bus for write response channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named bsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named buser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_BSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_B_TMO == 1) (AXI_B_TMO == 2)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
bready_sj	O	<p>Slave response ready. This signal indicates that the master can accept the response information.</p> <ul style="list-style-type: none"> 1: Master ready 0: Master not ready <p>Default State (after Power On Reset): When AXI_B_TMO is not Combinatorial or AXI_B_PL_ARB is true, then the default value of bready_s(j) is high. When AXI_B_TMO is Combinatorial and AXI_B_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.13 Slave Port j Read Address Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

arready_sj -		arvalid_sj
slv_priority_sj -		arid_sj
tz_secure_sj -		araddr_sj
		arlen_sj
		arsize_sj
		arburst_sj
		arlock_sj
		arcache_sj
		arprot_sj
		arsideband_sj
		aruser_sj
		arqos_sj
		arsnoop_sj
		ardomain_sj
		arbar_sj
		arregion_sj

Table 6-13 Slave Port j Read Address Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

Port Name	I/O	Description
arvalid_sj	O	<p>Slave read address valid. This signal indicates that a valid read address and control information are available.</p> <ul style="list-style-type: none"> 1: Address and control information available 0: Address and control information unavailable <p>Default State (after Power On Reset): When AXI_AR_TMO is not Combinatorial or AXI_AR_PL_ARB is true, then the default value is low. When AXI_AR_TMO is Combinatorial and AXI_AR_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi. The address and control information remain stable until the address acknowledge signal, arready, goes high.</p> <p>Exists: AXI_NUM_SLAVES $\geq j$</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

Table 6-13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
arid_sj[(AXI_SIDW-1):0]	O	<p>Slave read address ID. This signal is the identification tag for the read address group of signals.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
araddr_sj[(AXI_AW-1):0]	O	<p>Slave read address. The read address bus gives the address of the first transfer in a read burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arlen_sj[(AXI_BLW-1):0]	O	<p>Slave burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The AXI protocol specifies this as a 4 bit value, but this width can be configured to 8 bits wide to support longer bursts up to 256 data beats.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arsize_sj[2:0]	O	<p>Slave burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
arburst_sj[1:0]	O	<p>Slave burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arlock_sj[(((AXI_INTERFACE_TYPE>0)?1:2)-1):0]	O	<p>Slave lock type. This signal provides additional information about the atomic characteristics of the transfer. The signal width and functionality changes between the AXI3 and AXI4/ACE-Lite interface types; for more information, refer to the 'Locked Transfers' section in the DW_axi Databook.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arcache_sj[3:0]	O	<p>Slave read cache type. This signal indicates the bufferable, cacheable/modifiable, write-through, write-back, and allocatable attributes of the transaction.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arprot_sj[2:0]	O	<p>Slave protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
arsideband_sj[(AXI_AR_SBW-1):0]	O	<p>Optional. Slave AXI3 sideband bus for read address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named arsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named aruser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_ARSB = 1) && (AXI_INTERFACE_TYPE=AXI3)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
aruser_sj[(AXI_AR_SBW-1):0]	O	<p>Optional. Slave AXI4 or ACE-Lite user bus for read address channel. The signal name changes between the AXI3 and the AXI4/ACELite configurations.</p> <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named arsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named aruser_s. <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_ARSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
arqos_sj[3:0]	O	<p>Slave Quality of Service identifier, conveying priority information associated with each transaction at read address channel of slave port(j).</p> <p>Note:: If the DW_axi is configured with QoS enabled and the system design requires the QoS value to be propagated to the slave, the sideband signals in the other DesignWare Library AXI components can be used to propagate this value. If the QoS value does not need to be propagated, this output can be ignored.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_QOS = 1) && (AXI_INTERFACE_TYPE > 0)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arready_sj	I	<p>Slave read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.</p> <ul style="list-style-type: none"> ■ 1: Slave ready ■ 0: Slave not ready <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
arsnoop_sj[3:0]	O	<p>This signal indicates the transaction type for shareable slave read transactions.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-13 Slave Port j Read Address Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
ardomain_sj[1:0]	O	<p>This signal indicates the shareability domain of a slave read transaction.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arbar_sj[1:0]	O	<p>This signal indicates a slave read barrier transaction.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
arregion_sj[3:0]	O	<p>Slave region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_INTERFACE_TYPE = ACELITE) && (AXI_HAS_REGIONS_S(j) = 1)</p> <p>Synchronous To: aclk</p> <p>Registered: Yes if (AXI_AR_LAYER_S(j)_M1 to AXI_AR_LAYER_S(j)_M{AXI_NUM_MASTERS} = 1) and (AXI_AR_SHARED_PL = 1) or No otherwise</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
slv_priority_sj[(AXI_SLV_PRIORITY_W-1):0]	I	<p>Slave arbitration priority associated with Slave Port(j). This priority can be changed during operation of the component. Legal values on this input are all bits zero (0) up to all bits 1. It is not recommended to set all bits to 0, because that setting is used by the default slave (the lowest priority slave). All bits one (1) is considered highest priority.</p> <p>Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_EXT_PRIORITY = 1)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 6-13 Slave Port j Read Address Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals (Continued)

Port Name	I/O	Description
tz_secure_sj	I	<p>Slave trustzone signal. This signal is asserted if the attached slave is a secure slave; only secure accesses can be forwarded to secure slaves.</p> <p>Exists: ($\text{AXI_NUM_SLAVES} \geq j$) && ($\text{AXI_HAS_TZ_SUPPORT} = 1$)</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.14 Slave Port j Read Data Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

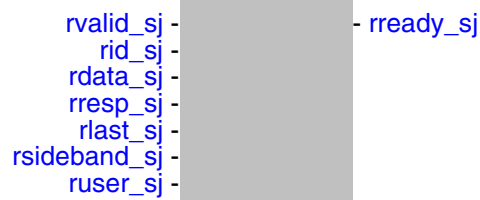


Table 6-14 Slave Port j Read Data Channel (for $j = 1; j \leq \text{AXI_NUM_SLAVES}$) Signals

Port Name	I/O	Description
rvalid_sj	I	<p>Read valid. This signal indicates that the required read data is available and the read transfer can complete.</p> <ul style="list-style-type: none"> 1: read data available 0: read data not available <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$ Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High</p>
rid_sj[(AXI_SIDW-1):0]	I	<p>Read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.</p> <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$ Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A</p>
rdata_sj[(AXI_DW-1):0]	I	<p>Read data.</p> <p>Exists: $\text{AXI_NUM_SLAVES} \geq j$ Synchronous To: aclk Registered: $((\text{AXI_R_TMO} == 1) \parallel (\text{AXI_R_TMO} == 2)) ? \text{Yes} : \text{No}$ Power Domain: SINGLE_DOMAIN Active State: N/A</p>

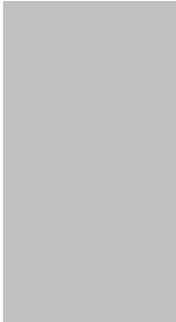
Table 6-14 Slave Port j Read Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
rresp_sj[1:0]	I	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. Exists: AXI_NUM_SLAVES >= j Synchronous To: aclk Registered: ((AXI_R_TMO == 1) (AXI_R_TMO == 2)) ? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A
rlast_sj	I	Read last. This signal indicates the last transfer in a read burst. Exists: AXI_NUM_SLAVES >= j Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
rsideband_sj[(AXI_R_SBW-1):0]	I	Optional. AXI3 sideband bus for read data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations. <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named rsideband_s. When the AXI4 or ACE-Lite interface is enabled, this signal is named ruser_s. Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_RSB = 1) && (AXI_INTERFACE_TYPE=AXI3) Synchronous To: aclk Registered: ((AXI_R_TMO == 1) (AXI_R_TMO == 2)) ? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A
ruser_sj[(AXI_R_SBW-1):0]	I	Optional. AXI4/ACE-Lite user bus for read data channel. The signal name changes between the AXI3 and the AXI4/ACE-Lite configurations. <ul style="list-style-type: none"> When the AXI3 interface is enabled, this signal is named rsideband_s. When the AXI4/ACE-Lite interface is enabled, this signal is named ruser_s. Exists: (AXI_NUM_SLAVES >= j) && (AXI_HAS_RSB = 1) && (AXI_INTERFACE_TYPE=AXI4/ACE-Lite) Synchronous To: aclk Registered: ((AXI_R_TMO == 1) (AXI_R_TMO == 2)) ? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A

Table 6-14 Slave Port j Read Data Channel (for j = 1; j <= AXI_NUM_SLAVES) Signals (Continued)

Port Name	I/O	Description
rready_sj	O	<p>Read ready. This signal indicates that the master can accept the read data and response information.</p> <ul style="list-style-type: none"> ■ 1: Master ready ■ 0: Master not ready <p>Default State (after Power On Reset): When AXI_R_TMO is not Combinatorial or AXI_R_PL_ARB is true, then the default value is high. When AXI_R_TMO is Combinatorial and AXI_R_PL_ARB is false, then the default value is a function of the inputs and not the state of the DW_axi.</p> <p>Exists: AXI_NUM_SLAVES >= j</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.15 Default Slave Write Address Channel Signals



- dbg_awid_s0
- dbg_awaddr_s0
- dbg_awlen_s0
- dbg_awsizs_s0
- dbg_awburst_s0
- dbg_awlock_s0
- dbg_awcache_s0
- dbg_awprot_s0
- dbg_awvalid_s0
- dbg_awready_s0

