



DesignWare Cores

PCI Express

Reference Manual

Dual Mode (DM) Core
Root Complex (RC) Core
End Point (EP) Core
Switch (SW) Core
AHB Bridge Module
AXI Bridge Module

Copyright Notice and Proprietary Information

Copyright © 2008 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

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.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, and Vera are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BOA, BRT, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, DC Expert, DC Professional, DC Ultra, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, Direct RTL, Direct Silicon Access, Discovery, Dynamic-Macromodeling, Dynamic Model Switcher, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Formal Model Checker, FoundryModel, Frame Compiler, Galaxy, Gatran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSIM^{plus}, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Milkyway, ModelSource, Module Compiler, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Raphael, Raphael-NES, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, Softwire, Source-Level Design, Star-RCXT, Star-SimXT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSI, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

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

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

www.synopsys.com

Contents

Chapter 1

Architecture	19
1.1 Common Xpress Port Logic (CXPL)	24
1.2 Transmit Application-Dependent Module (XADM)	26
1.2.1 Arbitration: DM	26
1.2.2 Arbitration: RC	27
1.2.3 Arbitration: EP	28
1.2.4 Arbitration: SW	28
1.2.5 Credit Checking	29
1.3 Receive Application-Dependent Module (RADM)	30
1.3.1 Receive Application-Dependent Module (RADM): DM / RC / EP	30
1.3.2 Receive Application-Dependent Module (RADM): SW	31
1.3.3 Received Completion TLP Processing: DM / RC / EP	33
1.3.4 Message Processing	34
1.3.5 Posted and Non-Posted Request and Completion TLP Processing	34
1.3.6 Lookup Table: DM / RC / EP	35
1.3.7 RADM Demux: DM / RC / EP	35
1.3.8 RADM Demux: SW	35
1.4 Configuration-Dependent Module (CDM)	36
1.4.1 Configuration-Dependent Module (CDM): DM	36
1.4.2 Configuration-Dependent Module (CDM): RC	38
1.4.3 Configuration-Dependent Module (CDM): EP	39
1.4.4 Configuration-Dependent Module (CDM): SW	40
1.5 Power Management Controller (PMC)	41
1.5.1 Power Management Controller (PMC): DM / RC / EP	41
1.5.2 Power Management Controller (PMC): SW	42
1.6 Local Bus Controller (LBC) (DM / EP / SW)	43
1.6.1 LBC Switching	44
1.7 Message Generation (MSG_GEN)	45
1.7.1 Message Generation (MSG_GEN): DM	45
1.7.2 Message Generation (MSG_GEN): RC	46
1.7.3 Message Generation (MSG_GEN): EP	46
1.7.4 Message Generation (MSG_GEN): SW	47
1.8 Hot Plug Control (hotplug_ctrl) Module: DM / RC / SW	48

Chapter 2

Core Operations	49
2.1 Initialization	49
2.1.1 Initialization: DM / RC / EP	49
2.1.2 Initialization: SW	50

2.2 Link Establishment	51
2.3 Basic TLP Processing	52
2.3.1 Basic TLP Processing: Transmit (DM / RC / EP)	52
2.3.2 Basic TLP Processing: Transmit (SW)	53
2.3.3 Basic TLP Processing: Receive (DM / RC / EP)	54
2.3.4 Basic TLP Processing: Receive (SW)	55
2.3.5 Transmit TLP Arbitration	56
2.3.6 Transmit Retry	58
2.3.7 Transmit DLLP Priorities	59
2.3.8 Receive Filtering	60
2.3.9 Receive Queuing (DM / RC / EP)	74
2.3.10 Receive Queuing (SW)	77
2.3.11 Error Handling for Received TLPs	79
2.4 Interrupts	83
2.4.1 Interrupts (EP Mode)	83
2.4.2 Interrupts (RC Mode)	84
2.4.3 Interrupts (SW)	85
2.5 Flow Control	87
2.6 Address Translation	88
2.6.1 Address Translation for the Transmit Path	88
2.6.2 Address Translation for the Receive Path	89
2.6.3 PCI-SIG Address Translation Services (ATS)	90
2.7 Peer-to-Peer Support (P2P)	91
2.8 PCI Express 2.0 Features (Gen2)	92
2.9 Power Management	93
2.9.1 L0s Power Down	95
2.9.2 L1 Power Down	96
2.9.3 L2/L3 Power Down	98
2.10 Single Root I/O Virtualization (SR-IOV) (EP / DM in EP mode)	99
2.10.1 Overview	99
2.10.2 Features and Limitations	100
2.10.3 Function Level Reset (FLR) (EP / DM in EP mode)	102
2.10.4 Alternate Routing ID Interpretation (ARI) (EP / DM in EP mode)	103
2.11 Completion Timeout Ranges	104

Chapter 3

Signal Descriptions	105
3.1 Top-Level I/O Diagrams	105
3.2 Transmit Client Interfaces	114
3.2.1 Transmit Client Interface Protocol Rules	114
3.2.2 Transmit Client0 Interface (XALI0) Signals	115
3.2.3 Transmit Client1 Interface (XALI1) Signals	125
3.2.4 Transmit Client2 Interface (XALI2) Signals (optional)	125
3.2.5 Example Transmit Client Transactions	126
3.2.6 Transmit Address Alignment	129
3.3 Receive Completion Interface (RCPL) (DM / RC / EP)	133
3.3.1 RCPL Protocol Rules	133
3.3.2 RCPL Interface Signals	134
3.3.3 Example RCPL Transactions	141
3.4 Bypass Interface	145

3.4.1 Bypass Interface Protocol Rules	145
3.4.2 Bypass Interface Signals	146
3.5 Receive Target 1 Interface (RTRGT1) (optional)	155
3.5.1 RTRGT1 Protocol Rules	155
3.5.2 RTRGT1 Interface Signals	156
3.5.3 Target Completion LUT (optional) (DM / RC / EP)	166
3.5.4 Example RTRGT1 Transaction	169
3.6 External Local Bus Interface (ELBI) (DM / EP / SW)	172
3.6.1 ELBI Protocol Rules	172
3.6.2 ELBI Signals	173
3.6.3 Example ELBI Transactions	176
3.7 Data Bus Interface (DBI)	177
3.7.1 DBI Protocol Rules	177
3.7.2 DBI Signals	178
3.7.3 Example DBI Transactions	182
3.8 Message Signaled Interrupt (MSI) Interface (DM / EP / SW)	183
3.8.1 MSI Interface Signals	183
3.8.2 Example MSI Transaction	187
3.9 MSI-X Interface (optional) (DM / EP / SW)	188
3.9.1 MSI-X Interface Signals (optional)	189
3.9.2 Example MSI-X Transaction	192
3.10 Vital Product Data (VPD) Support (optional)	193
3.10.1 Vital Product Data (VPD) Read	193
3.10.2 Vital Product Data (VPD) Write	193
3.11 Power Budgeting (optional)	194
3.12 Vendor Message Interface (VMI)	195
3.12.1 VMI Signals	195
3.12.2 Example VMI Transaction	199
3.13 System Information Interface (SII)	200
3.13.1 SII Signals: Overall Core Control	200
3.13.2 SII Signals: Interrupt	203
3.13.3 SII Signals: Power Management	213
3.13.4 SII Signals: Electromechanical	217
3.13.5 SII Signals: Messages	221
3.13.6 SII Signals: Configuration Information	225
3.13.7 SII Signals: Transmit Control	236
3.13.8 SII Signals: Receive Control/Completion Timeout	242
3.13.9 SII Signals: Reset Generation	247
3.13.10 SII Signals: Debug	248
3.13.11 SII Signals: Diagnostic and Debug Control (optional)	253
3.14 PHY Interface (PIPE)	258
3.15 Synopsys PHY Interface	268
3.16 Clock and Reset	273
3.17 External RAM Interface (optional)	277
3.17.1 Retry Buffer and SOT Buffer Interface	277
3.17.2 Receive Queue Interface	282
3.18 Address Translation (optional)	305
3.19 Peer-to-Peer (optional)	310
3.20 Function Level Reset (FLR) (DM / EP)	313
3.20.1 FLR Timing	315

3.20.2 FLR Operation	316
Chapter 4	
Registers	317
4.1 PCIe Registers (DM in EP Mode / EP)	317
4.1.1 Register Space Layout	318
4.1.2 PF Register Maps	320
4.1.3 VF Register Maps	328
4.1.4 Accessing Configuration Registers	332
4.1.5 Register Default Values	334
4.1.6 PF PCI-Compatible Configuration Header Register Details	335
4.1.7 PF PCI Standard Capability Structures Register Details	364
4.1.8 PF PCI Express Extended Capability Register Details	388
4.1.9 VF PCI-Compatible Configuration Header Register Details	415
4.1.10 VF PCI Standard Capability Structures Register Details	422
4.2 PCIe Registers (DM in RC Mode / RC / SW)	435
4.2.1 Register Space Layout	436
4.2.2 Register Maps	438
4.2.3 Accessing Configuration Registers	445
4.2.4 Register Default Values	447
4.2.5 PCI-Compatible Configuration Header Register Details	448
4.2.6 PCI Standard Capability Structures Register Details	464
4.2.7 PCI Express Extended Capabilities Register Details (DM / RC / SW)	488
4.3 PCIe Registers: Port Logic	504
4.3.1 Port Logic Register Map	505
4.3.2 Port Logic Registers: General	508
4.3.3 Port Logic Registers: Flow Control Credit Status	522
4.3.4 Port Logic Registers: Transmit Arbitration	524
4.3.5 Port Logic Registers: Receive Queue Control	525
4.3.6 Port Logic Registers: Segmented Buffer Depth	550
4.3.7 Port Logic Registers: Gen2	574
4.3.8 Port Logic Registers: PHY Status and Control Registers	575
Chapter 5	
Parameters	577
5.1 Parameters Available in the coreConsultant GUI	577
5.1.1 General Configuration Parameters	577
5.1.2 Transmit Configuration Parameters	587
5.1.3 Queuing and Buffer Configuration Parameters	591
5.1.4 VC Configuration Parameters	596
5.1.5 Filter Configuration Parameters	604
5.1.6 Common Register Configuration Parameters	606
5.1.7 Function Configuration Parameters	620
5.1.8 RAM Configuration Parameters	636
5.1.9 PHY Configuration Parameters	639
5.1.10 Design Pipelining Configuration Parameters	643
5.2 TP Parameter	650
Appendix A	
Core Frequencies and Sizes	651

Appendix B	
AHB Bridge Module	653
B.1 Product Overview	653
B.1.1 General Product Description	653
B.1.2 System Overview	656
B.1.3 Features List	658
B.1.4 Feature Limitations	660
B.1.5 Clocks and Resets	661
B.1.6 Supported Core Configurations	662
B.2 PCIe AHB Core Operations	663
B.2.1 Supported AHB Transfer Type	663
B.2.2 Supported AHB Burst Operations	663
B.2.3 Supported AHB Transfer Size	664
B.2.4 Early Burst Termination (EBT) Support	665
B.2.5 SPLIT Response Mode	666
B.2.6 Endianess Support	668
B.2.7 Accessing Native PCIe Core Registers and Application-Specific Registers	669
B.2.8 Shared Slave Interface for DBI Access	672
B.2.9 Address Translation When Using the AHB Bridge	673
B.2.10 Zero-Byte Transfers Over the AHB Bridge (Flush Semantics)	674
B.2.11 I/O and CFG Transaction Handling	675
B.2.12 Message Transaction Handling	676
B.2.13 Transaction Order Enforcement Through the AHB Bridge	677
B.2.14 Memory Read/Write With Data Gaps ("Holes")	679
B.2.15 MTU Read Request Size and Page Boundary Support	680
B.2.16 Legacy Interrupt Handling and Other Message Handling Over the AHB Bridge	681
B.2.17 PCIe Vendor-Defined Messages	682
B.2.18 MSI Handling Over the AHB Bridge	683
B.2.19 Parity Checking Over AHB Bridge	683
B.3 Top-Level I/O Diagrams	684
B.4 Signal Descriptions	686
B.4.1 AHB Clock and Reset Signals	687
B.4.2 AHB Master Interface Signals	689
B.4.3 AHB Slave Interface Signals	693
B.4.4 AHB Optional DBI Access Slave Interface Signals	698
B.4.5 AHB External RAM Interface Signals	700
B.4.6 AHB Bridge SII Interface Addition	710
B.5 Parameters	711
B.6 Implementation Guidelines	715
B.6.1 Scope of the Implementation Guidelines	715
B.6.2 Detailed Descriptions of the Structural Chip-Level Reference Design	717
B.6.3 Considerations During Integration of the DWC PCIe AHB IP	737
B.6.4 Configuration Parameters	738
B.6.5 RAMs	741
B.6.6 The Suggested Integration Method for Application Logic	743
B.6.7 VTB Test Bench Integration Method (for EP Core Only)	744
B.6.8 How to Proceed	752
Appendix C	
AXI Bridge Module	753

C.1 Product Overview	753
C.1.1 General Product Description	753
C.1.2 System Overview	756
C.1.3 Features List	758
C.1.4 Feature Limitations	759
C.1.5 Clocks and Resets	760
C.1.6 Supported Core Configurations	761
C.2 PCIe AXI Core Operations	762
C.2.1 Supported AXI Transfer Type	762
C.2.2 Supported AXI Burst Operations	762
C.2.3 Supported AXI Transfer Size	763
C.2.4 Early Termination Support	765
C.2.5 SPLIT Response Mode	765
C.2.6 Endianess Support	765
C.2.7 Accessing Native PCIe Core Registers	766
C.2.8 Shared Slave Interface for DBI Access	770
C.2.9 Address Translation When Using the AXI Bridge	771
C.2.10 Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)	772
C.2.11 I/O and CFG Transaction Handling	773
C.2.12 Message Transaction Handling	774
C.2.13 Transaction Order Enforcement Through the AXI Bridge	775
C.2.14 Memory Read/Write With Data Gaps ("Holes")	777
C.2.15 MTU Read Request Size and Page Boundary Support	778
C.2.16 Legacy Interrupt Handling and Other Message Handling Over the AXI Bridge	779
C.2.17 PCIe Vendor-Defined Messages	780
C.2.18 MSI Handling Over the AXI Bridge	781
C.2.19 Parity Checking Over AXI Bridge	782
C.3 Top-Level I/O Diagrams	784
C.4 Signal Descriptions	787
C.4.1 AXI Clock and Reset Signals	788
C.4.2 AXI Master Interface Signals	790
C.4.3 AXI Slave Interface Signals	801
C.4.4 AXI Optional DBI Access Slave Interface Signals	811
C.4.5 AXI External RAM Interface Signals	819
C.4.6 AXI Bridge SII Interface Addition	829
C.5 Parameters	830
C.6 Implementation Guidelines	833
C.6.1 Scope of the Implementation Guidelines	833
C.6.2 Detailed Descriptions of the Structural Chip-Level Reference Design	835
C.6.3 Considerations During Integration of the DWC PCIe AXI IP	857
C.6.4 Configuration Parameters	858
C.6.5 RAMs	861
C.6.6 The Suggested Integration Method for Application Logic	863
C.6.7 VTB Test Bench Integration Method (EP Core only)	863
C.6.8 How to Proceed	864
Appendix D	
App Note: FGPA Synthesis for PCIe Cores	865
D.1 Introduction	865
D.2 Requirements for FPGA Implementation	867