Table 6-15 Default Slave Write Address Channel Signals

Port Name	I/O	Description
dbg_awid_s0[(AXI_SIDW-1):0]	O	dbg_awid_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awaddr_s0[(AXI_AW-1):0]	O	dbg_awaddr_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awlen_s0[(AXI_BLW-1):0]	O	dbg_awlen_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

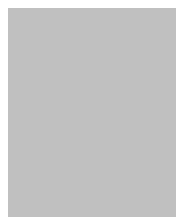
Table 6-15 Default Slave Write Address Channel Signals (Continued)

Port Name	I/O	Description
dbg_awsizs_s0[2:0]	O	dbg_awsizs_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awburst_s0[1:0]	O	dbg_awburst_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awlock_s0[(((AXI_INTERFACE_TY PE>0)?1:2)-1):0]	O	dbg_awlock_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awcache_s0[3:0]	O	dbg_awcache_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_awprot_s0[2:0]	O	dbg_awprot_s0 - See the corresponding description for 'Slave Port j Write Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

Table 6-15 Default Slave Write Address Channel Signals (Continued)

Port Name	I/O	Description
dbg_awvalid_s0	O	<p>dbg_awvalid_s0 - See the corresponding description for 'Slave Port j Write Address Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: ((AXI_AW_SHARED_PL == 1) && (AXI_AW_PL_ARB == 0) && (AXI_AW_LAYER_S0_M1 == 0 && (AXI_NV_S0_BY_M1 AXI_BV_S0_BY_M1)))? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
dbg_awready_s0	O	<p>dbg_awready_s0 - See the corresponding description for 'Slave Port j Write Address Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: None</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.16 Default Slave Write Data Channel Signals



- dbg_wid_s0
- dbg_wdata_s0
- dbg_wstrb_s0
- dbg_wlast_s0
- dbg_wvalid_s0
- dbg_wready_s0

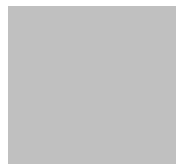
Table 6-16 Default Slave Write Data Channel Signals

Port Name	I/O	Description
dbg_wid_s0[(AXI_SIDW-1):0]	O	<p>dbg_wid_s0 - See the corresponding description for 'Slave Port j Write Data Channel'. dbg_wid_s0 is not valid when AXI_INTERFACE_TYPE = AXI4/ACELITE.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
dbg_wdata_s0[(AXI_DW-1):0]	O	<p>dbg_wdata_s0 - See the corresponding description for 'Slave Port j Write Data Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
dbg_wstrb_s0[((AXI_DW/8)-1):0]	O	<p>dbg_wstrb_s0 - See the corresponding description for 'Slave Port j Write Data Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
dbg_wlast_s0	O	<p>dbg_wlast_s0 - See the corresponding description for 'Slave Port j Write Data Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

Table 6-16 Default Slave Write Data Channel Signals (Continued)

Port Name	I/O	Description
dbg_wvalid_s0	O	<p>dbg_wvalid_s0 - See the corresponding description for 'Slave Port j Write Data Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: (((AXI_W_LAYER_S0_M1 == 0) && (AXI_NV_S0_BY_M1 AXI_BV_S0_BY_M1)) && (AXI_W_MCA_NC_S0 == 1) && (AXI_W_PL_ARB == 0)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
dbg_wready_s0	O	<p>dbg_wready_s0 - See the corresponding description for 'Slave Port j Write Data Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: Yes</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.17 Default Slave Write Response Channel Signals

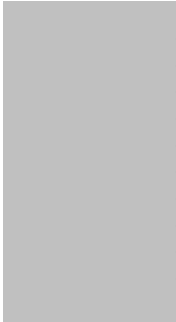


- dbg_bid_s0
- dbg_bresp_s0
- dbg_bvalid_s0
- dbg_bready_s0

Table 6-17 Default Slave Write Response Channel Signals

Port Name	I/O	Description
dbg_bid_s0[(AXI_SIDW-1):0]	O	dbg_bid_s0 - See the corresponding description for 'Slave Port j Write Response Channel'. Exists: Always Synchronous To: aclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_bresp_s0[1:0]	O	dbg_bresp_s0 - See the corresponding description for 'Slave Port j Write Response Channel'. Exists: Always Synchronous To: None Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_bvalid_s0	O	dbg_bvalid_s0 - See the corresponding description for 'Slave Port j Write Response Channel'. Exists: Always Synchronous To: aclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High
dbg_bready_s0	O	dbg_bready_s0 - See the corresponding description for 'Slave Port j Write Response Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

6.18 Default Slave Read Address Channel Signals



- dbg_arid_s0
- dbg_araddr_s0
- dbg_arlen_s0
- dbg_arsize_s0
- dbg_arburst_s0
- dbg_arlock_s0
- dbg_arcache_s0
- dbg_arprot_s0
- dbg_arvalid_s0
- dbg_arready_s0

Table 6-18 Default Slave Read Address Channel Signals

Port Name	I/O	Description
dbg_arid_s0[(AXI_SIDW-1):0]	O	dbg_arid_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_araddr_s0[(AXI_AW-1):0]	O	dbg_araddr_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_arlen_s0[(AXI_BLW-1):0]	O	dbg_arlen_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

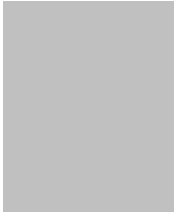
Table 6-18 Default Slave Read Address Channel Signals (Continued)

Port Name	I/O	Description
dbg_arsize_s0[2:0]	O	dbg_arsize_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_arburst_s0[1:0]	O	dbg_arburst_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_arlock_s0[(((AXI_INTERFACE_TYPE > 0)?1:2)-1):0]	O	dbg_arlock_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_arcache_s0[3:0]	O	dbg_arcache_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_arprot_s0[2:0]	O	dbg_arprot_s0 - See the corresponding description for 'Slave Port j Read Address Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

Table 6-18 Default Slave Read Address Channel Signals (Continued)

Port Name	I/O	Description
dbg_arvalid_s0	O	<p>dbg_arvalid_s0 - See the corresponding description for 'Slave Port j Read Address Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: (((AXI_AR_LAYER_S0_M1 == 0) && (AXI_NV_S0_BY_M1 AXI_BV_S0_BY_M1)) && (AXI_AR_MCA_NC_S0 == 1) && (AXI_AR_PL_ARB == 0)) ? Yes : No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
dbg_arready_s0	O	<p>dbg_arready_s0 - See the corresponding description for 'Slave Port j Read Address Channel'.</p> <p>Exists: Always</p> <p>Synchronous To: aclk</p> <p>Registered: Yes</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

6.19 Default Slave Read Data Channel Signals



- dbg_rid_s0
- dbg_rdata_s0
- dbg_rresp_s0
- dbg_rvalid_s0
- dbg_rlast_s0
- dbg_rready_s0

Table 6-19 Default Slave Read Data Channel Signals

Port Name	I/O	Description
dbg_rid_s0[(AXI_SIDW-1):0]	O	dbg_rid_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: aclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_rdata_s0[(AXI_DW-1):0]	O	dbg_rdata_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: None Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_rresp_s0[1:0]	O	dbg_rresp_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: None Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
dbg_rvalid_s0	O	dbg_rvalid_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: aclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High

Table 6-19 Default Slave Read Data Channel Signals (Continued)

Port Name	I/O	Description
dbg_rlast_s0	O	dbg_rlast_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
dbg_rready_s0	O	dbg_rready_s0 - See the corresponding description for 'Slave Port j Read Data Channel'. Exists: Always Synchronous To: aclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

7

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 7-1 Internal Parameters

Parameter Name	Equals To
AXI_B_HAS_SHARED_LAYER	{[::DW_axi::calcSharedChannelUsedSlv B M S]}
AXI_BV_S0_BY_M1	=((AXI_NUM_SLAVES >= 0) && (AXI_NUM_MASTERS >= 1) && (AXI_REMAP_EN == 1))
AXI_HAS_APB3	0
AXI_HAS_ICM1	=(AXI_NUM_ICM >= 1 ? 1 : 0)
AXI_IDW_M1	=(AXI_HAS_ICM1 ? AXI_SIDW : AXI_MIDW)
AXI_INITIAL_LOCKDOWN	0
AXI_LOG2_LCL_NM	{[::DW_axi::calc_num_in_log2 AXI_NUM_MASTERS]}
AXI_LOG2_NSP1	{[::DW_axi::calc_nump1_in_log2 AXI_NUM_SLAVES]}
AXI_MAX_SBW	256
AXI_MAX_UIDA	64
AXI_MST_PRIORITY_W	AXI_LOG2_LCL_NM
AXI_NV_S0_BY_M1	=((AXI_NUM_SLAVES >= 0) && (AXI_NUM_MASTERS >= 1))

Table 7-1 Internal Parameters (Continued)

Parameter Name	Equals To
AXI_R_HAS_SHARED_LAYER	{[::DW_axi::calcSharedChannelUsedSlv R M S]}
AXI_SHARED_LAYER_MASTER_PRIORITY_EN_VAL	{[::DW_axi::calcShrdMstPriEn]}
AXI_SHARED_LAYER_SLAVE_PRIORITY_EN_VAL	{[::DW_axi::calcShrdSlvPriEn]}
AXI_SLV_PRIORITY_W	AXI_LOG2_NSP1
AXI_W_HAS_SHARED_LAYER	{[::DW_axi::calcSharedChannelUsedMst W S M]}
DW_AXI_VERSION_ID	32'h3430342a

8

Verification

This chapter provides an overview of the testbench available for DW_axi verification. DW_axi has been functionally verified using the following verification environments:

- Verilog Test Environment (VTE) – uses Synopsys Verification IP for AMBA
- System Verilog Test Environment (SVTE) – uses Synopsys Discovery Verification IP for AMBA

Once you have configured DW_axi in coreConsultant and have set up the verification environment, you can automatically run simulations.



Note

- The SVTE testbench is a readable verification environment that can be extended and re-used in a test environment.
- The VTE environment is not visible through coreConsultant and it is used for internal regressions.

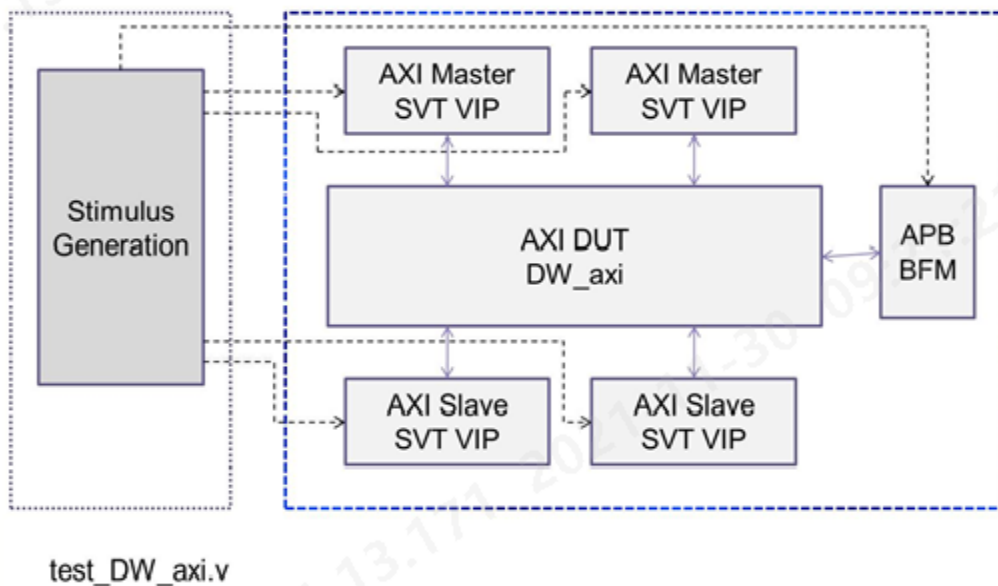
8.1 Overview of Test Environments

To test the DW_axi, you can use the SystemVerilog Test Environment (SVTE) using Synopsys Discovery Verification IP for AMBA.

8.1.1 SystemVerilog Test Environment (SVTE)

The SystemVerilog Test Environment (SVTE) is a SystemVerilog-directed testbench that uses the Synopsys Discovery Verification IP for AMBA to test DW_axi.

Figure 8-1 shows the block diagram of the SystemVerilog Test Environment (SVTE).

Figure 8-1 SVTE Block Diagram

If QoS is enabled in the DW_axi:

- AXI Verification IP `awqos/arqos` signals are used to drive the QoS ports of DW_axi. This is applicable only in AXI4 configurations. In AXI3 configurations, QoS ports are driven to 0, due to a limitation in the AXI Verification IP.
- APB Master BFM is used to program QoS registers in DW_axi.

There are VIP wrapper tasks as well as tasks for masters, slaves, and APB-related tasks.

8.1.2 VIP Wrapper Tasks

VIP wrapper tasks are the wrappers for VIP commands used for particular instance of the VIP. Each instance of the VIP is identified by two parameters – ‘`transactor`’ and ‘`xactor_id`’.

`transactor`: Specifies the type of VIP instance (master, slave)

`xactor_id`: Specifies the id of the instance (1 to 16)

For example, if `transactor` = “master”, and `xactor_id` = 1, the operation corresponds to master[1] instance of the VIP.

8.1.3 SVTE Task Usage

The tasks used by the SVTE to create the tests are designed to be reusable and flexible, to enable you to quickly create your own tests. For example, to create and send two randomly configured write transactions from masters 1 and 2 to slaves 1 and 2 respectively:

```
...
master_id=1; slave_id=1; region=1;
axi_master_rand_xact(master_id,slave_id,region,`SIM_WRITE`,`SIM_LOCK_RND`,`SIM_SECURE_RND`,`
SIM_BURST_RND,1,handle);
masterID=2; slaveID=2; write=1;
```

```

    axi_master_rand_xact(master_id,slave_id,region,`SIM_WRITE,`SIM_LOCK_RND,`SIM_SECURE_RND,`
    SIM_BURST_RND,1,handle);
    ...

```

The VIP tasks for the slaves and masters models are contained in the following Verilog files:

- tb_axi_tasks_mst.v
- tb_axi_tasks_slv.v

The VIP tasks that are common across the master/slave models are contained in the tb_axi_tasks_vip.v Verilog file.

8.1.4 SVTE APB Usage

When QoS is enabled, the SVTE uses APB-related tasks to configure all registers.

For example, to configure the registers through the APB interface, use the following task:

```

...
master_num=1;
program_qos_registers(master_num,1,burstiness,peak_rate_value,qos_value,slv_rd_value))
master_num=4;
program_qos_registers(master_num,1,burstiness,peak_rate_value,qos_value,slv_rd_value))
...

```

Use this task to configure the following QoS values:

- burstiness
- peak rate
- qos_value
- slv_rdy

Each master must have its own program_qos_registers task. To configure registers for n masters, the task is called n times.

8.1.5 SVTE Tasks

The common VIP tasks are described as follows:

Table 8-1 SVTE Tasks

vip_start	
Task Function	Start the run flow of a particular instance of the VIP.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.

Table 8-2

vip_stop	
Task Function	Start the run flow of a particular instance of the VIP.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.

vip_new_data	
Task Function	Used to create new data for a particular instance of the VIP.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
data_type	The input string identifier of the new data.
handle	The output integer handle of the new data object resulting from the new_data operation.

vip_apply_data	
Task Function	Used to accept the contents of a data object and apply them to the model.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
handle	The input integer handle of the data object.

vip_delete_data	
Task Function	Used to delete the handle associated with an active data object.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
handle	The input integer handle of the data object.

vip_copy_data	
Task Function	Instructs the model to make a (deep) copy of the data object referenced by the src_handle argument, store it as a new data object, and assign the handle var argument with a pointer to the new copy.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
src_handle	Input handle to point object of VIP instance from which copy is made.
handle	The input integer handle of the data object.

vip_display_data	
Task Function	Instructs the transactor to display (to log or transcript) the contents of the data object associated with the given handle.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
handle	The input handle to point to the object of the instance of the VIP.
str	This corresponds to the input comment to be displayed in log.

vip_set_data_prop	
Task Function	Sets the value provided by the prop_val argument of a specified public property, in the data object pointed to by the handle argument.
Arguments	
transactor	The input string corresponding to the VIP instance name
xactor_id	The input transactor id.
handle	Input instance handle of the data object.
prop_name	Input property name of data object like ID, addr, xact_type, and so on.
prop_val	Input property value that needs to be set.
array_ix	Input array index for array properties like data, wstrb, and so on.
is_valid	Returned is_valid value. 1 if successful, 0 if unsuccessful.

vip_get_data_prop	
Task Function	Gets the value provided of a specified public property, in the data object pointed to by the handle argument.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
handle	Input instance handle of the data object
prop_name	Input property name of data object like ID, addr, xact_type, and so on.
prop_val	Output property value that needs to be set.
array_ix	Input array index for array properties like data, wstrb etc.
is_valid	Returned is_valid value. 1 if successful, 0 if unsuccessful.

vip_notify_wait_for	
Task Function	When called from the command, does not return until the specified event occurs within the model.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
event_name	The name of an event.
is_valid	Returned is_valid value. 1 If successful, 0 if unsuccessful.

vip_callback_wait_for	
Task Function	When called from the command, does not return until the specified callback event occurs within the model and returns the handle associated with the callback.
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
cb_notify_name	Input callback notification name.
handle	The handle to point to the data object of the VIP instance.
is_valid	This flags the status of function executed; 1 if success, 0 if fail.

vip_callback_proceed	
Task Function	When called from the command, instructs the model to proceed with processing after the specified callback point
Arguments	
transactor	The input string corresponding to the VIP instance name.
xactor_id	The input transactor id.
cb_notify_name	Input callback notification name.
handle	The handle to point to the data object of the VIP instance.
is_valid	This flags the status of function executed; 1 if success, 0 if fail.

8.1.5.1 Master Tasks

The master tasks are described as follows:

Table 8-3 Master Tasks in SVTE Tasks

axi_master_rand_xact	
Task Function	Generates a random AXI transaction.
Arguments	
master_id	Selects the master instance to issue the transaction.
slave_id	Selects the slave instance to be targeted by the master.
region_id	Selects the slave region to be targeted by the master.
write	<ul style="list-style-type: none"> ■ Read - 0 ■ Write - 1 ■ Randomly select - 2
lock	<ul style="list-style-type: none"> ■ Unlock - 0 ■ Lock - 1 ■ Randomly select - 2
secure	<ul style="list-style-type: none"> ■ Secure - 0 ■ Nonsecure - 1 ■ Randomly select - 2
burst	<ul style="list-style-type: none"> ■ MultiBurst - 0 ■ SingleBurst - 1 ■ Random - 2 ■ INCR16 - 3

Table 8-3 Master Tasks in SVTE Tasks

axi_master_rand_xact	
slv_addr_dcd	Used when doing unaligned or default slave testing.
xfer_handle	Returns the transaction handle created. All other aspects of the transaction (burst length, cache type, and so on) are generated randomly within the constraints of the AXI protocol.