D.3 Xilinx FPGA Flow	868
D.4 Altera FPGA Flow	868
D.5 Troubleshooting	869
D.6 Example DWC_pcie_ep FPGA Synthesis Summary Regression Results	870
Appendix E	
App Note: Interrupts (Legacy, MSI, MSI-X)	873
E.1 Introduction	873
E.1.1 PCI Legacy Interrupts	873
E.1.2 Message Signaled Interrupts (MSI)	873
E.1.3 MSI-X	874
E.2 Application Logic for PCI Express Interrupts	875
E.2.1 PCI Legacy Interrupt Application Logic	875
E.2.2 MSI Application Logic	876
E.2.3 MSI-X Application Logic	878
E.3 Interrupt Selection, Configuration, and Initialization	880
E.3.1 DesignWare IP Selection	880
E.3.2 DesignWare IP Configuration	880
E.3.3 DesignWare IP Hardware Initialization	880
E.3.4 DesignWare IP Software Initialization	880
E.4 Additional MSI-X Implementation Notes	881
E.4.1 Large MSI-X Tables	881
E.4.2 Application Control of Pending Bits	881
E.5 Miscellaneous Interrupt FAQ	882
E.5.1 MSI, MSI-X, and Legacy Interrupts	882
E.5.2 MSI-X Tables and Pending Bit Array	882
E.5.3 MSI-X Ordering	882
E.5.4 MSI-X and Switch/Root Ports	883
Appendix F	
App Note: Order Enforcement Using the PCIe Core and AXI/AHB Bridge Modules	885
F.1 PCIe Ordering Rule Overview	885
F.2 DWC PCIe Core Inbound Order Enforcement	886
F.3 DWC PCIe Core Outbound Order Enforcement	888
F.4 DWC PCIe AHB/AXI Bridge Inbound Order Enforcement	890
F.5 DWC PCIe AHB/AXI Bridge Outbound Order Enforcement	892
Appendix G	
App Note: Application Interfaces for PCI and PCIe	893
G.1 PCI to PCIe Comparison	895
G.1.1 PCI and PCIe Physical and Link Layers are Not Compatible	895
G.1.2 PCI and PCIe Bus Transactions are Similar	895
G.1.3 PCIe and PCI Base Configuration Registers are the Same	896
G.1.4 Other Differences Between PCI and PCIe	896
G.2 A Comparison of PCIe and PC IP Application Interfaces	898
G.2.1 Transmit Interfaces	898
G.2.2 Target Receive Interfaces	900
G.2.3 Read Response Interfaces	902
G.2.4 Read-Only Configuration Initialization	903
G.3 Adaptor and Redesign Checklist	904

G.4 Summary	904
Appendix H	
App Note: PCI Express Port Bifurcation	905
H.1 PCI Express Port	906
H.2 Bifurcation Methods	908
H.2.1 Bifurcation Approaches	908
H.2.2 PHY (Mixed-Signal) Bifurcation	908
H.2.3 Digital Logic Approach 1: Multi-threading and Duplication	909
H.2.4 Digital Logic Approach 2: Multiple Instantiation	909
H.2.5 Summary Comparison of Bifurcation Approaches	910
H.3 Bifurcation Example	911
H.3.1 Overview	911
H.3.2 Digital IP to PHY Connections	911
H.3.3 Lane Reversal Considerations	913
H.3.4 Static or Dynamic Bifurcation	913
Appendix I	
App Note: Selecting PCI Express IP for your Design	915
I.1 Introduction	915
I.2 PCI Express Overview	916
I.2.1 PCI Express System Example: PC Motherboard Based System	916
I.2.2 PCI Express System Example: Chip-to-Chip System	917
I.2.3 PCI Express Protocol Stack	917
I.2.4 PCI Express Links	918
I.2.5 Digital IP to PHY Interface (PIPE)	919
I.3 Selecting PCI Express IP for Your Application	920
I.3.1 Selecting the Lane Width	920
I.3.2 Selecting Mixed-Signal PCI Express IP (PHY)	920
I.3.3 Selecting Digital PCI Express IP Types	921
I.4 Implementing PCI Express in Your Design	925
I.5 Summary	926
Appendix J	
App Note: Introduction to PCIe Switches	927
J.1 Introduction	927
J.2 Switch Architecture	929
J.3 Digital IP	930
J.4 Mixed Signal IP	930
J.5 Switch Application Logic Summary	931
J.5.1 Routing, Arbitration, and Response Logic	931
J.5.2 Packet Buffering	935
J.5.3 Error Messages and Error Detection	938
J.5.4 Power Management	938
J.5.5 Legacy Interrupt Messages	939
J.5.6 Reset and Link Down	940
J.5.7 Packet Arbitration	940
J.6 Advanced Switch Feature Summary	941
J.6.1 Digital IP Initialization	941
J.6.2 Poisoned TLPs	941

J.6.3 Virtual Channels and Traffic Classes	941
J.6.4 Hot Plug	941
J.6.5 Locked Transactions	942
J.6.6 Ordering Rules	942
J.6.7 Deadlock Prevention	942
J.6.8 Bifurcation	942
J.7 Non-Standard Switch Feature Summary	943
J.7.1 Non-transparent Switches	943
J.7.2 Reconfigurable Upstream / Downstream Ports	943
J.7.3 Application in Switches	943
J.8 Conclusion	943

Appendix K

App Note: Calculating PCI Express Throughput	945
K.1 PCI Express Throughput	945
K.2 Configuring the PCI Express Core With Respect to Throughput	946
K.3 Effective Throughput	947
K.3.1 Effective Throughput Calculation	947
K.3.2 Other Factors Impacting Throughput	949
K.4 Configuring PCI Express with Other Applications	950
K.4.1 Configuring the PCI Express Core for Gigabit Ethernet Applications	950
K.4.2 Configure the PCI Express core for SATA II Applications	950
K.5 Summary	951

Appendix L

App Note: How to Tie Off Unused Lanes	953
---	-----

Revision History

Version	Date	Description
3.22a	November 2008	There are no documentation revisions associated with the 3.22a release.
3.20a	October 2008	<p>Added the following features to the DM core in EP mode:</p> <ul style="list-style-type: none"> • Alternate Routing-ID (ARI) • Address Translation Services (ATS) • Completion Timeout Ranges • Per-Vector Masking (PVM) in MSI (optional) • Single Root I/O Virtualization (SR-IOV) • Function Level Reset (FLR)
3.10a	July 2008	<p>Added the following features to the EP core:</p> <ul style="list-style-type: none"> • Alternate Routing-ID (ARI) • Address Translation Services (ATS) • Completion Timeout Ranges • Per-Vector Masking (PVM) in MSI (optional) • Single Root I/O Virtualization (SR-IOV) • Function Level Reset (FLR)
3.00a	December 2007	<ul style="list-style-type: none"> • Consolidated databooks of DM, RC, EP, and SW cores into a unified databook. • Split the databook into a User Manual and Reference Manual. • Added Peer-to-Peer support in DM and RC cores. • Increased support for Synopsys PHY • Added VTB support for EP core. • Incorporated chip_top simplifications into all cores.
2.80a	March 2007	<ul style="list-style-type: none"> • Implemented new features/changes per <i>PCI Express 2.0 Base Specification</i>. • Added LBC Feature Enhancement (Section 3.5.1) • Added Address Translation Support (Section 4.7). • Updated Signal, Register, and Parameter Descriptions in Chapters 5 through 9. • Fixed documentation errata.

Version	Date	Description
2.75b	November 2006	There are no documentation revisions associated with the 2.75b database release.
2.75a	October 2006	Fixed documentation errata.
2.71a	October 2006	There are no documentation revisions associated with the 2.71a database release.
2.70a	September 2006	<ul style="list-style-type: none"> • Added top-level signal I/O diagrams (Figures 5-1 through 5-6). • Updated Signal, Register, and Parameter Descriptions in Chapters 5 through 9. • DesignWare Cores support for the <i>PCI Express 2.0 Base Specification</i> continues to evolve as the specification undergoes further revisions. Please refer to the DesignWare PCI Express Application Notes (2.0 Features chapter) for database release 2.70a implementation of the latest PCI Express 2.0 features. • Fixed documentation errata.
2.60a	June 2006	<p>DesignWare Cores support for the <i>PCI Express 2.0 Base Specification</i> continues to evolve as the specification undergoes further revisions. Please refer to the DesignWare PCI Express 2.0 Features Application note for database release 2.60a implementation of the latest PCI Express 2.0 features.</p> <p>There are no other documentation revisions associated with the 2.60a database release.</p>
2.50a	March 2006	<ul style="list-style-type: none"> • Added tables. (1) Single-, Multiple-, and Segmented-Buffer configurations: Advantages and Disadvantages (2) Single-, Multiple-, and Segmented-Buffer configurations: Examples • Updated “ELBI Protocol Rules” section. • Updated “Device Serial Number Capability Register” • Updated “Device Serial Number Capabilities Registers” section. • Updated “Implementation Guidelines” chapter. • Fixed documentation errata.

Preface

Book Organization

The DesignWare Cores for PCI Express include the following products:

- ❖ PCI Express Dual Mode (DM) core
- ❖ PCI Express Root Complex (RC) core
- ❖ PCI Express End Point (EP) core
- ❖ PCI Express Switch (SW) core



Note Throughout this document, content that is specific to a core is identified by using the abbreviated core name; for example, DM, RC, EP, SW. Content that applies to the DM core specifically in either RC mode or EP mode is further qualified by using “RC Mode” or “EP Mode.” Content that applies to all cores is not identified as being specific to any core; rather, the term “core” or “cores” is used.

The chapters of this manual are organized as follows:

Chapter 1 “[Architecture](#)” describes the core architecture and the functions performed by the major internal components.

Chapter 2 “[Core Operations](#)” describes the functional operation of the core.

Chapter 3 “[Signal Descriptions](#),” provides detailed descriptions of the top-level interface signals and protocols.

Chapter 4 “[Registers](#),” describes the core implementation of PCI Express configuration registers.

Chapter 5 “[Parameters](#),” provides detailed descriptions of the hardware configuration parameters.

Appendix A “[Core Frequencies and Sizes](#),” provides supported core frequencies and sizes.

Appendix B “[AHB Bridge Module](#)” provides operations, signal descriptions, registers, and parameters for the optional AHB Bridge Module.

Appendix C “[AXI Bridge Module](#)” provides operations, signal descriptions, registers, and parameters for the optional AXI Bridge Module.

Appendices D through M provide a collection of application notes that deal with a variety of PCI Express design topics. Each application note includes an introduction to the design topic and a discussion of how to solve the problem or implement the PCIe feature using the Synopsys digital and mixed-signal IP for PCIe. Each application note can be used independently and does not depend on information in other application notes.

The *DesignWare Cores PCI Express User Manual* includes the following information:

Chapter 1, "Product Overview" provides a general description of the core.

Chapter 2, "Building and Verifying Your Core" describes how to configure, synthesize, and simulate the core.

Chapter 3, "Implementation Guidelines" provides guidelines for implementing a PCIe device with the PCIe IP core instantiated, and provides a reference design and an FPGA demo.

Chapter 4, "Integration" provides guidelines for hardware integration and verification.

Appendix A, "Unit Test Environment" describes the unit-level testbench used internally by Synopsys to verify the PCIe IP core.

Appendix B: "Installation and Workspace Files," describes the product directory structure.

Reference Documentation

Refer to the *Guide to Documentation for DesignWare Cores PCI Express* for the following reference documentation:

- ❖ DesignWare Cores PCI Express Documentation
- ❖ DesignWare Cores PCI Express PHY Documentation
- ❖ DesignWare Verification IP (VIP) Documentation
- ❖ Other PCI Documentation

After installing the core, the DWC PCI Express documents can be found under:
\$DESIGNWARE_HOME/iip/DWC_pcie/latest/doc.

Terms and Abbreviations

The following terms are used throughout this document:

DM	PCI Express Dual Mode (DM) core
RC	PCI Express Root Complex (RC) core
EP	PCI Express End Point (EP) core
SW	PCI Express Switch (SW) core
PCIe	PCI Express
CXPL	Common Xpress Port Logic: An internal Port Logic core module that implements the majority of the PCI Express protocol
x16	16 lanes (2.5 Gbps or 5.0 Gbps lanes)
x8	8 lanes (2.5 Gbps or 5.0 Gbps lanes)
x4	4 lanes (2.5 Gbps or 5.0 Gbps lanes)
x2	2 lanes (2.5 Gbps or 5.0 Gbps lanes)



x1	1 lane (2.5 Gbps or 5.0 Gbps lane)
PIPE	PHY Interface for the PCI Express Architecture
NW	Number of double words; 1 stands for a 32-bit DWORD
DW	Data width: 32, 64, or 128 bits
NF	Number of functions; 1 stands for one function
TLP	Transaction Layer Packet
DLLP	Data Link Layer Packet
VC	Virtual Channel
BAR	Base Address Register

Customer Support

For customer support:

- ❖ Enter a call through SolvNet.
 - ◆ Go to <https://solvnet.synopsys.com/ManageCase?ccf=1> and provide the requested information, including:
 - ❖ Product: DesignWare Cores
 - ❖ Sub Product: PCI Express
 - ❖ Version: 3.10a or earlier
 - ❖ Subject: core
 - ◆ Provide the following additional information, if applicable:
 - ❖ For environment setup problems, run the debug_info command in the coreConsultant GUI and include the text file generated.
 - ❖ For configuration failures, include the error messages that appear in the coreConsultant GUI console pane.
 - ❖ For simulation failures, include a text file with your specific configuration. Generate this text file using the coreConsultant GUI's write_batch_script command. Also include the log file from the <workspace>/simulation/ directory, the log file for the specific test, and a VPD/VCD waveform dump file.
 - ❖ For synthesis failures, include the log file from <workspace>/syn/ directory.
- ❖ Send an e-mail message to support_center@synopsys.com.
 - ◆ Include the Product Name, Sub Product name, and Version (product release number) in our e-mail so it can be routed correctly.
 - ◆ Provide the following additional information, if applicable:
 - ❖ For environment setup problems, run the debug_info command in the coreConsultant GUI and include the text file generated.
 - ❖ For configuration failures, include the error messages that appear in the coreConsultant GUI console pane.

- ❖ For simulation failures, include a text file with your specific configuration. Generate this text file using the coreConsultant GUI's write_batch_script command. Also include the log file from the <workspace>/simulation/ directory, the log file for the specific test, and a VPD/VCD waveform dump file.
- ❖ For synthesis failures, include the log file from <workspace>/syn/ directory.
- ❖ Telephone your local support center:
 - ◆ United States:
Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific Time, Monday through Friday.
 - ◆ Canada:
Call 1-650-584-4200 from 7 AM to 5:30 PM Pacific Time, Monday through Friday.
 - ◆ All other countries:
http://www.synopsys.com/support/support_ctr

1

Architecture

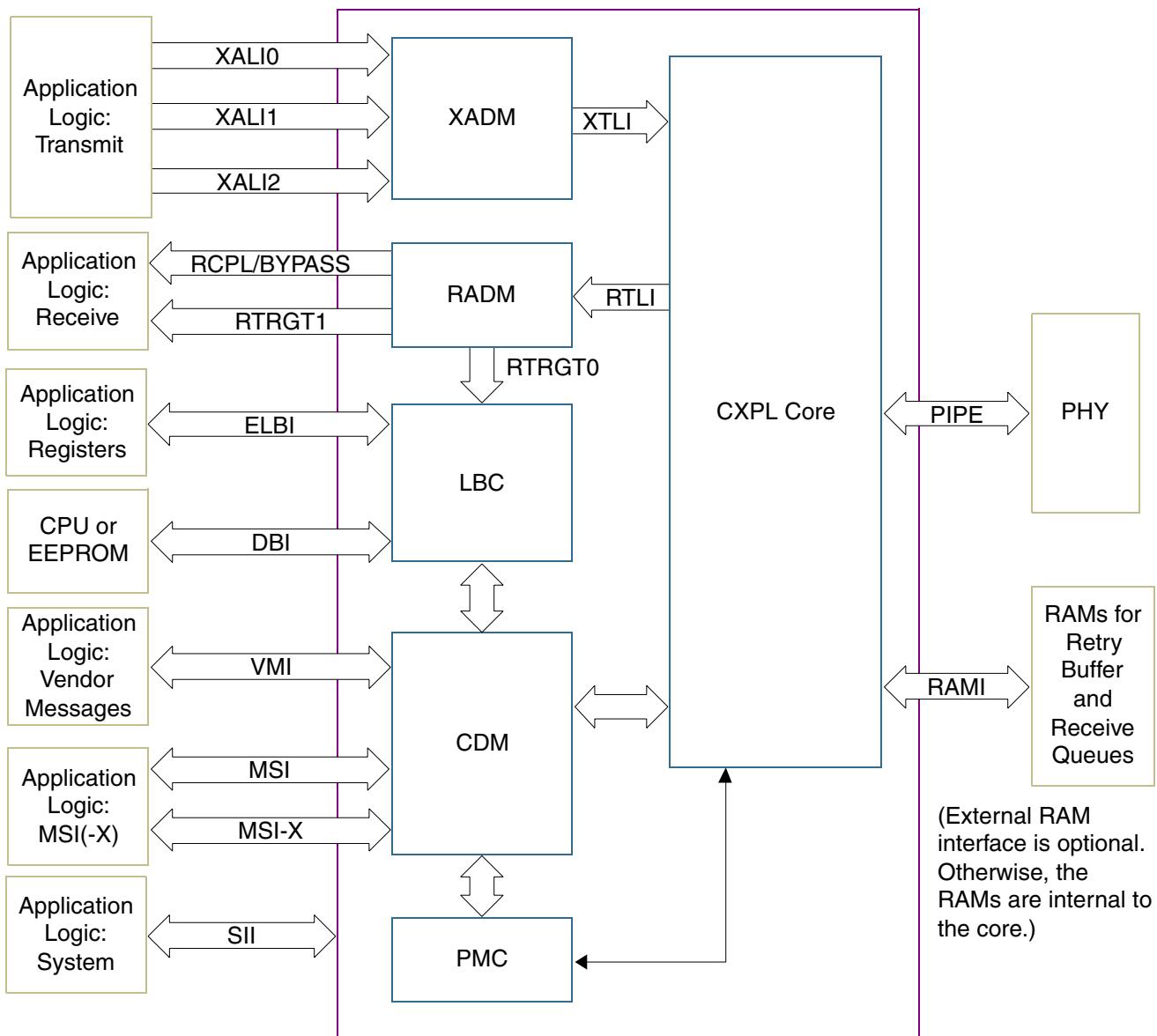
[Figure 1-1](#) through [Figure 1-4](#) show the top-level structure of the PCIe DM, RC, EP, and SW cores, respectively. The Common Xpress Port Logic (CXPL) module implements the basic functionality for the PCI Express Physical, Link, and Transaction Layers.

In addition to the CXPL, there are several top-level modules that provide the configuration and mode-specific features:

- ❖ Transmit application-dependent module (XADM)
- ❖ Receive application-dependent module (RADM)
- ❖ Configuration-dependent module (CDM)
- ❖ Power management controller (PMC)
- ❖ Local bus controller (LBC) (DM/EP/SW only)
- ❖ Message generation (MSG_GEN)
- ❖ Hot plug control (hotplug_ctrl) (DM/RC/SW only)



The Message Generation and Hot Plug Control modules reside at the top level of the core, but are not shown in [Figure 1-1](#) through [Figure 1-4](#). The Hot Plug Control module is in the DM, RC, and SW cores only (not in the EP core).

Figure 1-1 Core Block Diagram (DM)

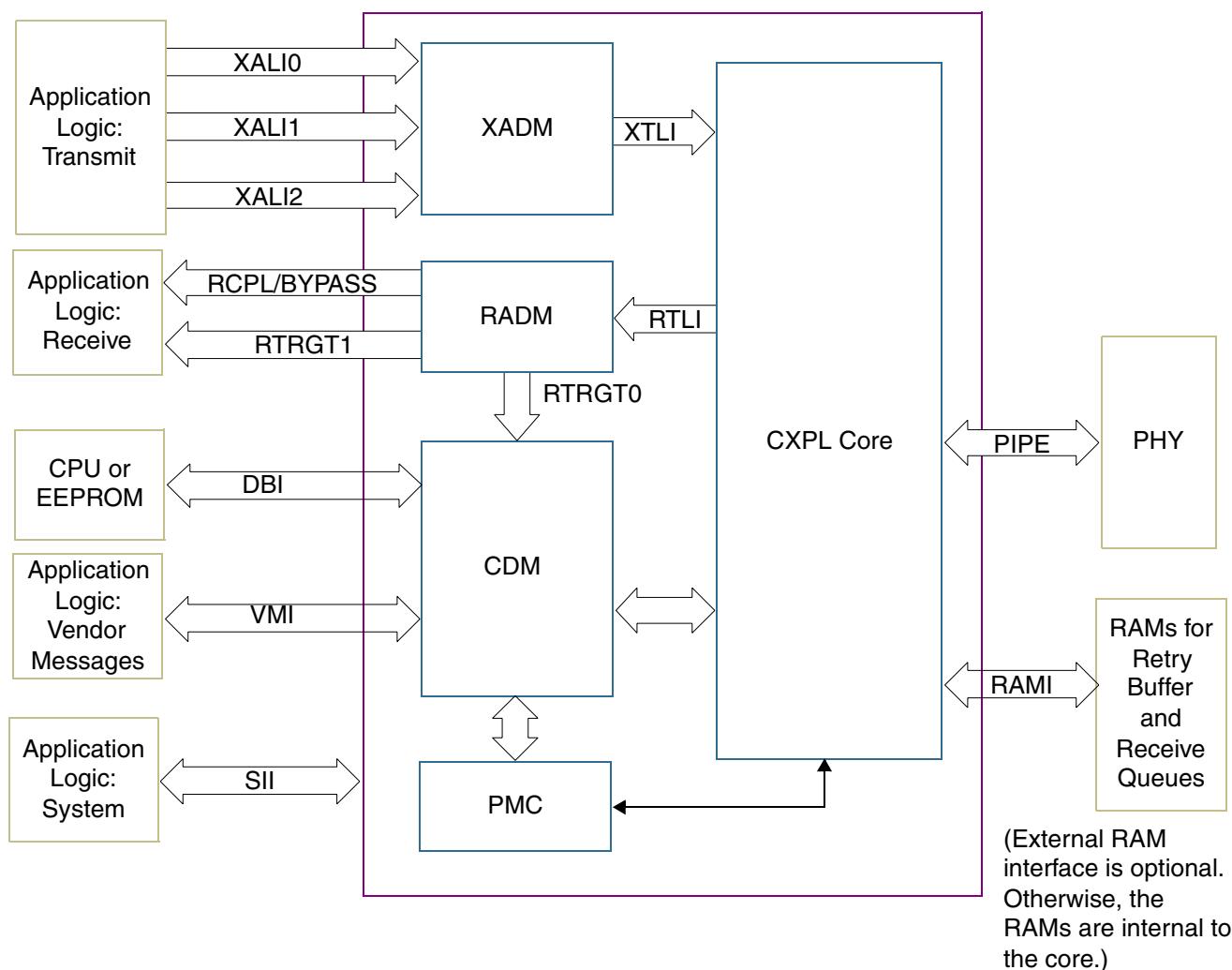
**Figure 1-2 Core Block Diagram (RC)**

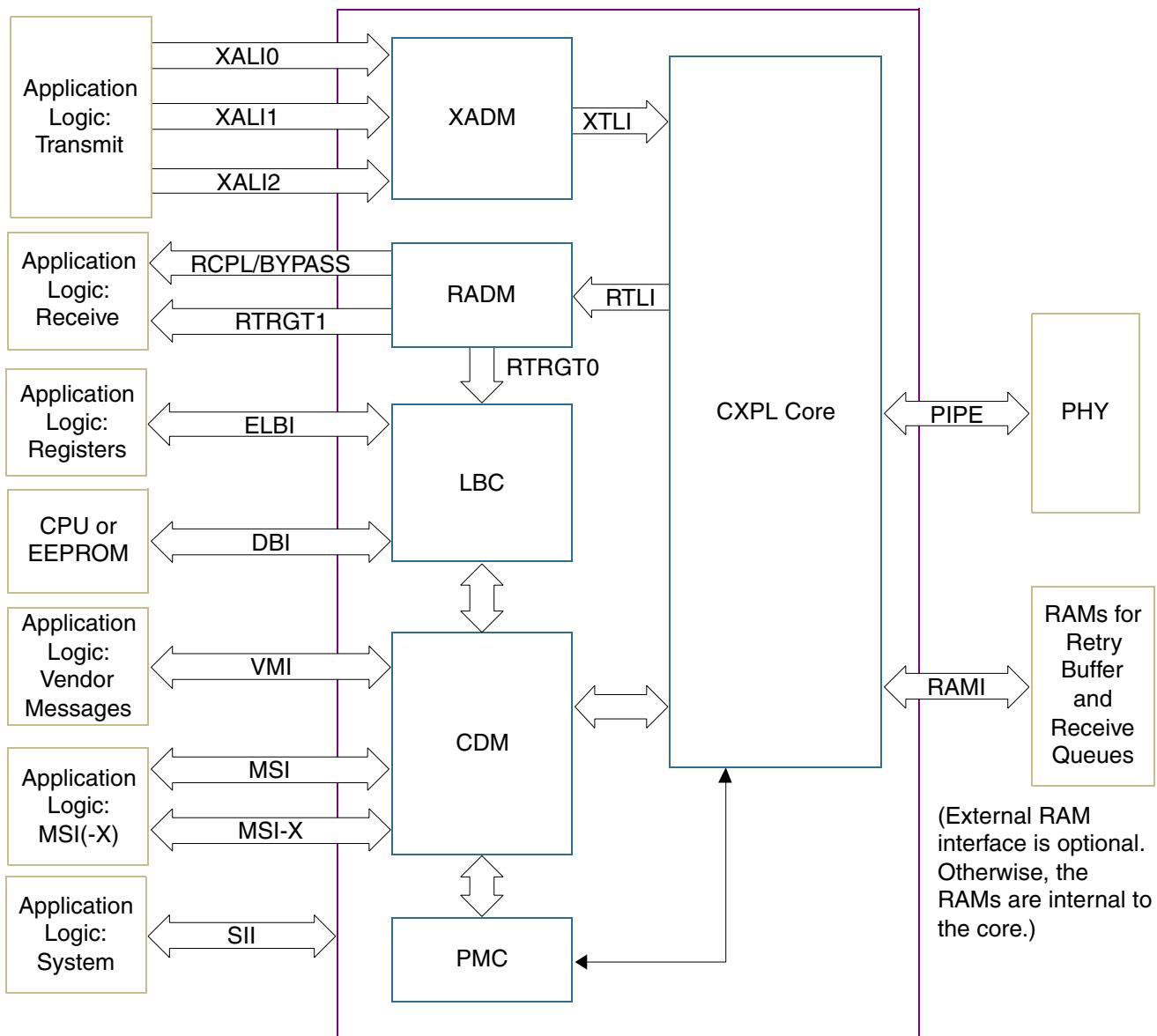
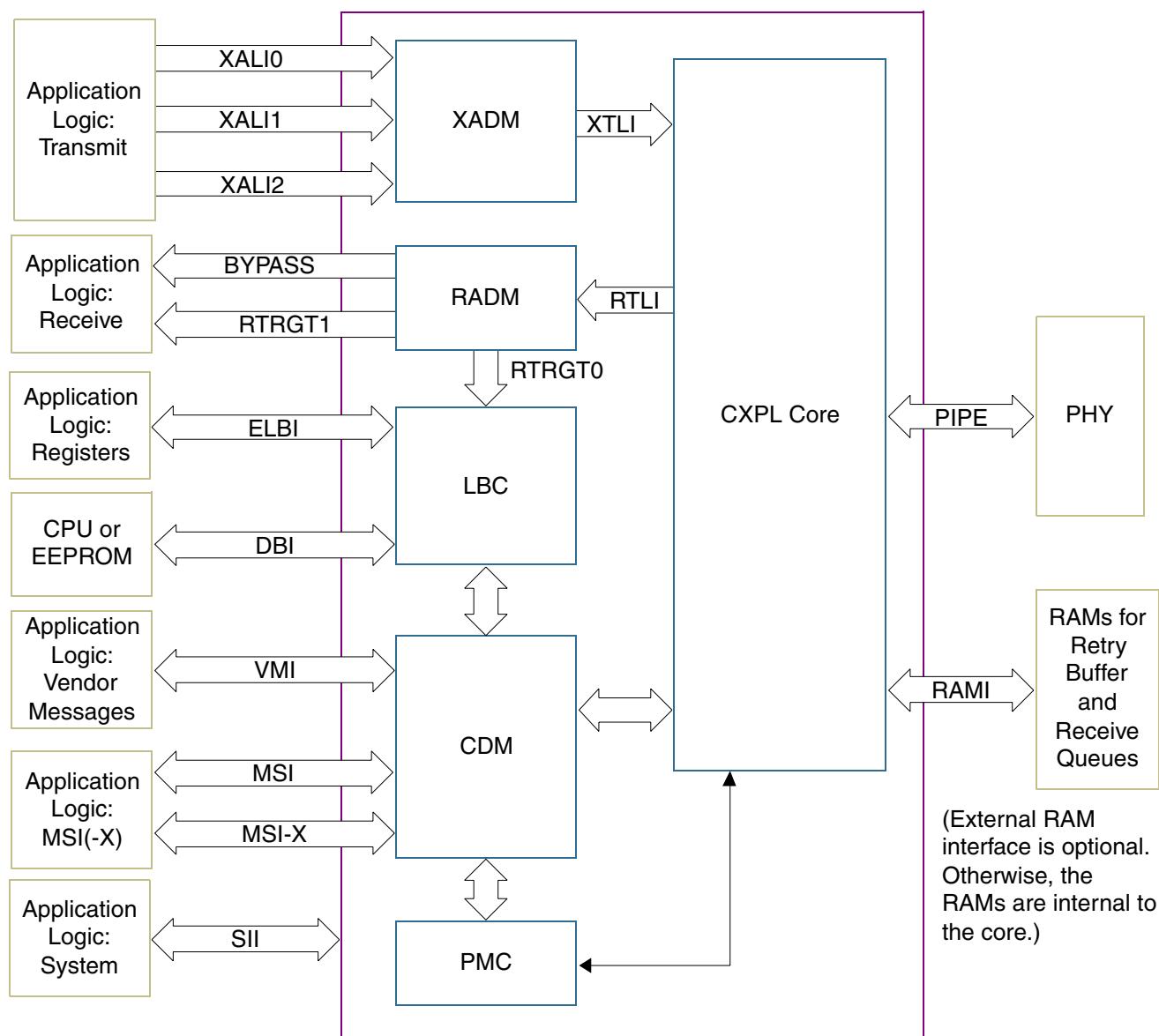
Figure 1-3 Core Block Diagram (EP)

Figure 1-4 Core Block Diagram (SW)

1.1 Common Xpress Port Logic (CXPL)

The CXPL module implements a large portion of the Transaction Layer logic, all of the Data Link Layer logic, and the MAC portion of the Physical Layer, including the Link Training and Status State Machine (LTSSM). The CXPL connects to the external PHY through the PIPE.