8.1.5.2 Slave Tasks

The slave tasks are described as follows:

Table 8-4 Slave Tasks in SVTE Tasks

axi_slave_rand_xact	
Task Function	Generates the random slave response.
Arguments	
slave_id	Selects the slave instance to respond to the transaction.
resp_type	Okay, error, or random.
xfer_handle	Returns the transaction handle of slave response.

axi_slave_send_rand_response	
Task Function	This task waits for the response request, randomizes the transaction, applies to the model, and waits till it is consumed.
Arguments	
slave_id	Selects the slave instance to respond to the transaction.

8.1.5.3 APB BFM Tasks

The APB tasks are described as follows:

Table 8-5 APB BFM Tasks in SVTE Tasks

apb_read	
Task Function	Generates appropriate reads on the APB bus. The number of reads/writes are automatically managed based on the configurable APB data width.
Arguments	
apb_addr	Register address to be accessed.
apb_rd_data	The read value of the addressed register.

apb_write	
Task Function	Generates appropriate reads on the APB bus. The number of reads/writes are automatically managed based on the configurable APB data width.
Arguments	
apb_addr	Register address to be accessed.
apb_rd_data	The read value of the addressed register.

program_qos_registers	
Task Function	The task is used to configure each master's channels with the corresponding QoS register values. The register values are written through the APB interface, depending on the RTL configuration. The write commands are automatically generated and registers updated.
Arguments	
masterID	Selects the master to configure.
rd_wr_chan	Specifies whether the transaction is for the address read (AR) or address write (AW) channel.
qos_cmd_burst	Value for burstiness and enable / disable rate regulation.
qos_cmd_peak_rate	Value for peak and transaction rate.
qos_cmd_qos_value	Specify QOS value.
qos_cmd_slv_rdy	Enable/disable the SLV_RDY signal without waiting for interconnect.

command_reg_write	
Task Function	This task allows you to program any of the available QoS registers individually. The usage is on a per master basis for the corresponding AR or AW channel.
Arguments	
masterID	Selects the master to configure.
qos_cmd	Specifies the QoS command register to be updated. <ul style="list-style-type: none"> ■ 3'b000 – Burstiness/regulator enable ■ 3'b001 – Peak/transaction rate. ■ 3'b010 – QoS value ■ 3'b011 – SLV_RDY
data	The value to be updated.

command_reg_write	
rd_wr_chan	Specifies whether the transaction is for the address read (AR) or address write (AW) channel.

9

Integration Considerations

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

9.1 Performance

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

9.1.1 Power Consumption, Frequency, Area and DFT Coverage

[Table 9-1](#) provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_axi using the industry standard 16nm technology library.

Table 9-1 Synthesis Results for DW_axi

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Default Configuration	aclk=400M Hz	8327	0.283 mW	7 nW	100	99.9	99.4

Table 9-1 Synthesis Results for DW_axi

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Typical Configuration 1 AXI_ALL_AR_LAYER_SHARED = 1 AXI_ALL_AW_LAYER_SHARED = 1 AXI_ALL_B_LAYER_SHARED = 1 AXI_AR_TMO = 1 AXI_AW = 32 AXI_AW_TMO = 1 AXI_B_TMO = 1 AXI_DW = 32 AXI_HAS_LOCKING = 0 AXI_HAS_TZ_SUPPORT = 0 AXI_HAS_XDCDR = 0 AXI_NUM_MASTERS = 12 AXI_NUM_SLAVES = 12 AXI_REMAP_EN = 0 AXI_R_TMO = 1 AXI_W_TMO = 1 USE_FOUNDATION = 1	aclk=400M Hz	88384	3.09 mW	70 nW	100	99.93	99.7

Table 9-1 Synthesis Results for DW_axi

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Typical Configuration 2 AXI_ACC_NON_LCL_SLV_S10 = 1 AXI_ACC_NON_LCL_SLV_S16 = 1 AXI_ACC_NON_LCL_SLV_S2 = 1 AXI_ACC_NON_LCL_SLV_S9 = 1 AXI_ALLOW_MST1_ICM1 = 2 AXI_ALLOW_MST1_ICM2 = 1 AXI_ALL_AW_LAYER_SHARED = 1 AXI_ALL_R_LAYER_SHARED = 1 AXI_ALL_W_LAYER_SHARED = 1 AXI_AR_ARB_TYPE_S(j) = 4 AXI_AR_HAS_QOS_REGULATOR_M(x) = 1 AXI_AR_SBW = 256 AXI_AR_TMO = 1 AXI_AW_ARB_TYPE_S(j) = 4 AXI_AW_HAS_QOS_REGULATOR_M(x) = 1 AXI_AW_LAYER_S1_M1 = 1 AXI_AW_LAYER_S1_M2 = 1 AXI_AW_PL_ARB = 1 AXI_AW_SBW = 256 AXI_AW_SHARED_PL = 1 AXI_AW_TMO = 2 AXI_B_ARB_TYPE_M1 = 2 AXI_B_SBW = 256 AXI_HAS_ARQOS_EXT_M(x) = 1 AXI_HAS_ARSB = 1 AXI_HAS_AWQOS_EXT_M(x) = 1 AXI_HAS_AWSB = 1 AXI_HAS_BICMD = 1 AXI_HAS_BSB = 1 AXI_HAS_QOS = 1 AXI_HAS_REGIONS_S(j) = 1 AXI_HAS_RSB = 1 AXI_HAS_TZ_SUPPORT = 1 AXI_HAS_WSB = 1 AXI_INTERFACE_TYPE = 1 AXI_NUM_ICM = 2	aclk=400M Hz pclk=100M Hz	306135	12.7 mW	200 nW	100	99.9	99.9

Table 9-1 Synthesis Results for DW_axi

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
AXI_NUM_MASTERS = 4 AXI_NUM_MST_THRU_ICM1 = 1 AXI_NUM_MST_THRU_ICM2 = 1 AXI_NUM_RB_S(j) = 8 AXI_NUM_RN_S(j) = 8 AXI_NUM_SLAVES = 16 AXI_NUM_SYS_MASTERS = 16 AXI_REMAP_EN = 1 AXI_R_ARB_TYPE_M1 = 2 AXI_R_LAYER_M1_S1 = 1 AXI_R_LAYER_M1_S2 = 1 AXI_R_SBW = 256 AXI_R_SHARED_PL = 1 AXI_R_TMO = 2 AXI_W_ARB_TYPE_S1 = 3 AXI_W_LAYER_S1_M1 = 1 AXI_W_LAYER_S1_M2 = 1 AXI_W_PL_ARB = 1 AXI_W_SBW = 256 AXI_W_SHARED_PL = 1 AXI_W_TMO = 2							
Typical Configuration 3 AXI_ALL_AR_LAYER_SHARED = 1 AXI_ALL_AW_LAYER_SHARED = 1 AXI_ALL_B_LAYER_SHARED = 1 AXI_ALL_R_LAYER_SHARED = 1 AXI_ALL_W_LAYER_SHARED = 1 AXI_NUM_MASTERS = 16 AXI_NUM_SLAVES = 16 USE_FOUNDATION = 1	acclk=400M Hz	61757	1.61 mW	50 nW	99.99	99.83	99.8

9.1.2 Latency

Table 9-2 lists best-case latencies in the DW_axi bridge under these circumstances:

- Forward Registered timing option selected for all channels
- All other configuration parameters at default values

Best-case assumes zero delay in the slave response.

Table 9-2 DW_axi Best-Case Latencies

Channel	Description	Number of ack Cycles
Write Address	AWVALID received on master port to AWVALID issued on slave port — awvalid_m to awvalid_s	1
Write Data (master asserts awvalid and wvalid in same cycle)	AWVALID received on master port to first beat of WDATA issued on slave port — awvalid_m to first beat of wdata_s	2
Write Data (master asserts wvalid one cycle after awvalid)	AWVALID received on master port to first beat of WDATA issued on slave port — awvalid_m to first beat of wdata_s	3
Read Address	RVALID received on master port to ARVALID issued on slave port — arvalid_m to arvalid_s	1
Read Data	ARVALID received on master port to first beat of RDATA issued on master port — arvalid_m to first beat of rdata_m	3
Write Response	BVALID received on slave port to BVALID issued on master Port — bvalid_s to bvalid_m	1

9.2 Usage Requirements

The following details a list of usage requirements for the DW_axi, in order for it to work as specified.

1. Write data interleaving – An external master is restricted to having a write interleaving depth of 1. You can configure the DW_axi to interleave write data from multiple masters to a single slave, but an external master is required to always send write data in the order it issued the write transactions in. The behavior of the DW_axi is unspecified if this rule is broken. This rule does not apply for a single-master system.
2. For single-master subsystems – The DW_axi places no restrictions on write data interleaving of the external master. The external master must ensure that it does not interleave write data to a depth not supported by an external slave. The DW_axi does not explicitly impose such a rule for a single-master AXI system.
3. If an external master wants to have multiple outstanding active transactions to different slaves, then it must generate these transactions with different ARID/AWID values. For more information, refer to [“Out-of-Order Deadlock Avoidance”](#) on page 54.
4. Lock sequence – AXI lock sequences cannot cross slave address boundaries, including crossing into the default slave. AXI locked sequences will break if you attempt to do this.
5. If the combinatorial timing option is configured for any AXI channel, then in order to avoid a combinatorial feedback loop, the external AXI master and external AXI slave must break the timing path between AXI channel flow control signals *valid and *ready.
6. External decoder – If the external decoder option is selected then:
 - a. Slave number output must use the same index as the DW_axi uses
 - b. Combinatorial timing path between address inputs and slave number output

7. Toggling – While an external master's read/write address channel is waited, toggling of `remap_n` or `tz_secure_s(j)` is not permitted. The `remap_n` or `tz_secure_s(j)` signals can toggle only after DW_axi accepts the master's read/write command. Toggling the `remap_n` or `tz_secure_s(j)` signals while an external master's command channel is waited may cause an AXI protocol violation.