Important aspects of the CXPL and overall core implementation include:

- ❖ Layer 3 (Transaction Layer) functionality is split between the XADM, RADM, CDM, and CXPL.
- ❖ Layer 2 and Layer 1 (Data Link and Physical Layers) are partitioned.
- ❖ The Physical Layer is split across the PIPE such that the MAC functionality is in the core and the PHY functionality is implemented in the PIPE-compliant PHY.
- ❖ Receive and transmit path functionality is decoupled except where communication between the two is required (such as Flow Control and other low-level Link management functions).



Note

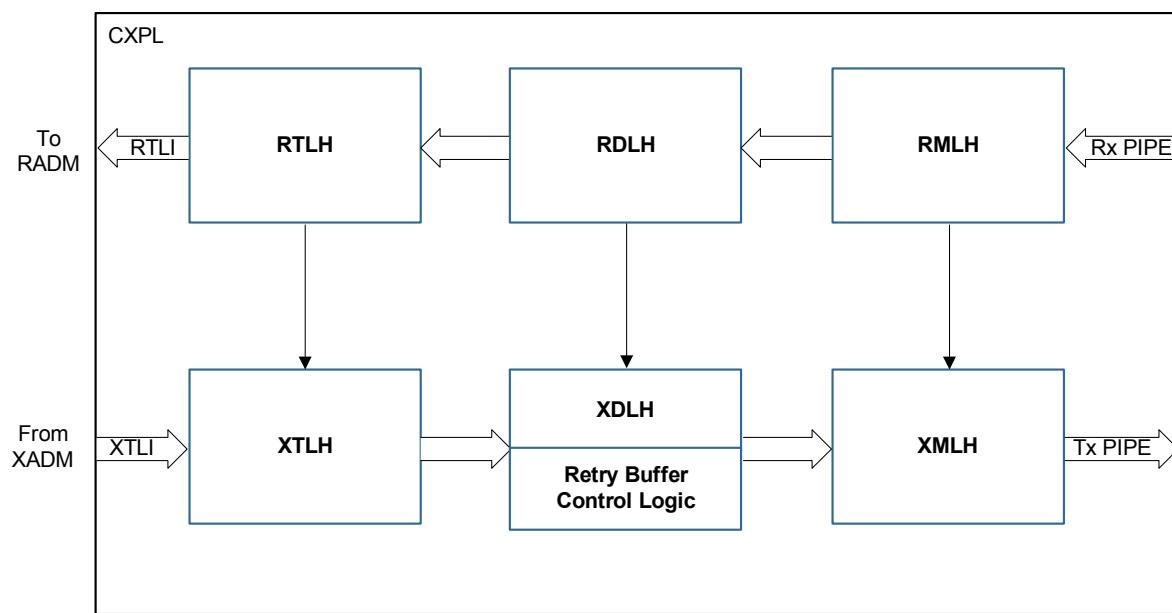
The PHY module resides outside of the core, interfacing through the standard PIPE. An example SerDes-dependent module (SDM), which includes PCS functionality and elastic buffering, is available for users who prefer to build a PIPE-compatible PHY based on a compatible SerDes component.

As mentioned in the *DesignWare Cores PCI Express User Manual*, Chapter 1, “Product Overview,” if the Synopsys PHY is used with the PCI Express core, then the PHY will be included inside the core, the PIPE interface will be internal to the core, and the external interface for the PHY will be a simple interface consisting of one or more transmit and receive differential pairs. Otherwise, if the Synopsys PHY is not chosen, the PIPE is the interface between the PHY macro and the core.

CXPL contains six modules, three for transmission and three for reception, as shown in [Figure 1-5](#).

- ❖ RTLH: Receive Transaction Layer Handler
- ❖ XTLH: Transmit Transaction Layer Handler
- ❖ RDLH: Receive Data Link Layer Handler
- ❖ XDLH: Transmit Data Link Layer Handler
- ❖ RMLH: Receive MAC Layer Handler
- ❖ XMLH: Transmit MAC Layer Handler

CXPL is consistent with the *PCI Express Specification* with regards to the physical layer, data link layer and transaction layer.

**Figure 1-5 CXPL Module Block Diagram**

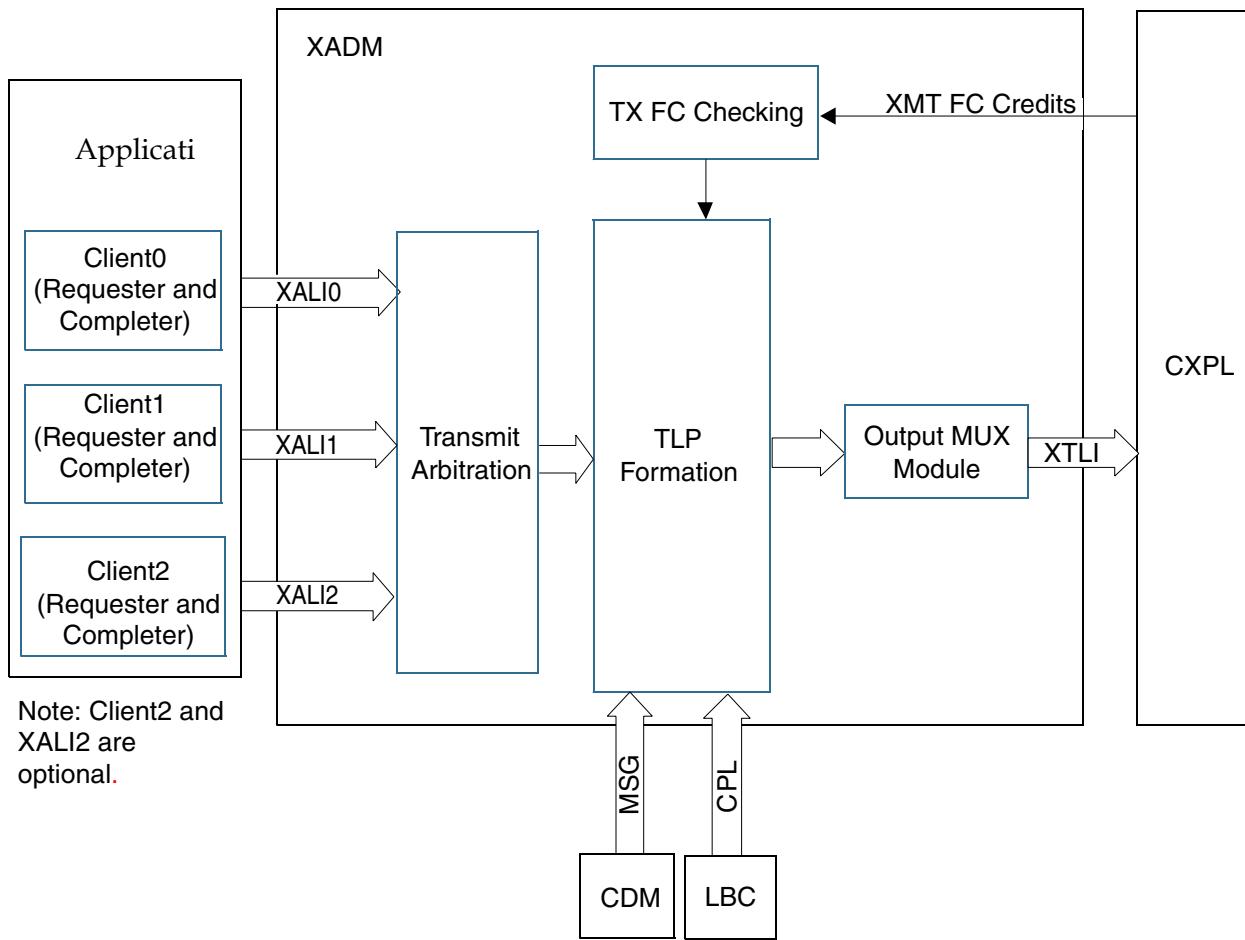
1.2 Transmit Application-Dependent Module (XADM)

The XADM sits between the application logic and the CXPL core and implements the mode-specific functionality of the PCI Express Transaction Layer for packet transmission.

[Figure 1-6](#) is a block diagram of the XADM. Its functions include arbitration, TLP formation, and credit checking.

The core does not implement transmit queues (other than the retry buffer). Depending on system design, an externally-implemented transmit queue can be used to handle rate matching if the CXPL and application transfer rates are different.

Figure 1-6 XADM Block Diagram



1.2.1 Arbitration: DM

XADM provides the arbitration of TLP transmission between the following:

- ❖ The transmit client interfaces (XALI0, XALI1, and the optional XALI2)
- ❖ Internally generated Messages from the CDM, triggered by PME, INTx (EP mode), errors, or application logic
- ❖ Internally generated Completions:



- ◆ EP mode: Internally generated Completions are responses for type 0 Configuration Read and Write Requests from upstream components, memory or I/O-mapped application register space Read and Write Requests, or responses to error conditions (Unsupported Requests).
- ◆ RC mode: Internally generated Completions are Unsupported Request or Completer Abort, as required by the incoming Request filtering function of the RADM.

In general, all internally generated TLP requests have higher priority than client interfaces.

For details about how the arbitration methods work and how to configure the arbitration methods, see “[Transmit TLP Arbitration](#)” on page [56](#).

Usage models for the client interfaces include:

- ❖ A master is connected to each client interface (EP mode only).

The XADM arbitrates among client interfaces. There is no guarantee that order will be preserved among client interfaces. In some cases, a Requester may consider implementing some ordering rules in the application logic, for example, holding off a Memory Read transaction until the Memory Write transaction is completed.

- ❖ A master is connected to Client1. A target completer is connected to Client0. A second master is optionally connected to Client2.

The XADM arbitrates around each client interface. There is no guarantee that order will be preserved among client interfaces.

- ❖ A master for posted traffic is connected to Client0. A master for non-posted traffic is connected to Client1. A target completer is connected to Client2 interface. This is a model with one type of TLP per client (Posted, Non-Posted, Completion).

Additional uses of the Client0, Client1, and optional Client2 interfaces are possible. See the *DesignWare Cores PCI Express User Manual*, Chapter 4, “Integration” for additional examples.

1.2.2 Arbitration: RC

XADM provides the arbitration of TLP transmission between the following:

- ❖ The transmit client interfaces (XALI0, XALI1, and the optional XALI2)
- ❖ Internally generated Messages from the CDM, errors, or application logic
- ❖ Internally generated Completions:
 - ◆ Internally generated Completions are Unsupported Request or Completer Abort, as required by the incoming Request filtering function of the RADM.

In general, all internally generated TLP requests have higher priority than client interfaces.

For details about how the arbitration methods work and how to configure the arbitration methods, see “[Transmit TLP Arbitration](#)” on page [56](#).

Usage models for the client interfaces include:

- ❖ A master is connected to Client1. A target completer is connected to Client0. A second master is optionally connected to Client2.

The XADM arbitrates around each client interface. There is no guarantee that order will be preserved among client interfaces.

- ❖ A master for posted traffic is connected to Client0. A master for non-posted traffic is connected to Client1. A target completer is connected to Client2 interface. This is a model with one type of TLP per client (Posted, Non-Posted, Completion).

Additional uses of the Client0, Client1, and optional Client2 interfaces are possible. See the *DesignWare Cores PCI Express User Manual*, Chapter 4, “Integration” for additional examples.

1.2.3 Arbitration: EP

XADM provides the arbitration of TLP transmission between the following:

- ❖ The transmit client interfaces (XALI0, XALI1, and the optional XALI2)
- ❖ Internally generated Messages from the CDM, triggered by PME, INTx, errors, or application logic
- ❖ Internally generated Completions:
 - ◆ Internally generated Completions are responses for type 0 Configuration Read and Write Requests from upstream components, memory or I/O-mapped application register space Read and Write Requests, or responses to error conditions (Unsupported Requests).

In general, all internally generated TLP requests have higher priority than client interfaces.

For details about how the arbitration methods work and how to configure the arbitration methods, see “[Transmit TLP Arbitration](#)” on page [56](#).

Usage models for the client interfaces include:

- ❖ A master is connected to each client interface.
The XADM arbitrates among client interfaces. There is no guarantee that order will be preserved among client interfaces. In some cases, a Requester may consider implementing some ordering rules in the application logic, for example, holding off a Memory Read transaction until the Memory Write transaction is completed.
- ❖ A master is connected to Client1. A target completer is connected to Client0. A second master is optionally connected to Client2.
The XADM arbitrates around each client interface. There is no guarantee that order will be preserved among client interfaces.
- ❖ A master for posted traffic is connected to Client0. A master for non-posted traffic is connected to Client1. A target completer is connected to Client2 interface. This is a model with one type of TLP per client (Posted, Non-Posted, Completion).