**Note**

Toggling the `remap` or `tz_secure_s(j)` signals after DW_axi accepts a command, but before the active command has completed, may not be desirable behavior; however, it does not cause a protocol violation.

9.3 Timing Exceptions

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

9.4 High Frequency Design Methods

When the critical timing path in the AXI subsystem includes DW_axi, there are multiple methods allowed by the AXI protocol to break that combinatorial path. It is often possible to achieve improved AXI subsystem performance by achieving higher clock frequency at the expense of additional area or increasing an isolated channel's latency.

If the critical timing path in the subsystem is a path through DW_axi, then timing mode options can be used to break the combinatorial path within DW_axi. After applying DW_axi timing mode options, if the critical path still includes inputs or outputs of DW_axi, then an external register slice (`DW_axi_rs`) can be used to further break those long timing paths.

For very large subsystems, with many masters and slaves attached to DW_axi, it's possible that the critical path for the subsystem could be a registered path starting or ending at DW_axi. This may be true even after applying the DW_axi timing mode options. In this situation, it is possible to improve this DW_axi timing by using interconnect tiling.

Another method that can be used to reduce registered timing paths starting or ending at DW_axi is Slave Visibility configuration. By limiting the number of slaves visible to a master, the complexity of interconnect tasks is reduced, and therefore, timing paths are often faster. Slave Visibility is not discussed further in this section, but is described in this databook on [“Slave Visibility”](#) on page 43.

9.4.1 Timing Mode Options

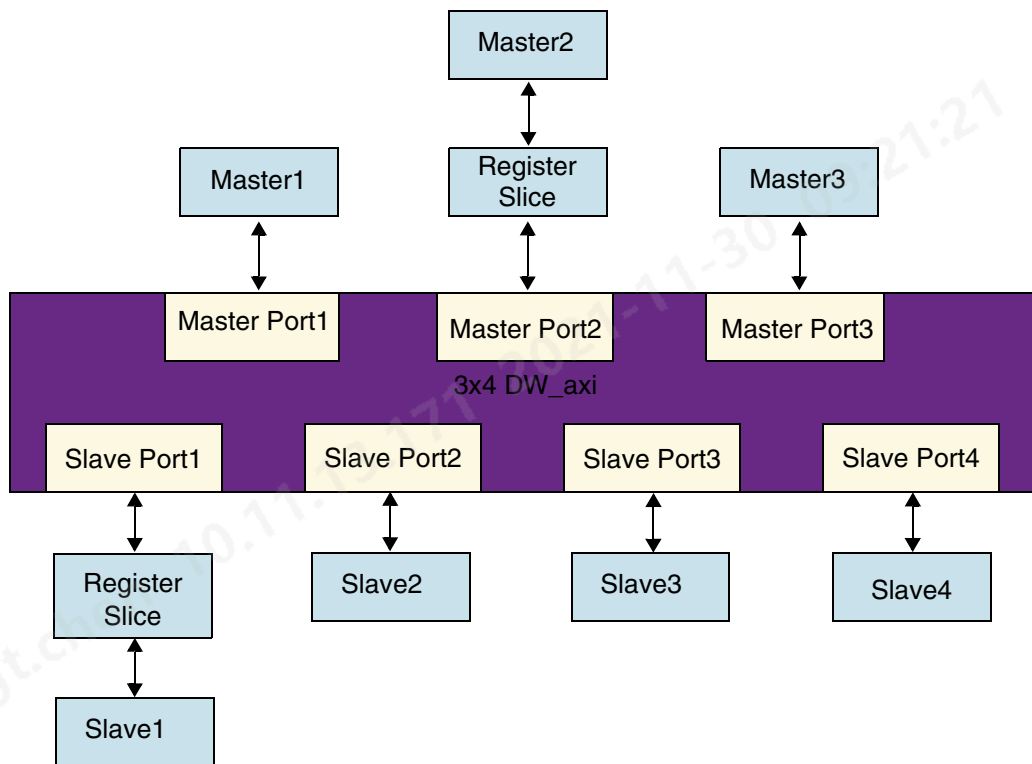
The configuration parameters `AXI_*_TMO` are used to select the timing mode option for each of the five AXI channels. For more information about these timing modes, refer to [“Timing Mode Options”](#) on page 72.

9.4.2 External Register Slice

If critical timing paths in a subsystem pass through inputs or outputs of DW_axi—even after applying timing mode options—it is possible to reduce those remaining critical paths using a register slice (`DW_axi_rs`). The `DW_axi_rs` is another component available with the DesignWare Synthesizable IP for AMBA 3 AXI. A register slice can be used either at a DW_axi master or slave port.

Figure 9-1 shows DW_axi where there is a register slice between Master 2 and Master Port 2 and Slave Port1 and Slave1.

Figure 9-1 Register Slice to Isolate Timing Paths



The DW_axi_rs register slice can configure each of the five AXI channels independently. The DW_axi_rs timing modes are:

- Forward Registered – Pipeline the forward control path and the corresponding AXI channel payload.
- Fully Registered – Pipeline the forward control path, the corresponding AXI channel payload, and the backward control path.
- Backward Registered – Pipeline the backward control path only.
- Pass Through Mode – All AXI channel signals are directly connected from input to output. DW_axi_rs does not register the forward control path, the backward control path, nor the AXI channel payload path.

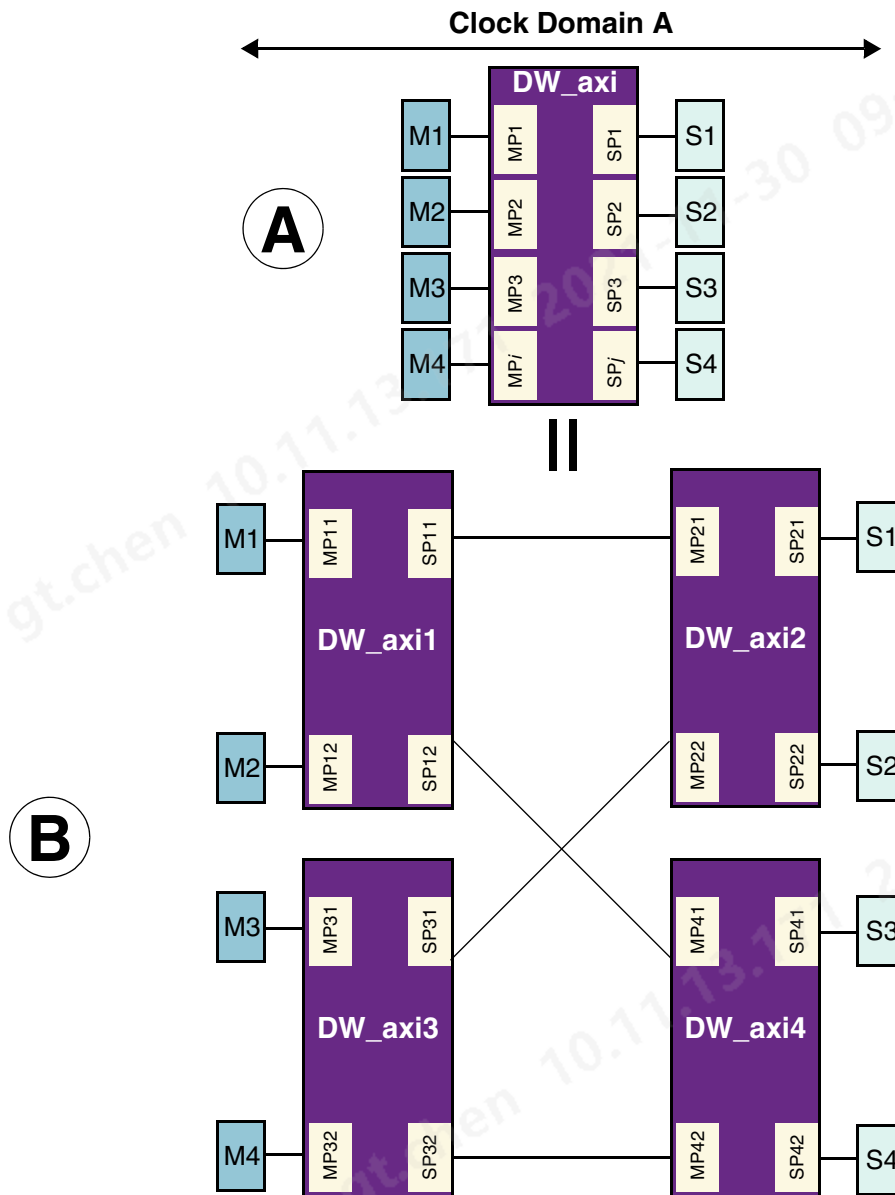
Register slices can be instantiated on the periphery of DW_axi to pipeline AXI channels regardless of the timing mode selected for that channel within DW_axi.

9.4.3 Tiling Interconnects

It is possible to reduce the critical path and routing density of a single DW_axi-based subsystem by splitting the AXI interconnect into multiple “tiled” interconnects. This design method allows the IP integrator to break up a single DW_axi instance into multiple instances of DW_axi. The multiple DW_axi instances are connected together to implement the same functionality as the single DW_axi architecture, but can achieve higher frequencies by breaking the different interconnect tasks into separate, less complex instances.

By a combination of configuration and additional intermediate master/slave connections, multiple tiled interconnects can function the same as a single interconnect. This is illustrated in [Figure 9-2](#) and described in the “[Example of Tiling Interconnects](#)” on page 263. Tiling interconnects can result in higher system clock frequency and improved routing by distributing the physical interconnect wiring.

Figure 9-2 Tiling DW_axi Interconnects



Single interconnect (“A”) is configured with:
 Four masters and four slaves
 All slaves visible to all masters
 Master priority set so that M1 is highest and M4 is lowest

Multiple interconnects (“B”) are each configured with:
 Two masters and two slaves
 Two slaves visible to two masters
 Master priority set so that MP_{n1} is higher than MP_{n2}

The “tiled” interconnects are connected to each other so that functionality is equal to that of the single interconnect.

9.4.3.1 Example of Tiling Interconnects

The four DW_axi interconnects illustrated in [Figure 9-2](#) are configured by setting the priority parameters in each of the DW_axi interconnects and connecting to the other DW_axi interconnects. In the example configuration/connectivity described below, the master priorities of the A and B systems shown in [Figure 9-2](#) are made to be functionally similar. Other configuration/connectivity options are possible and left to the IP integrator to evaluate. Although this 4x4 interconnect is unlikely to be a frequency limitation in a subsystem, it is a useful size to create a simple example of tiled interconnects.

1. Systems A and B are made similar by configuration of priority parameters in each instance of DW_axi:
 - ❑ DW_axi1 set MP11 higher priority than MP12
 - ❑ DW_axi2 set MP21 higher priority than MP22
 - ❑ DW_axi3 set MP31 higher priority than MP32
 - ❑ DW_axi4 set MP41 higher priority than MP42
2. Systems A and B are also made similar by connecting the intermediate master/slave connections between the DW_axi instances as shown:
 - a. SP11 of DW_axi1 to MP21 of DW_axi2 – This gives M1 and M2 access to S1 and S2 through MP21, the highest priority port in DW_axi2. DW_axi1 has M1 at a higher priority than M2.
 - b. SP12 of DW_axi1 to MP41 of DW_axi4 – This gives M1 and M2 access to S3 and S4 through MP41, the highest priority port in DW_axi4. At this point M1 and M2 have access to S1, S2, S3, S4, where M1 is a higher priority than M2.
 - c. SP31 of DW_axi3 to MP22 of DW_axi2 – This gives M3 and M4 access to S1 and S2 through MP22, a lower priority than MP21. M3 and M4 access S1 and S2 at a lower priority than M1 and M2.
 - d. SP32 of DW_axi3 to MP42 of DW_axi4 – This gives M3 and M4 access to S3 and S4. M3 and M4 access S3 and S4 at a lower priority than M1 and M2.

So if:

- M1 and M2 have access to S1 and S2 through DW_axi2 port MP21 at the highest priority in DW_axi2;
- M1 and M2 have access to S3 and S4 through DW_axi4 port MP41 at the highest priority in DW_axi4;
- M3 and M4 have access to S1 and S2 through DW_axi2 port MP22 at a lower priority than M1 and M2 coming through MP21;
- M3 and M4 have access to S3 and S4 through DW_axi4 port MP42 at a lower priority than M1 and M2 coming through MP41; and
- M3 has a higher priority than M4; and
- M1 higher than M2

Then:

- All slaves are visible to all masters and the priority is M1, M2, M3, M4. This shows that System B is equivalent to the slave visibility and priorities of System A. Transactions between masters and slaves in the two systems would complete similarly.

9.4.3.2 Design Considerations for Tiling Interconnects

Tiled interconnects can dramatically increase design frequency, compared to any other method supplied in this section. If a subsystem with high numbers of masters and slaves is required, and high frequency is also required, then interconnect tiling may be the only option for meeting frequency goals. However, before you implement tiled interconnects, please note the following design considerations:

1. **Bottlenecks** – Using a single DW_axi provides parallel access to every slave; Slave A and Slave B can both receive transactions from masters on the same cycle. However, when interconnect tiling is implemented to reduce slave connections on any given interconnect, bottlenecks are created.

- **Bottleneck Example**

- 3Mx16S (3 Master, 16 Slave subsystem)
- Frequency goal is not met using other critical path reduction methods and a single interconnect instance.

- **Tiled Architecture**

Replace the single DW_axi interconnect instance with multiple distributed DW_axi instances.

- IC-1 (interconnect #1) – 3Mx8S
Eight slaves of IC-1 tie to eight masters split between IC-2 and IC-3
- IC-2 – 4Mx8S
- IC-3 – 4Mx8S
16 slaves output from IC-2 and IC-3 attach to original 16 slaves.

- **Potential Bottleneck Created**

Eight outputs of IC-1 are used to connect to 16 slaves (through IC-2 and IC-3). Transactions that formerly could have been parallel in single interconnect implementation could become serial in tiled interconnect implementation, depending on transaction activity.

- **Benefits Achieved**

Replace the single DW_axi interconnect instance with multiple distributed DW_axi instances.

- **Higher frequency** – The frequency limit in DW_axi timing is directly related to the number of masters or slaves attached. The 3Mx16S original single interconnect has been changed to a three interconnects: 3Mx8S, 4Mx8S, and 4Mx8S. By reducing the number of slaves of any one DW_axi, and also using DW_axi timing mode options or DW_axi_rs (register slice) instances, the frequency of the interconnect in the subsystem is increased.
- **Reduced congestion** – If slaves in the subsystem are spread out, IC-2 and IC-3 can be located closer to the attached slaves than when using the single DW_axi architecture. This reduces the total interconnect wire length required, reducing overall congestion, compared to the greater routing resources required due to long wire routes to far-away slaves of the single DW_axi architecture.

- **Side Effects**

Increased area and increased latency, depending on what timing mode and/or register slices are used.

2. **Write interleaving** – AXI interfaces attached to DW_axi master ports are limited to a write interleaving depth of 1. This means that write data cannot be interleaved to DW_axi master ports. For

tilled interconnects, DW_axi slave ports connected to another DW_axi instance's master port must set the AXI_WID_S(j) configuration parameter to 1 (write interleave depth for a slave port).

Write Interleaving Example:

For a single DW_axi attached to a slave that supports a write interleaving depth of 4, this means that four masters attached to DW_axi could interleave write data to that slave simultaneously.

If that single DW_axi is removed and replaced with a tiled interconnect architecture – to increase frequency or to reduce routing congestion – this could potentially limit write interleaving. Taking the 4x4 example (see [Figure 9-2](#)), in the tiled architecture M1 and M2 are sharing a single master port (MP21) on DW_axi2. Since master ports on DW_axi are limited to a write interleaving depth of 1, then M1 and M2 could not interleave write data to either S1 or S2. The single DW_axi architecture, such as System A in [Figure 9-2](#), does allow write interleaving to S1/S2 from M1 and M2.

In certain situations, using a DW_axi_x2x (AXI to AXI Bridge) could be a useful solution to this write interleaving limitation, if it were a significant performance bottleneck for your design. For more information on DW_axi_x2x and interconnect tiling options, please contact Synopsys customer support.

3. Outstanding transactions – Careful consideration should be followed when choosing master port and slave port configuration parameters for DW_axi. This is just as important when DW_axi is part of a tiled interconnect.

DW_axi master ports are configured by the IP integrator to define the number of outstanding reads/writes per transaction ID and the number of outstanding transaction IDs that can be received by that master port. This limit is required for DW_axi to avoid deadlock.

DW_axi slave ports are configured by the IP integrator to define a maximum number of outstanding forwarded read and write commands from that slave port. The limit is required for DW_axi to monitor locked access transactions.

When using interconnect tiling, a DW_axi slave port output is connected to another DW_axi master port input. Care should be taken when configuring each DW_axi so that a slave port is not configured to allow many more transactions than the attached master port can accept. Similarly, a master port should not be configured to accept many more transactions than the attached slave port can allow. Either of these scenarios would result in logic being included in a DW_axi configured design that would never be needed.

4. Slave Visibility – The DW_axi slave visibility configuration parameters affect how you connect your tiled interconnects. Slave visibility can be used to significantly reduce the arbitration logic required inside DW_axi, decreasing area and potentially increasing frequency. When tiling interconnects, it may even allow DW_axi-to-DW_axi interfaces to be eliminated.

Slave Visibility Example:

For example, if you have a 4x4 DW_axi and tile it to make four 2x2 interconnects (see [Figure 9-2](#) on page 262), and if M1 and M2 are both configured to have S3 and S4 not visible (slave visibility turned off), then the connection between DW_axi1 and DW_axi4 is not required.

9.5 Quality of Service (QoS) Integration

The following section provides details regarding the integration of DW_axi QoS functionality into a design.

When configuring the slave port QoS Arbiter on the write address and read address channels, note the following:

- You can select slave port arbiters on write address channels and/or read address channels as QoS Arbiter. However, selecting the default slave arbiter alone as a QoS Arbiter is not recommended. You must select the default slave arbiter as a QoS Arbiter *along with* other slave arbiters for the read address channel and/or write address channel.
- The QoS Arbiter uses the QoS priority value for arbitration. You can provide the QoS priority value in either of the following ways:
 - Externally through pin-level master port signals – `awqos_m(x)/arqos_m(x)`.
 - Internally through register programming – applicable only for AXI4 configurations. In AXI4 configurations, QoS priority values are provided only through the pin-level master port signals `awqos_m(x)/arqos_m(x)`.
- If QoS functionality is enabled in AXI4 configurations, all master port read address and write address channels have `awqos_m(x)/arqos_m(x)`. However, in AXI3 configurations, you can configure external QoS signals for read address channels or write address channels separately on each master port.
- If QoS functionality is enabled in the DW_axi, then all slave ports drive the `arqos_s(j)` and `awqos_s(j)` output signals. In the case of multitle architecture or for slave(s) with QoS support, these signals must be connected to appropriate ports. Otherwise, you must discard these signals.