Additional uses of the Client0, Client1, and optional Client2 interfaces are possible. See the *DesignWare Cores PCI Express User Manual*, Chapter 4, “Integration” for additional examples.

1.2.4 Arbitration: SW

XADM provides the arbitration of TLP transmission between the following:

- ❖ The transmit client interfaces (XALI0, XALI1, and the optional XALI2)
- ❖ Internally generated Messages from the CDM, triggered by PME, INTx, errors, or application logic
- ❖ Internally generated Completions:
 - ◆ Internally generated Completions are responses for type 0 Configuration Read and Write Requests from upstream components, memory or I/O-mapped application register space Read and Write Requests, or responses to error conditions (Unsupported Requests).

- ◆ Internally generated Completions are Unsupported Request or Completer Abort, as required by the incoming Request filtering function of the RADM.

In general, all internally generated TLP requests have higher priority than client interfaces.

For details about how the arbitration methods work and how to configure the arbitration methods, see “[Transmit TLP Arbitration](#)” on page [56](#).

Usage models in the Switch are different than in the other cores. It is up to the Switch application designer to select whether to include all three transmit client interfaces and how to use them.

Usage models for the transmit client interfaces in the Switch include:

- ❖ Using XALI0 for traffic passing through the Switch and using XALI1 for internally-generated traffic
 - ❖ Using one interface for each TLP type (Posted, Non-Posted, and Completion)
 - ❖ Using only a single client interface and perform all arbitration external to the XADM.
- The XADM arbitrates around each client interface. There is no guarantee that order will be preserved among client interfaces.

Additional uses of the Client0, Client1, and optional Client2 interfaces are possible. See the *DesignWare Cores PCI Express User Manual*, Chapter 4, “Integration” for additional examples.

1.2.5 Credit Checking

The core checks that enough FC credits are available in the remote device for the specific type of transaction (P, NP, CPL) before allowing a transmission of a TLP. TLPs that passed the credit check are arbitrated according to the supported arbitration method. Internally generated Completions and Messages are also gated by the arbitration logic, though at highest priority, and must also pass the FC credit test before they are accepted for transmission.

If the application is using a single transmit client interface for more than one Request type (for example, Posted and Non-Posted), and the current Request (for example, a Posted Request) is being blocked due to lack of available FC credits, then that client interface is effectively blocked from sending other Requests (for example, Non-Posted) even though credits may be available for that type.

To avoid this situation, the application can use different transmit client interfaces for different Request types (for example, Client0 for Posted Requests, Client1 for Non-Posted Requests, and Client2 for Completions).

Another way the application can avoid this situation is to monitor the current FC credit availability on the `xadm_*_cdts` outputs from the core and only generate Requests for which FC credits are available. When using `xadm_*_cdts` to monitor credit availability, the application must consider that it is possible that the core may generate a Message TLP (which would change the number of credits available) between the time the application samples `xadm_*_cdts` and when the application generates the Request. See the *DesignWare Cores PCI Express Reference Manual*, Chapter 1, “Signal Descriptions” and *DesignWare Cores PCI Express User Manual*, Chapter 3, “Implementation Guidelines” for more details on these signals.

1.3 Receive Application-Dependent Module (RADM)

1.3.1 Receive Application-Dependent Module (RADM): DM / RC / EP

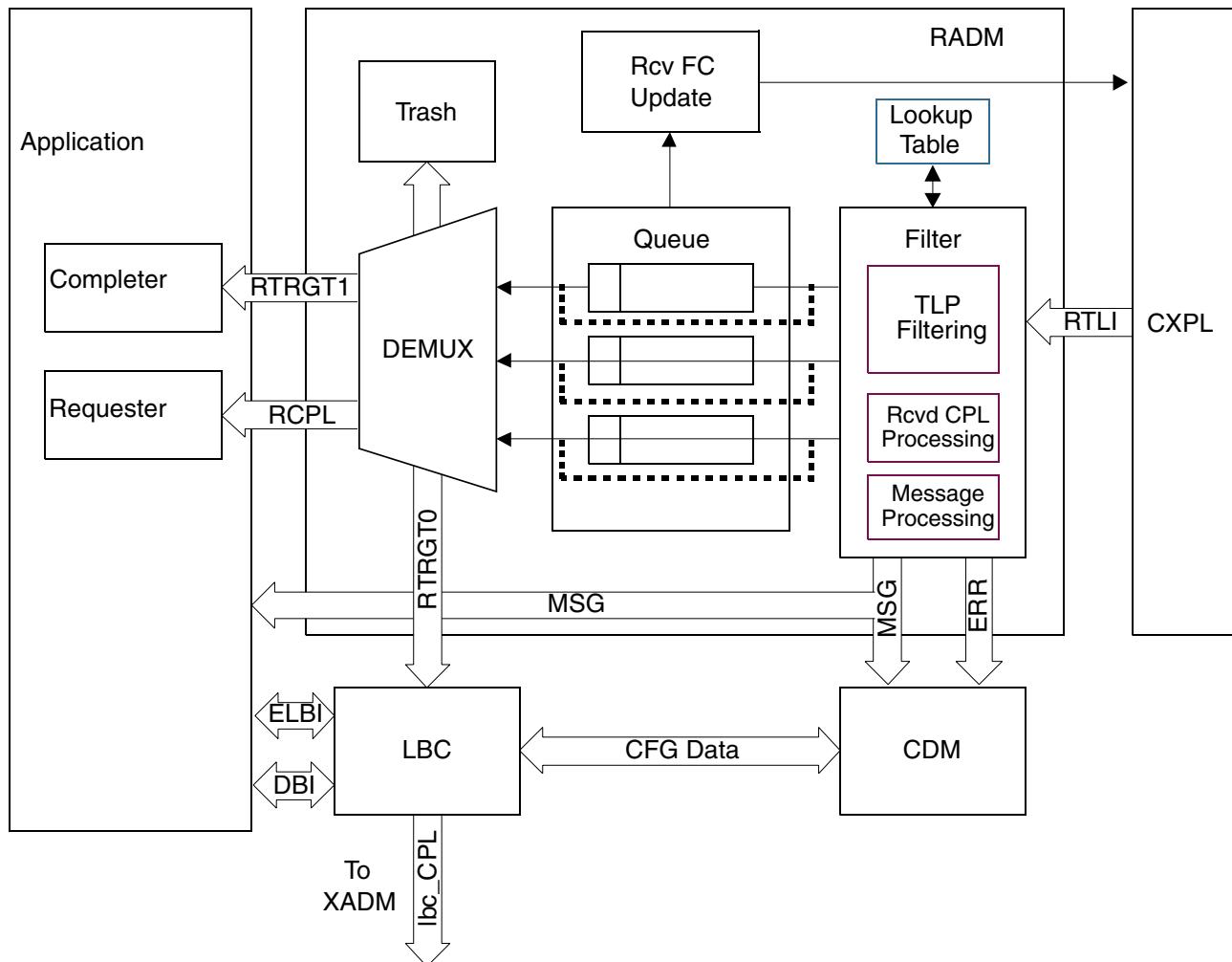
The RADM sits between the application logic and the CXPL core and implements the mode-specific functionality of the PCI Express Transaction Layer for TLP packet reception.

[Figure 1-7](#) shows a block diagram of the RADM in the DM, RC, and EP cores. The RADM serves four major functionalities as following:

- ❖ Sort/Filter received TLPs
- ❖ Completion lookup table
- ❖ Provide queuing (or bypass) of the received TLP
- ❖ Output received TLP to the core's receive interface (Demux function)

The filtering rules and routing for all TLP receive options are configurable for all TLPs received. For details about filtering and routing of incoming TLPs, see “[Receive Filtering](#)” on page [60](#).

Figure 1-7 RADM Block Diagram: DM / RC / EP





1.3.2 Receive Application-Dependent Module (RADM): SW

The RADM sits between the application logic and the CXPL core and implements the mode-specific functionality of the PCI Express Transaction Layer for TLP packet reception.

Figure 1-8 shows a block diagram of the RADM in the Switch. The RADM serves three major functionalities as following:

- ❖ Sort/Filter received TLP
- ❖ Provide queuing or bypass of the received TLP
- ❖ Output received TLP to the core's receive interface (Demux function)

The filtering rules and routing for all TLP receive options are configurable for all TLPs received. For details about filtering and routing of incoming TLPs, see “[Receive Filtering](#)” on page 60.

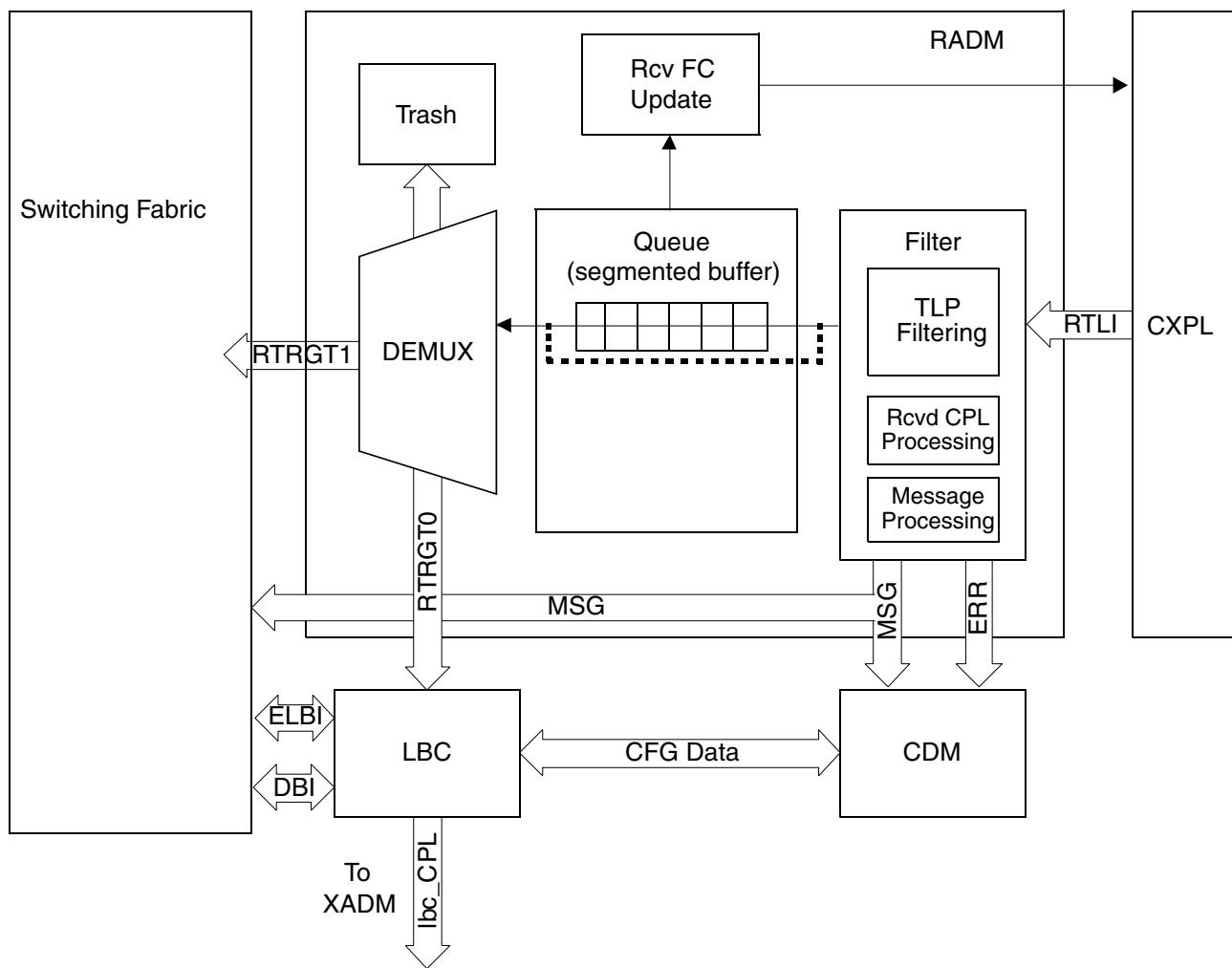


Note The Completion lookup table (LUT) is not normally used in the SW application because there are typically no locally generated requests.



Note Bypass mode is not typically used in a Switch since it implies packet reordering.



Figure 1-8 RADM Block Diagram: SW



1.3.3 Received Completion TLP Processing: DM / RC / EP

Received Completions are filtered against the completion lookup table content before presenting the Completion to the queue.

The RADM also implements a Completion time-out mechanism and notifies the application when a Completion timeout occurs. Typically, infinite Completion credits are advertised and the received completion is configured in bypass mode which means that there is no queue in the core to store completions.



Note It is fully expected that the application will have enough buffering space ready for its requests, so no backpressure mechanism is needed. By default, the Completion queue operates in bypass mode.

Completions can be configured in store and forward mode if the application has chosen to do so. If a completion lookup has failed or other completion filtering has failed, the core will assert an abort signal at the end of the transaction. If the core is configured to have Completions in bypass mode, it is the application's responsibility to roll back any actions at the application's queue when an abort signal is asserted. If the core is configured with Completions enqueued, the Completion will be discarded by the core and flow control credits will be updated, as necessary, when an abort signal is detected.



1.3.4 Message Processing

The RADM filter provides a Message interface (grouped as part of the SII) to handle the Message TLPs received from the upstream component. The RADM filter processes the Message and decodes the header before sending it to the application logic on the SII. You can also select a configuration option to send the entire Message TLP to the application in addition to providing the decoded Message on the SII.

1.3.5 Posted and Non-Posted Request and Completion TLP Processing

The RADM filter passes the Posted and Non-Posted Request and Completion transactions (such as Write Transactions and Memory Reads) directly to the application through the RTRGT1 interface or to RTRGT0 for internal modules, as determined by the filtering and routing rules for the current operating mode, as described in “[Receive Filtering](#)” on page 60.

The RADM filter segregates Posted and Non-Posted TLPs into valid supported and valid un-supported Requests, and forwards them to the queue. The filter processes each Request and determines each TLP's destination along with other controls that may be needed to generate TLPs.

For Requests that the core forwards to the RTRGT1 or Bypass interface, the application must process the Request and generate the Completion.

For Requests that the core forwards to RTRGT0, the core automatically generates the Completion.

EP mode: The core automatically executes any required ELBI access before generating the Completion.

1.3.6 **Lookup Table: DM / RC / EP**

The RADM filter also contains a lookup table (LUT), which is used to store information associated with outbound Non-Posted Requests. The RADM uses the lookup table for incoming Completion tracking and Completion timeout monitoring.

1.3.7 **RADM Demux: DM / RC / EP**

The RADM demux is designed to mux out a received TLP to the TRGT1 and CPL/BYPASS interfaces from single queue or multiple queue configurations. The filter determines the destination and the action for each TLP, then sends this to the queue. The demux decides whether to discard or forward the TLP onto the TRGT1 or TRGT0 interfaces.