When configuring the QoS regulator logic on the write address and read address channels, note the following:

- You can select QoS regulator logic for the read address channel and/or write address channel for each master port separately.

9.5.1 Considerations

You should take the following into account when enabling QoS functionality:

- The QoS Arbiter is a dynamic-priority arbiter; that is, the priority of a transaction is controlled by the `awqos_m(x)/arqos_m(x)` signals at the master port.
- It is recommended that the burstiness value be programmed to less than or equal to the maximum outstanding transaction limit configured on the master port.
- A request from a master is regulated by QoS regulator logic only when both of the following are true:
 - Master-to-targeted-slave access for that channel is on a dedicated link
 - Slave arbiter is of type QoS Arbiter
- A request from a master is bypassed by QoS regulator logic when either of the following is true:
 - Master-to-targeted-slave access for that channel is on a shared link
 - Slave arbiter is any arbiter other than QoS Arbiter

9.5.2 Restrictions

The following restrictions apply to the QoS functionality within the DW_axi.

- In AXI3 configurations, locking functions are not supported if QoS is enabled.
- Shared-layer arbiters can be of type “QoS Arbiter”, but any request to access the slave on a shared layer channel is not regulated. This restriction is based on the assumption that slaves on a shared channel are of a low-bandwidth type.

DW_axi Application Notes

This appendix provides application notes for the DW_axi.

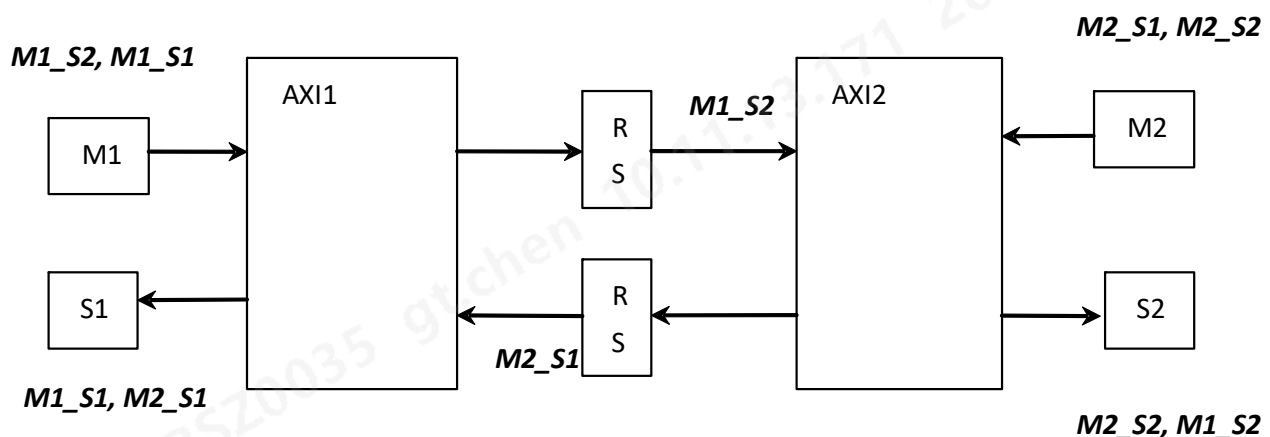
A.1 Multitile Deadlock Scenarios

This document describes the known deadlock conditions that exist in multitile configurations of DW_axi. All of the deadlock conditions relate to the protocol requirement that the first beat of write data for a transaction be issued to a slave in address order.

A.1.1 Bidirectional Multitile Systems

A bidirectional multitile system allows masters on each tile to access slaves on the other tile. In such systems, a deadlock occurs when a master on each tile sends a write transaction to a slave on the other tile, followed immediately by a write transaction to the slave on the same tile as the master. Deadlock can occur only when there is pipelining either internal to both tiles, or between the tiles. [Figure A-1](#) shows a bidirectional multitile system with pipelining using the register slice component (RS).

Figure A-1 Bidirectional Multitile Deadlock



In the scenario shown in [Figure A-1](#), each master first issues a transaction to the slave on the other tile. This transaction is buffered in the register slice component, which then allows each master to issue a second transaction, this time to the slave local to the master.

Due to the difference in pipelining between masters and slaves on the same tile compared to masters and slaves on different tiles, each slave receives the transaction from the local master (on the same tile) first, in spite of the fact that the non local master issued a transaction to that slave first.

The problem arises when write data is sent from the masters. For example, master M1 sends write data to slave S2 first. But slave S2 is waiting for write data from master M2 first (the first beat in address order protocol rule), and does not accept write data from M1 until it gets at least one beat of write data from M2. At the same time, master M2 sends write data to slave S1, but again, this write data is not accepted by this slave, because slave S1 is waiting for data from master M1. So in effect, each master has locked the other master out.

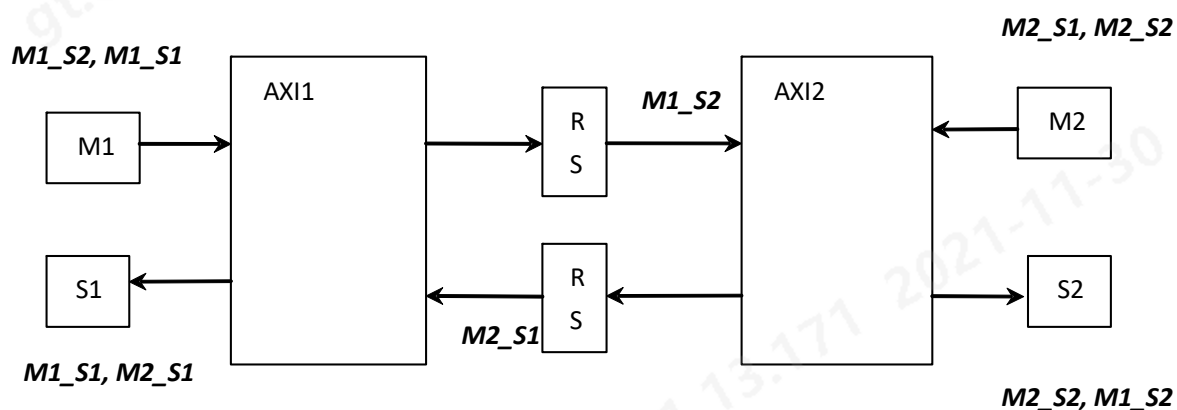
For such a deadlock scenario to occur, pipelining must be present, as it allows an address from a master to be accepted without reaching the actual slave, which then allows the master to issue another address to a different slave, resulting in the deadlock.

To find out how this deadlock scenario can be prevented, refer to “[Multitile Deadlock Prevention](#)” on page 67.

A.1.2 Multitile Systems

This deadlock scenario is functionally very similar to the one described in section 1. The main difference is that this scenario occurs in a non bidirectional, multitile system. [Figure A-2](#) illustrates this scenario.

Figure A-2 Multitile Deadlock



In the scenario shown in [Figure A-2](#), a deadlock occurs when masters on two tiles access the same two slaves on other tiles in the following manner:

1. Master M1 issues write transactions to slave S2, then to slave S1.
2. At the same time, master M2 issues write transactions to slave S1 and then to slave S2.
3. M1_S2 is delayed in the path that it takes, such that M2_S2 reaches slave S2 first.
4. Similarly, M2_S1 is delayed reaching slave S1, such that M1_S1 reaches slave S1 first.

At this point, master M1 sends write data to slave S2 first, but S2 does not accept it because it is waiting for a beat from master M2 (since M2_S2 reached slave S2 first). However, M2 is prevented from sending this beat because it must first complete the transaction M2_S1; and write data for this transaction is stalled, because

slave S1 is waiting for a beat from master M1. Now the pipelining on the write data channel is full, and no write data beats can be accepted, so the system is deadlocked.

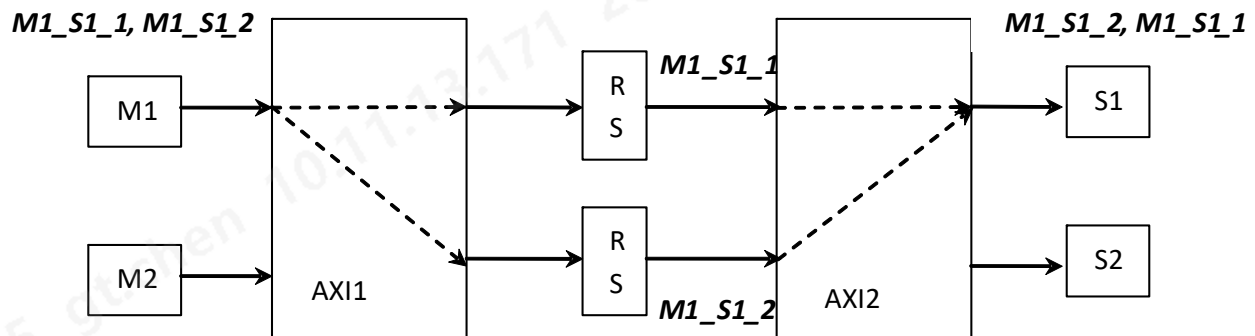
This deadlock scenario can also occur with just three DW_axi tiles, if, for example, AXI4 is removed and slave S2 is moved to AXI1.