1.3.8 **RADM Demux: SW**

The RADM demux is designed to mux out a received TLP to the TRGT1 and CPL/BYPASS interfaces from single queue configurations. The filter determines the destination and the action for each TLP, then sends this to the queue. The demux decides whether to discard or forward the TLP onto the TRGT1 or TRGT0 interfaces.

1.4 Configuration-Dependent Module (CDM)

1.4.1 Configuration-Dependent Module (CDM): DM

The CDM implements the standard PCI Express configuration space and the core-specific register space. The CDM also requests the Message generation module to send Messages, as required, including MSI and interrupts.

In RC mode, the specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 1 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ VPD (Vital Product Data) Capability
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

In EP mode, the specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 0 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ VPD (Vital Product Data) Capability
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

The configured device type (determined by the device_type[3:0] input signal) affects the behavior of the Message generation engine, error reporting mechanism, as well as some PCI Express configuration space registers.



The CDM communicates with application's host bus controller through the DBI. The host bus controller controls accesses to registers within each CDM in multiple instances of the core in a multi-port design.

1.4.2 Configuration-Dependent Module (CDM): RC

The CDM implements the standard PCI Express configuration space and the core-specific register space. The CDM also requests the Message generation module to send Messages, as required, including MSI and interrupts.

The specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 1 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ VPD (Vital Product Data) Capability
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

The configured device type (determined by the device_type[3:0] input signal) affects the behavior of the Message generation engine, error reporting mechanism, as well as some PCI Express configuration space registers.

The CDM communicates with application's host bus controller through the DBI. The host bus controller controls accesses to registers within each CDM in multiple instances of the core in a multi-port design.



1.4.3 Configuration-Dependent Module (CDM): EP

The CDM implements the standard PCI Express configuration space and the core-specific register space. The CDM also requests the Message generation module to send Messages, as required, including MSI and interrupts.

The specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 0 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ VPD (Vital Product Data) Capability
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

The configured device type (determined by the device_type[3:0] input signal) affects the behavior of the Message generation engine, error reporting mechanism, as well as some PCI Express configuration space registers.

The CDM communicates with application's host bus controller through the DBI. The host bus controller controls accesses to registers within each CDM in multiple instances of the core in a multi-port design.



1.4.4 Configuration-Dependent Module (CDM): SW

The CDM implements the standard PCI Express configuration space and the core-specific register space. The CDM also requests the Message generation module to send Messages, as required, including MSI and interrupts.

In the upstream port of the Switch, the specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 0 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ Slot ID Capability Structure
 - ◆ VPD (Vital Product Data Capability)
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

In the downstream port of the Switch, the specific PCI Express configuration structures implemented in the CDM include the following:

- ❖ PCI-Compatible Configuration Registers (type 1 header)
- ❖ PCI Capability Structures:
 - ◆ PCI Power Management Capability Structure
 - ◆ MSI Capability Structure
 - ◆ MSI-X Capability Structure
 - ◆ VPD (Vital Product Data Capability)
- ❖ PCI Express Capability Structure
- ❖ PCI Express Extended Capabilities:
 - ◆ Advanced Error Reporting Capability
 - ◆ Virtual Channel Capability
 - ◆ Device Serial Number Capability
 - ◆ Power Budgeting Extended Capability

The configured device type (determined by the `device_type[3:0]` input signal) affects the behavior of the Message generation engine, error reporting mechanism, as well as some PCI Express configuration space registers.

The CDM communicates with application's host bus controller through the DBI. The host bus controller controls accesses to registers within each CDM in multiple instances of the core in a multi-port design.

1.5 Power Management Controller (PMC)

1.5.1 Power Management Controller (PMC): DM / RC / EP

The PMC module supports PCI software-compatible Power Management (PM) mechanisms and the native PCI Express Active State Power Management (ASPM). The PMC is the only module that must be powered by auxiliary power (Vaux) in the core when the core is in a lower power state. It is also the only module containing contexts that are resetable only at power-up.

The following features are implemented in the PMC module:

- ❖ ASPM support: L0s and L1
- ❖ Control of the LTSSM to perform Link power management: L0, L1, L2 and L3
- ❖ Software-controlled device PM states: D0, D1, and D3hot/cold
- ❖ Generation of PM Message transmission requests
- ❖ Control of beacon generation
- ❖ Side-band wake mechanism:
 - ◆ Supports application wake-up (for example, WOL and WAKE# signal support from the platform system)
 - ◆ Generates wake to request system to restore power and clock
- ❖ Power management event (PME) generation
- ❖ Output of current power state status to the application

1.5.2 Power Management Controller (PMC): SW

The PMC module supports PCI software-compatible Power Management (PM) mechanisms and the native PCI Express Active State Power Management (ASPM). The PMC is the only module that must be powered by auxiliary power (Vaux) in the core when the core is in a lower power state. It is also the only module containing contexts that are resetable only at power-up.

The following features are implemented in the PMC module:

- ❖ Support both upstream and downstream Port configurations
- ❖ ASPM support: L0s and L1
- ❖ Control of the LTSSM to perform Link power management: L0, L1, L2 and L3
- ❖ Software-controlled device PM states: D0, D1, and D3hot/cold
- ❖ Generation of PM Message transmission requests
- ❖ Control of beacon generation
- ❖ Side-band wake mechanism:
 - ◆ Supports application wake-up (for example, WOL and WAKE# signal support from the platform system)
 - ◆ Generates wake to request system to restore power and clock
- ❖ Power management event (PME) generation
- ❖ Output of current power state status to the application
- ❖ Device power state is managed by accumulating link state from all Switch ports.



1.6 Local Bus Controller (LBC) (DM / EP / SW)

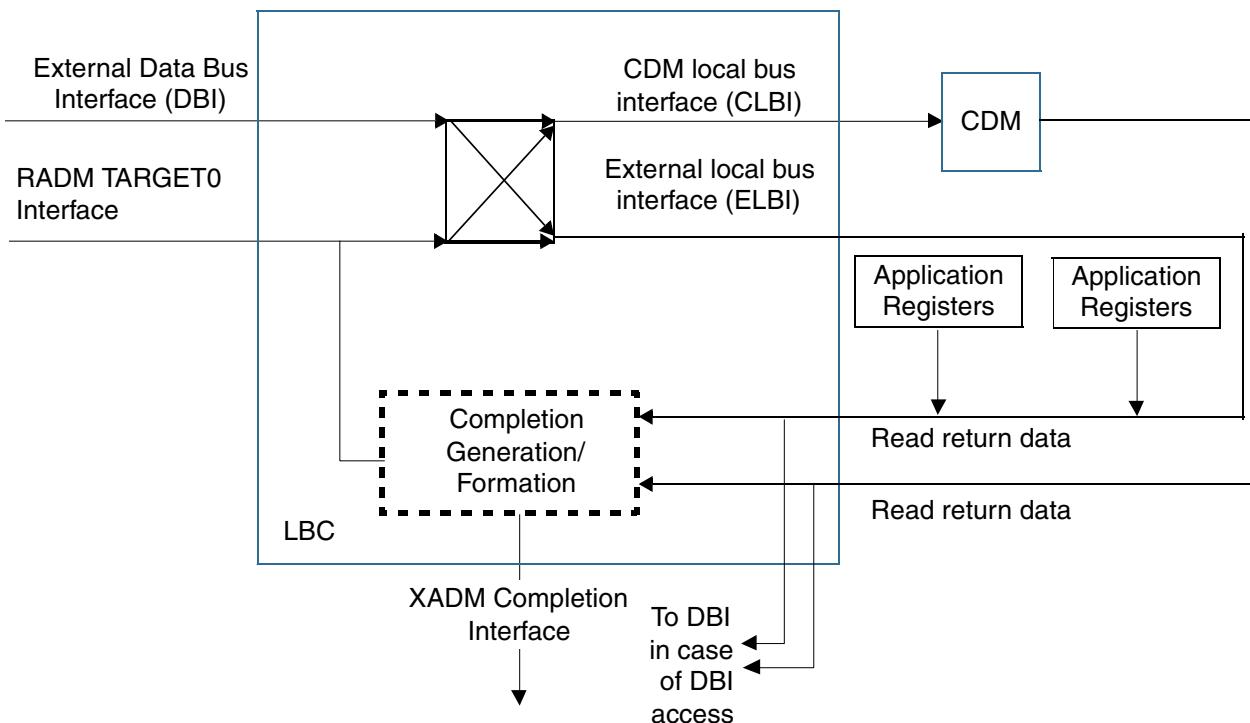
The LBC module provides a mechanism for accessing the core registers and external application registers. The LBC provides a switching function for the core's or application's RD/WR registers between the DBI, ELBI, TRGT0 and CDM LBC interfaces.

The LBC provides the following functions:

- ❖ Supports DBI RD/WR to CDM registers.
- ❖ Supports TRGT0 RD/WR (received PCIe RD/WR TLPs) to CDM registers.
- ❖ Supports DBI RD/WR to application registers residing on the ELBI.
- ❖ Supports TRGT0 RD/WR (received PCIe RD/WR TLPs) to application registers at ELBI.
- ❖ Generates a PCIe completion of a received PCIe RD/WR request that is targeted to CDM registers or application registers residing on the ELBI.
- ❖ Supports both 32-bit and 64-bit addresses for RD/WR requests.

[Figure 1-9](#) shows the internal function diagram of the local bus controller (LBC).

Figure 1-9 LBC Block Diagram



- The CDM local bus interface is for reading or writing registers internal to the core, including the configuration space.
- The ELBI is a local bus interface for reading or writing application-owned register space.

The LBC is a module that switches the traffic between two input interfaces and two output interfaces. It allows a received request TLP or application logic to access the configuration registers that are built into the

DWC PCIe core or application-specific registers external to the core. The two inputs are external data bus interface (external DBI) and RADM TARGET0 interface (over which the TLP requests from RADM are received). The two output interfaces are CDM local bus interface (CLBI) and the external local bus interface (ELBI). The LBC also generates a completion which is a result of accessing ELBI or CDM registers.

SW only: The Switch decision of an inbound Config RD/WR request to either registers in CDM or application registers external to the core is determined by filters inside the RADM.

SW only: For Configuration Requests targeted to downstream Switch ports, the upstream instance of the core passes the Configuration Request to the application on RTRGT1. The Switch application logic must respond to the Configuration Request by executing a transaction on the DBI of the target downstream port, then generating the Completion and presenting it on one of the upstream port's XALI interfaces.

1.6.1 LBC Switching

The parameter `CONFIG_LIMIT must be calculated by the application once it gets all of the configuration space laid out. If a configuration request received by the core has an address above this limit, then this configuration request will be directed to the ELBI. This allows the customer to design extended configuration registers that are beyond what the DWC PCIe core natively supports.



Note The Application is required to understand its configuration space and correctly add the extended capabilities. The CONFIG_LIMIT parameter should be set above all configuration registers that the DWC PCIe core is programmed to support.

All inbound memory RD/WR requests are switched to the ELBI interface except when Port Logic (PL) registers of CDM are part of the memory mapped BAR range. Applications can select ENABLE_MEM_MAP_PL_REG to map the PL registers to address 0x700-0x8FF of BAR0 of device function 0.

The DBI RD/WR requests can access either CDM registers or application registers external to the core. This is controlled by bit0 of the DBI address signal. Setting bit0 of the DBI address equal to 1'b0 accesses an internal (CDM) register. Setting bit0 to 1'b1 accesses registers on the ELBI bus.

There are additional signals defined when DBI_MULTI_FUNC_BAR_EN is set. These optional signals allow the DBI interface to direct a DBI RD/WR to a particular BAR#, Function# or ROM by asserting the appropriate enables. This optional capability is needed for multiple BAR and multiple function enabled devices.

1.7 Message Generation (MSG_GEN)

1.7.1 Message Generation (MSG_GEN): DM

In EP mode, the MSG_GEN module generates the following Messages and transmits them to the upstream component (Root Port or downstream Port of Switch):

- ❖ Power Management Messages:
 - ◆ PM_PME
 - ◆ PME_TO_Ack
- ❖ Error Messages:
 - ◆ COR_ERR
 - ◆ ERR_NONFATAL
 - ◆ ERR_FATAL
- ❖ INTx Messages:
 - ◆ Assert_INTA
 - ◆ Assert_INTB
 - ◆ Assert_INTC
 - ◆ Assert_INTD
 - ◆ Deassert_INTA
 - ◆ Deassert_INTB
 - ◆ Deassert_INTC
 - ◆ Deassert_INTD
- ❖ Vendor-defined Messages requested by application logic on the VMI

In RC mode, the MSG_GEN module generates the following Messages and transmits them to the downstream component (Endpoint or upstream Port of Switch):

- ❖ Power Management Messages, triggered by PMC:
 - ◆ PM_Active_State_Nak
 - ◆ PME_Turn_Off
- ❖ Unlock Message, triggered by Root Complex application logic:
 - ◆ Unlock
- ❖ Set Power Limit Support Message, triggered by CDM programming:
 - ◆ Set_Slot_Power_Limit
- ❖ Vendor-defined Messages requested by application logic on the VMI

The MSG_GEN module contains a Message arbiter, which provides arbitration for all Message requests from different functions for a multi-function Endpoint application.

1.7.2 Message Generation (MSG_GEN): RC

The MSG_GEN module generates the following Messages and transmits them to the downstream component (Endpoint or upstream Port of Switch):

- ❖ Power Management Messages, triggered by PMC:
 - ◆ PM_Active_State_Nak
 - ◆ PME_Turn_Off
- ❖ Unlock Message, triggered by Root Complex application logic:
 - ◆ Unlock
- ❖ Set Power Limit Support Message, triggered by CDM programming:
 - ◆ Set_Slot_Power_Limit
- ❖ Vendor-defined Messages requested by application logic on the VMI

1.7.3 Message Generation (MSG_GEN): EP

The MSG_GEN module generates the following Messages and transmits them to the upstream component (Root Port or downstream Port of Switch):

- ❖ Power Management Messages:
 - ◆ PM_PME
 - ◆ PME_TO_Ack
- ❖ Error Messages:
 - ◆ COR_ERR
 - ◆ ERR_NONFATAL
 - ◆ ERR_FATAL
- ❖ INTx Messages:
 - ◆ Assert_INTA
 - ◆ Assert_INTB
 - ◆ Assert_INTC
 - ◆ Assert_INTD
 - ◆ Deassert_INTA
 - ◆ Deassert_INTB
 - ◆ Deassert_INTC
 - ◆ Deassert_INTD
- ❖ Vendor-defined Messages requested by application logic on the VMI

The MSG_GEN module contains a Message arbiter, which provides arbitration for all Message requests from different functions for a multi-function Endpoint application.