To find out how this deadlock scenario can be prevented, refer to [“Multitile Deadlock Prevention”](#) on page 67.

A.1.3 Multiple Paths from Master to Slave

Deadlock can also occur on the write data channel if a master sends two transactions to the same slave through two different paths, as shown in [Figure A-3](#).

Figure A-3 Multi-path Deadlock Scenario



In the scenario shown in [Figure A-3](#), master M1 sends M1_S1_1 through path 1, and M1_S1_2 through path 2. If M1_S1_1 is delayed (possibly by other transactions in this path), it is possible that M1_S1_2 reaches slave S1 before M1_S1_1.

Then when M1 sends write data for M1_S1_1, it is not accepted by slave S1 because slave S1 is waiting for at least one beat for M1_S1_2. However, master M1 cannot issue a beat for M1_S1_2 until all of M1_S1_1 has left AXI1, which it is not guaranteed to do since slave S1 is not accepting those beats. At this point the system is deadlocked.

To find out how this deadlock scenario can be prevented, refer to [“Multitile Deadlock Prevention”](#) on page 67.

B

Basic Core Module (BCM) Library

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

This chapter provides more information about the BCMs used in DW_axi.

- [“BCM Library Components”](#) on page 273
- [“Synchronizer Methods”](#) on page 273

B.1 BCM Library Components

[Table B-1](#) describes the list of BCM library components used in DW_axi.

Table B-1 List of BCM Library Components used in the Design


BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_bcm01.v	Minimum/Maximum Value	DW_minmax
DW_axi_bcm06.v	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctrl_s1_df
DW_axi_bcm21.v	Single Clock Data Bus Synchronizer	DW_sync
DW_axi_bcm57.v	Sync. Write-Port, Asynchronous. Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff

B.2 Synchronizer Methods

This section describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_axi to cross clock domain crossing signals.

This section contains the following:

- [“Synchronizers Used in DW_axi”](#) on page 274
- [“Synchronizer 1: Simple Double Register Synchronizer \(DW_axi\)”](#) on page 274

 **Note**

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

B.2.1 Synchronizers Used in DW_axi

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

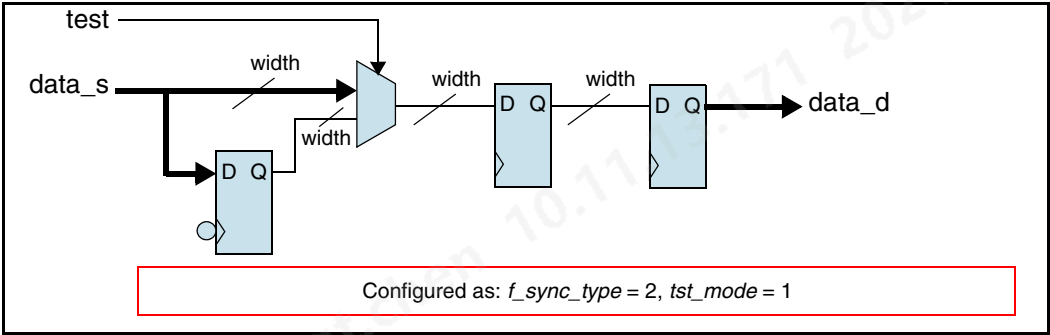
Table B-2 Synchronizer Used in DW_axi

Synchronizer Module File	Synchronizer Type and Number
DW_axi_bcm21.v	Synchronizer 1: Simple Multiple register synchronizer

B.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_axi)

This is a single clock data bus synchronizer for synchronizing control signals that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. As aclk and pclk are asynchronous, when QOS is enabled i.e. AXI_HAS_QOS = 1, then DW_axi_bcm21 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameter AXI_NUM_SYNC_FF. The following example shows the two stage synchronization process (Figure B-1) both using positive edge of clock.

Figure B-1 Block Diagram of Synchronizer 1 with two Stage Synchronization (Both Positive Edge)



C

Glossary

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

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

peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
sparse data	Data bus data which is individually byte-enabled. The AXI bus allows sparse data transfers using the WSTRB signal, each byte lane individually enabled. The AHB bus does not allow sparse data transfers.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index

A

Accesses

atomic [52](#)

active command queue

definition [275](#)

Addresses

regions, multiple [42](#)

application design

definition [275](#)

Arbiter, pipeline channel [77](#), [78](#)

Arbitration

in DW_axi [46](#)

Architectures

multiple address [31](#)

multiple data bus [31](#)

Atomic accesses [52](#)

Avoidance, deadlock, out-of-order [54](#)

AXI system, single master [60](#)

AXI*_ARB parameters [77](#), [78](#)

B

BFM

definition [275](#)

big-endian

definition [275](#)

blocked command stream

definition [275](#)

blocking command

definition [275](#)

Boot mode

and slave visibility [43](#)

description of [42](#)

Buses

ID [53](#)

C

Cascading interconnects [261](#)

Channels

read address [61](#)

read data [63](#)

write address [61](#)

write data [62](#)

write data, stalling [58](#)

write response [63](#)

command channel

definition [275](#)

command stream

definition [275](#)

component

definition [275](#)

configuration

definition [275](#)

configuration intent

definition [275](#)

Customer Support [14](#)

cycle command

definition [275](#)

D

Deadlock avoidance, out-of-order [54](#)

decoder

definition [276](#)

Decoders

external [44](#)

system [41](#)

Default slave [40](#)

design context

definition [276](#)

design creation

definition [276](#)

DesignWare Library

definition [276](#)

DesignWare Synthesizable Components

remap feature [42](#)

dual role device

definition [276](#)

DW_axi

- arbitration
 - scenarios of [46](#)
- features of [21](#)
- memory map [43](#)
- remap operation [42](#)
- slaves
 - visibility [43](#)

E

- endian
 - definition [276](#)
- Exclusive accesses
 - Accesses
 - exclusive [53](#)
- External decoder [44](#)
- External register slice [80](#), [260](#)

F

- Features
 - default slave [40](#)
- Features, of DW_axi [21](#)
- Full-Functional Mode
 - definition [276](#)

G

- GPIO
 - definition [276](#)
- GTECH
 - definition [276](#)

H

- hard IP
 - definition [276](#)
- HDL
 - definition [276](#)

I

- ID bus [53](#)
- IIP
 - definition [276](#)
- implementation view
 - definition [276](#)
- instantiate
 - definition [276](#)
- Interconnects, cascading [261](#)
- interface
 - definition [276](#)
- Interfaces

- low-power [82](#)

- Interleaving
 - write data [56](#), [60](#)
- IP
 - definition [276](#)

L

- little-endian
 - definition [276](#)
- Locked transfers [52](#)
- Low-power interface [82](#)

M

- master
 - definition [276](#)
- Master port
 - description [29](#)
- Masters
 - single, AXI system [60](#)
- Memory map
 - of DW_axi [43](#)
- model
 - definition [276](#)
- Modes
 - timing, combinatorial [72](#)
 - timing, forward_reg [73](#)
 - timing, full_reg [75](#)
 - timing, options [72](#)
- monitor
 - definition [276](#)
- Multiple address architecture [31](#)
- Multiple address regions [42](#)
- Multiple data bus architecture [31](#)

N

- non-blocking command
 - definition [276](#)
- Normal mode
 - and slave visibility [43](#)
 - description of [42](#)

O

- Operation modes
 - and slave visibility [43](#)
 - description of [42](#)
- Operations
 - remap [42](#)
- Out-of-order deadlock avoidance [54](#)

P

Parameters

AXI*_ARB 77, 78

Paths

timing, register slice 261

Payload source

stalling 61

peripheral

definition 277

Pipeline channel arbiter 77, 78

Ports

Master, description 29

Slave, description 30

R

Read address channel 61

stalling, payload source 61

Read data

re-ordering 56

Read data channel 63

stalling, payload source 63

Regions

multiple address 42

Register slice, external 260

Registers

slice, external 260

Remap operation 42

Re-ordering

read data 56

write response 56

RTL

definition 277

S

SDRAM

definition 277

SDRAM controller

definition 277

Simulation 245

Slave

default 40

visibility 43

slave

definition 277

Slave port

description 30

SoC

definition 277

SoC Platform

AHB contained in 17

APB, contained in 17

defined 17

soft IP

definition 277

sparse data

definition 277

Stalling

payload source 61

read address channel 61

read data channel 63

write address channel 61

write data channel 62

write response channel 63

static controller

definition 277

synthesis intent

definition 277

synthesizable IP

definition 277

System decoders 41

T

technology-independent

definition 277

Testsuite Regression Environment (TRE)

definition 277

Timing

External register slice 80

Pipeline channel arbiter 77, 78

Timing mode options 72

combinatorial 72

forward_reg 73

full_Reg 75

Timing paths

register slice 261

Transfers

locked 52

TRE

definition 277

Trustzone support 44

V

Verification 245

VIP

definition 277

VTE

block diagram [246](#)

W**wrap**

definition [277](#)

wrapper

definition [277](#)

Write address channel [61](#)

stalling, payload source [61](#)

Write data

interleaving [56](#), [60](#)

Write data channel [62](#)

stalling [58](#)

stalling, payload source [62](#)

Write response channel [63](#)

stalling, payload source [63](#)

Write responses

re-ordering [56](#)

Z**zero-cycle command**

definition [277](#)