1.7.4 Message Generation (MSG_GEN): SW

In upstream ports of the Switch, the MSG_GEN module generates the following Messages and transmits them to the upstream component (Root Port or downstream Port of Switch):

- ❖ Power Management Messages:
 - ◆ PM_PME
 - ◆ PME_TO_Ack
- ❖ Error Messages:
 - ◆ COR_ERR
 - ◆ ERR_NONFATAL
 - ◆ ERR_FATAL
- ❖ INTx Messages:
 - ◆ Assert_INTA
 - ◆ Assert_INTB
 - ◆ Assert_INTC
 - ◆ Assert_INTD
 - ◆ Deassert_INTA
 - ◆ Deassert_INTB
 - ◆ Deassert_INTC
 - ◆ Deassert_INTD
- ❖ Vendor-defined Messages requested by application logic on the VMI

In downstream ports of the Switch, the MSG_GEN module generates the following Messages and transmits them to the downstream component (Endpoint or upstream Port of Switch):

- ❖ Power Management Messages, triggered by PMC:
 - ◆ PM_Active_State_Nak
 - ◆ PME_Turn_Off
- ❖ Unlock Message, triggered by Root Complex application logic:
 - ◆ Unlock
- ❖ Set Power Limit Support Message, triggered by CDM programming:
 - ◆ Set_Slot_Power_Limit
- ❖ Vendor-defined Messages requested by application logic on the VMI

The MSG_GEN module contains a Message arbiter, which provides arbitration for all Message requests from different functions for a multi-function Endpoint application.



1.8 Hot Plug Control (hotplug_ctrl) Module: DM / RC / SW

In RC mode and Switch devices, the Hot-Plug logic supports generation of Hot-Plug interrupts on the following Hot-Plug events:

- ❖ Power Fault Detected
- ❖ MRL Sensor Changed
- ❖ Presence Detect Changed
- ❖ Command Completed
- ❖ Attention Button Pressed
- ❖ Electromechanical Interlock Status Changed
- ❖ Data Link Layer State Changed

When MSI or MSI-X mode is enabled, the core notifies the application of Hot-Plug events using the `hp_msi` output. When INTx interrupt mode is enabled, the core notifies the application of Hot-Plug events using the `hp_int` output.

If PME is enabled, the The Hot-Plug logic generates a Hot-Plug wake-up signal on `hp_pme`, triggered by the above Hot-Plug events. The RC Core does not check if the PM state is D1, D2, or D3_{hot}. It is up to the application to check the value on `pm_dstate` to make sure the device is in D1, D2, or D3_{hot}.



Note A command completion event does not trigger a wake event, even if the system is in a sleep state or if the device is in device state D1, D2, or D3_{hot}.

2

Core Operations

2.1 Initialization

2.1.1 Initialization: DM / RC / EP

Immediately after reset, the DM core goes into either EP mode, RC mode depending on the state of the device_type input. The internal configuration registers in the CDM assume their default reset values as listed in the following sections:

- ❖ “PCIe Registers (DM in EP Mode / EP)” on page 317: PCI Express configuration space for EP Mode
- ❖ “PCIe Registers (DM in RC Mode / RC / SW)” on page 435: PCI Express configuration space for RC Mode
- ❖ “PCIe Registers: Port Logic” on page 504: Vendor-specific registers common to both EP Mode and RC Mode

The application must keep the app_ltssm_enable signal deasserted after reset until the application is ready to establish a Link and start receiving and transmitting TLPs. If the application needs to update configuration registers in the CDM as part of the initialization process, then the application must keep app_ltssm_enable deasserted until it has programmed all the necessary configuration registers through the DBI.

After initializing the necessary configuration registers, the application can assert app_ltssm_enable to allow the LTSSM to begin Link establishment. The LTSSM begins Link negotiation after the core_rst_n and phy_mac_phystatus signals are deasserted and app_ltssm_enable is asserted.

2.1.2 Initialization: SW

Immediately after reset, the SW core goes into either upstream or downstream port of the Switch depending on the state of the device_type input. The internal configuration registers in the CDM assume their default reset values as listed in the following sections:

- ❖ “PCIe Registers: Port Logic” on page 504: PCI Express configuration space for downstream ports
- ❖ “PCIe Registers: Port Logic” on page 504: Vendor-specific registers common to both upstream and downstream ports

The application must keep the app_ltssm_enable signal deasserted after reset until the application is ready to establish a Link and start receiving and transmitting TLPs. If the application needs to update configuration registers in the CDM as part of the initialization process, then the application must keep app_ltssm_enable deasserted until it has programmed all the necessary configuration registers through the DBI.

After initializing the necessary configuration registers, the application can assert app_ltssm_enable to allow the LTSSM to begin Link establishment. The LTSSM begins Link negotiation after the core_rst_n and phy_mac_phystatus signals are deasserted and app_ltssm_enable is asserted.



2.2 Link Establishment

The core and a PCI Express compliant PHY combine to provide a complete solution for setting up and maintaining a compliant PCI Express Link. The core implements the LTSSM function according to the *PCI Express 2.0 specification*.

In general, the process for establishing a Link is as follows:

1. Upon power-up (or directly out of reset), it is assumed that the power supply becomes stable and the ASIC/SoC and SerDes PLLs reach frequency lock before the devices attempt to establish a valid Link. Once in a valid state, the SerDes either communicates a ready status to the core or simply begins transmitting and receiving valid data.
2. Per the *PCI Express 2.0 specification*, once bit and symbol synchronization are complete, the core initiates the following sequence to establish a Link (assuming a valid and properly functioning Link partner):
 - a. Receiver detection on available Lanes for the Port.
 - b. Exchange of Training Sequences to determine Link configuration (for example, Link speed, number of Lanes, and order).
 - c. Once both partners reach a valid negotiated state, the Link state is set up and the LTSSM is in L0.
3. Once Link up is achieved, the data link modules take over to manage the Link and initialize Flow Control.
4. After Flow Control initialization is complete, the data link modules signal the transaction layer modules that the link is ready to allow transmission/reception of TLP traffic.
5. During normal operation, the LTSSM and data link modules continue to manage the underlying Link integrity while data traffic is communicated across the PCI Express Link.



Note The power management implementation also affects Link establishment.

2.3 Basic TLP Processing

The following sections describe the basic flow of transmit and receive TLPs through the core.

2.3.1 Basic TLP Processing: Transmit (DM / RC / EP)

Generally, all types of transmit TLPs (Posted, Non-Posted, and Completion) generated by the application travel through the core in the following flow:

1. The application presents a transaction transmission request with header information and payload (if applicable) on one of the transmit client interfaces (for example, XALI0).
2. The XADM forms the transaction into a TLP and checks the TLP against the current Flow Control credit availability. If the TLP passes the Flow Control checks and wins the arbitration with TLPs from the other the client interfaces, then the TLP goes to the CXPL.
3. The XTLH module inserts an ECRC (if applicable) and snoops/stores the necessary TLP information for Completion lookup (for Non-Posted requests only).
4. The XDLH inserts the Sequence Number and LCRC into the TLP and the retry buffer stores the TLP.
5. The XMLH inserts start and end delimiters and performs data scrambling.
6. The XMLH presents the packet to the PHY through the PIPE interface.
7. The PHY receives the packet, performs 8b10b encoding, and serialization, then sends the packet for transmission on the Link.



Note The core does not check for TLP errors; it sends the TLP as presented on the XALI interface.

2.3.2 Basic TLP Processing: Transmit (SW)

There are three basic classes of TLPs that the core transmits over the Link in a PCI Express Switch application:

- ❖ TLPs that are being forwarded from one Switch Port to another Switch Port. In this case, the Switch application logic stores the TLP received on one Port until it is ready to transmit the TLP upstream or downstream through another Port on the Switch. The application requests the core to transmit a TLP by asserting the request on one of the transmit client interfaces.
- ❖ Messages requested by the Switch application logic:
 - ◆ Legacy Interrupt Messages, which the core generates in response to interrupt requests on the sys_int and dp_intx pins (see “[Legacy PCI INTx Support](#)” on page 83)
 - ◆ MSI(-X) Messages, which the core generates in response to requests on the MSI(-X) interfaces (see “[Message Signaled Interrupt \(MSI\) Support](#)” on page 83 and “[Message Signaled Interrupt \(MSI\) Support](#)” on page 83)
 - ◆ Vendor-defined Messages, which the core generates in response to requests on the VMI (see “[Vendor Message Interface \(VMI\)](#)” on page 195)
- ❖ Internally generated TLPs, which the core generates automatically. Internally generated TLPs include:
 - ◆ Internally generated Messages
 - ◆ Completions for Configuration Requests targeted to the core
 - ◆ Completions for Memory or I/O Requests that are targeted to the Switch application. In this case data transfer between the core and the application occurs on the ELBI; the core automatically generates the Completion.

Generally, all types of transmit TLPs (Posted, Non-Posted, and Completion) generated by the application travel through the core in the following flow:

1. The application presents a transaction transmission request with header information and payload (if applicable) on one of the transmit client interfaces (for example, XALI0).
2. The XADM forms the transaction into a TLP and checks the TLP against the current Flow Control credit availability. If the TLP passes the Flow Control checks and wins the arbitration with TLPs from the other the client interfaces, then the TLP goes to the CXPL.
3. The XADM directs the CXPL to not insert ECRC because the ECRC (if present) in the pass-through TLP must be preserved.
4. In the CXPL, the XTLH module passes the TLP to the XDLH, including the ECRC if ECRC is present.
5. The XDLH inserts the Sequence Number and LCRC into the TLP and the retry buffer stores the TLP.
6. The XMLH inserts start and end delimiters and performs data scrambling.
7. The XMLH presents the packet to the PHY through the PIPE interface.
8. The PHY receives the packet, performs 8b10b encoding, and serialization, then sends the packet for transmission on the Link.



The core does not check for TLP errors; it sends the TLP as presented on the XALI interface.

2.3.3 Basic TLP Processing: Receive (DM / RC / EP)

Generally, received transactions travel through the core in the following flow:

1. The PHY receives a stream of bits and aligns/forms them into 10-bit symbols.
2. The PHY decodes the 10b stream into an 8b stream, crosses the clock domain from RX to TX, and presents the stream to the PIPE.
3. The RMLH deskews and descrambles the 8b data across all Lanes before extracting packets.
4. The RDLH strips off the LCRC and Sequence Number.
5. The RTLH strips off the ECRC (if applicable), checks for a malformed TLP, and forms a transaction across the RTLI interface to the RADM.
6. The RADM filters the transaction based on the transaction type (Posted, Non-Posted, or Completion) and the rules described in “[Receive Filtering](#)” on page [60](#).
7. Filtered transactions are sent to RADM queues.
8. Transactions residing in the RADM queues are presented to the application or locally handled by the LBC module, depending upon the filter result.

2.3.4 Basic TLP Processing: Receive (SW)

TLPs that the core receives over the Link in a Switch application fall into the following general classes:

- ❖ TLPs that are to be routed through the Switch to another Port of the Switch. The core transfers this class of TLP to the application through the RTRGT1 interface. This category includes type 1 Configuration Requests that are received by an upstream Switch Port and must be sent downstream.
- ❖ Configuration Requests that target the core. The core processes this class of TLP internally and automatically generates the required Completion.
- ❖ Memory or I/O Requests that target the Switch application logic. The core transfers this class of Request to the application through the ELBI and automatically generates the required Completion. Memory and I/O Requests targeted to the Switch application logic are limited to single-DWORD accesses.

Generally, received transactions travel through the core in the following flow:

1. The PHY receives a stream of bits and aligns/forms them into 10-bit symbols.
2. The PHY decodes the 10b stream into an 8b stream, crosses the clock domain from RX to TX, and presents the stream to the PIPE.
3. The RMLH deskews and descrambles the 8b data across all Lanes before extracting packets.
4. The RDLH strips off the LCRC and Sequence Number.
5. The RTLH strips off the ECRC (if applicable), checks for a malformed TLP, and forms a transaction across the RTLI interface to the RADM.
6. The RADM filters the transaction based on the transaction type (Posted, Non-Posted, or Completion) and the rules described in “[Receive Filtering](#)” on page [60](#).
7. Filtered transactions are sent to RADM queues.
8. Transactions residing in the RADM queues are presented to the application or locally handled by the LBC module, depending upon the filter result.

2.3.5 Transmit TLP Arbitration

The transmit arbitration mechanisms supported by the core are as follows:

- ❖ Client-based round robin arbitration
- ❖ Strict priority client-based arbitration
- ❖ VC-based priority arbitration

The transmit arbitration algorithm can be configured through the Transmit Arbitration Method configuration option ('CX_XADM_ARB_MODE) in core consultant.

Regardless of the transmit arbitration method selected, Messages (both internally-generated and Messages requested through the VMI) always have the highest priority, followed by internally-generated Completions. The priority order for all transmitted TLPs is:

1. Internally generated Messages
2. Internally-generated Completions
3. Transmit TLPs from Client0, Client1, and Client2 according to the selected arbitration method

2.3.5.1 Client-Based Arbitration

When you configure the core to use client-based arbitration ('CX_XADM_ARB_MODE = 1), the XADM uses round-robin arbitration between the two or (optionally three) transmit client interfaces.

2.3.5.2 Strict Priority Arbitration

When you configure the core to use strict priority arbitration ('CX_XADM_ARB_MODE = 2), the XADM uses strict priority arbitration between the two or (optionally) three transmit client interfaces. Client0 has the lowest priority, then Client1, then Client2 (if implemented).

2.3.5.3 VC-Based Arbitration

The VC-based arbitration method uses two different arbitration methods for the two groups of VCs:

- ❖ Strict priority for the high-priority VC (HPVC) group
- ❖ Round robin or weighted round robin for the low-priority VC (LPVC) group

The value of the Low Priority Extended VC Count field of Port VC Capability Register 1 determines the number of VCs (in addition to VC0) that are in the LPVC group. All other VCs are in the HPVC group. See *DesignWare Cores PCI Express Reference Manual*, “Virtual Channel Capability Registers”.

Between the two groups of VCs, arbitration is as follows:

1. The HPVC group is always the highest priority. Within the HPVC group, priority order is by VC ID. The highest VC ID has the highest priority.
2. The LPVC group is of lower priority than the HPVC group. Within the LPVC group, priority is determined by round robin or WRR arbitration as described in the following sections. Ties within the LPVC group are resolved by client-based round robin arbitration.

2.3.5.3.1 Setting Up VC-Based WRR

The WRR arbitration scheme uses the programmed values for each LPVC's weight (programmed by VC ID) to determine the percentage of the time that each VC will have the highest priority within the LPVC group. The factors that influence WRR arbitration are:



- ❖ Number of VCs in the LPVC group, as programmed in the Low Priority Extended VC Count field of Port VC Capability Register 1. See *DesignWare Cores PCI Express Reference Manual*, “[Virtual Channel Capability Registers](#)”.
- ❖ The number of phases in the selected WRR arbitration scheme, as determined by:
 - ◆ The VC Arbitration Capability field in Port VC Capability Register 2 (indicates the scheme(s) that the core is configured to support: 16, 32, 64, or 128-phase). See *DesignWare Cores PCI Express Reference Manual*, Section, “[Virtual Channel Capability Registers](#)”.
 - ◆ The VC Arbitration Select field in Port VC Control Register (selects the arbitration scheme). See *DesignWare Cores PCI Express Reference Manual*, “[Virtual Channel Capability Registers](#)”.
- ❖ The weight assigned to each VC in the LPVC group, as programmed in VC Transmit Arbitration Register 1 and VC Transmit Arbitration Register 2. See *DesignWare Cores PCI Express Reference Manual*, Section 4.3.4, “Port Logic Registers: Transmit Arbitration.” The sum of all the programmed weight values must equal the number of phases in the selected WRR arbitration scheme. For example, for 64-phase arbitration, the total of the weights assigned to all the VCs in the LPVC group must be 64.

VC-based round robin arbitration is a special case of VC-based WRR in which the weights are equal for all VCs in the LPVC group.

2.3.6 Transmit Retry

There is a Retry Buffer (RB) in the core that stores a copy of each transmitted TLP until an Ack is received. The RB consists of two buffers: retry buffer and start-of-TLP (SOT) buffer.



Note The Retry Buffer does not function as a transmit queue. The core transmits TLPs immediately after they pass arbitration. The copy in the Retry Buffer is only sent in the event that the TLP must be retransmitted.

The retry buffer is implemented with a single port RAM. The depth of the retry buffer is selected during hardware configuration either by enabling automatic buffer sizing or by setting the depth explicitly. The retry buffer width is set automatically (data bus width plus extra control bits).

The selected retry buffer size determines the size of the SOT buffer. The SOT buffer stores the starting address of each unacknowledged TLP stored in the retry buffer. The SOT buffer is implemented with a single port RAM and is indexed by the Sequence Number of the TLP whose starting address is being stored or retrieved.

The minimum depth of the SOT buffer is also a user configuration option. The selected size must allow the retry buffer to store the maximum number of shortest TLPs (3 DWORDs).

When a Nak is received or the replay timer times out, a replay is initiated.

A replay is terminated by two conditions:

- ❖ When the replay of all TLPs in the retry buffer is finished, or
- ❖ An Ack DLLP is received that acknowledges all TLPs in the retry buffer

The replay timer tracks the TLP replay time. It stays at 0 when every TLP has received an Ack and starts to count when a TLP is transmitted and the LTSSM is not in the training state. The replay timer is reset to 0 when an Ack or Nak is received that acknowledges a TLP that is in the retry buffer.

If you enable parity checking for the retry buffer (by setting `CX_PAR_MODE to 8, 16, or 32), and the core detects a parity error when reading data out of the retry buffer, the core nullifies the packet that contained the parity error when replaying the packet.

2.3.7 Transmit DLLP Priorities

The order of priority to transmit pending DLLPs is:

1. High-priority DLLPs
2. TLPs
3. Low-priority DLLPs

2.3.8 Receive Filtering

The core contains a filter module that is responsible for the following tasks:

- ❖ Determine the status of a received TLP using filtering rules.
- ❖ Determine the destination interfaces of a received TLP based on the status from applying the filter rules.
- ❖ Signal the application for the status of the received TLP by driving signals such as DLLP abort, TLP abort and ECRC error.
- ❖ Report Errors to AER registers based on filter results. If more than one types of error detected, section 6.2.3.2.3 Error Pollution of the *PCI Express 2.0 specification* is followed.

The core filters and routes received TLPs according to a set of rules determined by the TLP type based on the *PCI Express Base 2.0 Specification* and user-configurable filtering options. The filtering rules for a received TLP are affected by the configuration parameters (compile-time options), I/O signals (run-time options), and register values (run-time options). The application can mask some of the filtering and error handling rules by setting the corresponding bits in Symbol Timer Register and Filter Mask Register 1 and Filter Mask Register 2. Refer to “[Symbol Timer Register and Filter Mask Register 1](#)” on page 517 and “[Filter Mask Register 2](#)” on page 520 for specific bit assignments.

The filter in PCIe core is device specific. EP device contains an EP filter which the EP filtering rules are applicable. RC device contains an RC filter which the RC filtering rules are applicable. DM device contains both EP and RC filter which only one of the filter is active based on the mode that the device is running under.

There are three categories of the filtering rules designed in PCIe core. One category contains the rules that are applicable for all TLP received. One category contains the rules that are depended on the type of the TLP based on PCIe specification. One category contains the rules that are not from the PCIe spec but requested by specific applications.

2.3.8.1 Filtering Rules Applicable for all TLPs Received

The following general rules apply to all incoming TLPs:

- ❖ The core discards all incoming TLPs that have an invalid Type field. This TLP is treated as a TLP abort.
- ❖ A TLP with the poison bit set is considered an Unsupported Request (UR) only when the UR poison rule mask bit is not set. Applications can control the end result of a poisoned TLP filter through the corresponding filter mask bit. If the filter mask bit is not set, all locally terminated TLPs with poison bit set will be discarded.
- ❖ A locally terminated TLP with ECRC error detected is discarded in store-and-forward mode and an ECRC error reported only when the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.
- ❖ Filter rules have no affect on received TLP when DLLP abort signal is asserted.
- ❖ If a completion of a non-posted request is not received within a completion timeout period, this request will be treated as a completion timeout, and a non-advisory error will be reported.
- ❖ For messages to be accepted and decoded, the incoming Message must be one of the valid Message types with the correct payload length based on PCIe spec.2.0. Valid Messages will be decoded and passed onto the SII interface as necessary.

- ❖ By default, all Configuration Requests that pass filtering go to RTRGT0 for configuration register access. However, the application can configure the core to direct certain configuration TLPs to the RTRGT1 interface. The application is responsible for generating Completions for Configuration Requests that are routed to RTRGT1.

The following sections describe the filtering and routing details for various types of Request and Completion TLPs.

2.3.8.2 Filtering Rules Based on TLP Type Defined in PCIe Specification (EP Mode)

PCIe TLPs are categorized as Requests and Completions. [Table 2-1](#) describes the filtering rules for Request and Completion TLPs and the results of the DM core's filter in EP Mode.

If a received TLP passes all of the filter rules for Request and Completion TLPs, then it is considered to have no errors, and the TLP will be routed to the destination that is configured. Details on routing are provided in [“Routing of TLPs to Destination Interfaces \(EP Mode\)”, “Routing Rules Defined for Request TLPs \(EP Mode\)”](#) and [“Routing Rules Defined for Completion TLPs \(EP Mode\)”](#).

Notation of filter results:

UR = Unsupported request
 CA = Completion abort
 CRS = Completion retry request
 SU = Successful
 UC = Unexpected completion
 MLF = Malformed
 N/A = Filtering rule does not apply to TLP type

Table 2-1 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs (EP Mode)

Filtering Rule	TLP Type					
	MRd and IORd	MWr and IOWr	CFGWr and CFGRd	Message	CPL with UR/CA/CRS status	CPL with SU status
PowerState is not in D0.	UR	UR	SU	SU	UC	UC
Address is not within any configured Memory BAR or IO BAR if it is an IO request.	UR	UR	N/A	N/A	N/A	N/A
TLP header poison bit is set and the filter mask CX_FLT_MASK_UR_POIS bit is not set.	UR	UR	UR	UR	PASS	PASS
Address within a BAR that is configured to TRGT0 and TLP DW length > 1.	CA	CA	N/A	N/A	N/A	N/A
MRd with lock and filter mask CX_FLT_MASK_LOCKED_RD_AS_UR bit is not set.	UR	N/A	N/A	N/A	N/A	N/A

Table 2-1 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs (EP Mode)

Filtering Rule	TLP Type					
	MRd and IORd	MWr and IOWr	CFGWr and CFGRd	Message	CPL with UR/CA/C RS status	CPL with SU status
The function number of a completer ID within a CFG request does not match the function number of the receiver device and the filter mask CX_FLT_MASK_UR_FUNC_MISMATCH bit is not set.	N/A	N/A	UR	N/A	N/A	N/A
Configure type1 TLP request and the filter mask CX_FLT_MASK_CFG_TYPE1_REQ_ASUR is not set.	N/A	N/A	UR	N/A	N/A	N/A
Application requests the core filter to return CRS by asserting signal app_req_retry_en.	N/A	N/A	CRS	N/A	N/A	N/A
Not Valid Message for EP device	N/A	N/A	N/A	UR	N/A	N/A
Illegal payload length of a message.	N/A	N/A	N/A	UR	N/A	N/A
Vendor MSG Type0 with filter mask CX_FLT_MASK_VENMSG0_DROP bit not set.	N/A	N/A	N/A	UR	N/A	N/A
Vendor MSG Type1 with r[2:0] to 3'b010 and {Bus#, Dev#, Func#} mis-match.	N/A	N/A	N/A	UR	N/A	N/A
TLP with ECRC error detected	CA	CA	CA	N/A	N/A	N/A
Requester ID mis-match	N/A	N/A	N/A	N/A	UC	UC
Requester TAG mis-match	N/A	N/A	N/A	N/A	UC	UC
TAG error (non-pad zero for reserved TAG bits)	N/A	N/A	N/A	N/A	UC	UC
Byte Count Mismatch (PCIe Gen2)	N/A	N/A	N/A	N/A	UC	UC
Completion received with status of UR.	N/A	N/A	N/A	N/A	UR	N/A
Completion received with status of CA.	N/A	N/A	N/A	N/A	CA	N/A
Completion received with status of CRS	N/A	N/A	N/A	N/A	CRS	N/A
Completion received with CRS status and Completion is not a pending configuration request	N/A	N/A	N/A	N/A	MLF	N/A

2.3.8.3 Routing of TLPs to Destination Interfaces (EP Mode)

The possible destinations of a posted or non-posted Request TLP are TRGT1 interface, TRGT0 interface and Core Discard. The possible destinations of a Completion TLP are RADM CPL interface, RADM Bypass interface, RADM TRGT1 interface, and Core Discard.

Because the core supports three types of queue architecture (single queue, multiple queue, segmented buffer queue architecture) and three buffer modes (bypass, cut-through, store-forward mode), a configuration of the core receive queue structure will affect the destination of a received TLP.

In general, a TLP type that is configured as bypass will be sent to either RADM Bypass or RADM CPL if it is a completion. A TLP type that is configured as a cut-through or store-forward will be sent to TRGT1 interface.

2.3.8.3.1 Routing Rules Defined for Request TLPs (EP Mode)

[Table 2-2](#) shows the applicability of routing rules for Request TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

Notation of routing results:

yes = destination is as specified in rule when conditions of rule are met

no = destination is not as specified in rule even when conditions of rule are met

N/A = routing rule has no affect because it does not apply to TLP type

Table 2-2 Application of Routing Rules Applied to Request TLPs: EP Mode

Routing Rule	TLP Type						Vendor Message Type0	Vendor Message Type1
	MRd & IORd & IOWr	MWr	CFG Rd & WR	All messages except Vendor Message				
The MEM_FUNC#_BAR#_TARGET_MAP parameter determines the destination when requests are filtered with successful status.	yes	yes	no	no	no	no	no	no
The DEFAULT_TARGET parameter determines the destination when requests are filtered with UR/CA/CRS status.	yes	no	yes	no	no	no	no	no
Core Drop is the destination when requests are filtered with UR/CA/CRS status	no	yes	no	yes	yes	yes	yes	yes
The TARGET_ABOVE_CONFIG parameter determines the destination when a configuration request is filtered with successful status <u>and</u> the configurations' register address is above the CONFIG_LIMIT parameter set.	N/A	N/A	yes	no	no	no	no	no

Table 2-2 Application of Routing Rules Applied to Request TLPs: EP Mode

Routing Rule	TLP Type			All messages except Vendor Message	Vendor Message Type0	Vendor Message Type1
	MRd & IORd & IOWr	MWr	CFG Rd & WR			
TRGT0 interface is the destination when a configuration request is filtered with successful status <u>and</u> the configurations' register address is below the CONFIG_LIMIT parameter set.	N/A	N/A	yes	no	no	no
The DEFAULT_TARGET parameter determines the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	yes	no	yes	no	no	no
Core Drop is the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	no	yes	no	yes	yes	yes
Core Drop is the destination when the filter mask CX_FLT_MASK_MSG_DROP bit is not set <u>and</u> the messages are filtered with successful status.	N/A	N/A	N/A	yes	no	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG0_DROP bit is not set <u>and</u> the Vendor0 messages are filtered with successful status.	N/A	N/A	N/A	no	yes	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG1_DROP bit is not set <u>and</u> the Vendor1 messages are filtered with successful status.	N/A	N/A	N/A	no	no	yes
TRGT1 interface is the destination when the messages are filtered with successful status.	N/A	N/A	N/A	yes	yes	yes

2.3.8.3.2 Routing Rules Defined for Completion TLPs (EP Mode)

Table 2-3 shows the applicability of routing rules for Completion TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

Notation of routing results:

yes = destination is as specified in rule when conditions of rule are met
 no = destination is not as specified in rule even when conditions of rule are met
 N/A = routing rule has no affect because it does not apply to TLP type

Table 2-3 Application of Routing Rules based on Completion TLPs: EP Mode

Routing Rule	TLP Type	Unexpected completion or completion with DLLP abort signal asserted or ECRC error asserted	Completion with UR or CA or CRS status	Completion with successful status
RADM CPL interface is the destination when completion is in either a single queue <u>or</u> multiple queue architecture.	yes		yes	yes
Core Drop is the destination when completion is in store-forward buffer mode.	yes		no	no
RADM Bypass interface is the destination when completion is in by-pass mode of a segment buffer queue architecture.	yes		yes	yes
TRGT1 interface is the destination when completion is in cut-through mode of a segment buffer queue architecture.	yes		yes	yes
TRGT1 interface is the destination when completion is in store-forward mode of segment buffer queue architecture.	no		yes	yes

2.3.8.4 Filtering Rules Based on TLP Type Defined in PCIe Specification (RC Mode)

PCIe TLPs are categorized as Requests and Completions. [Table 2-4](#) describes the filtering rules for Request and Completion TLPs and the results of the DM core's filter in RC Mode.

If a received TLP passes all of the filter rules for Request and Completion TLPs, then it is considered to have no errors, and the TLP will be routed to the destination that is configured. Details on routing are provided in [“Routing of TLPs to Destination Interfaces \(RC Mode\)”, “Routing Rules Defined for Request TLPs: RC Mode”](#) and [“Routing Rules Defined for Completion TLPs \(RC Mode\)”](#).

Notation of filter results:

UR = Unsupported request
 CA = Completion abort
 CRS = Completion retry request
 SU = Successful
 UC = Unexpected completion
 MLF = Malformed
 N/A = Filtering rule does not apply to TLP type

Table 2-4 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs: RC Mode

Filtering Rule	TLP Type					CPL with UR/CA status	CPL with CRS status	CPL with SU status
	MRd	MWr	CFG Rd/Wr	IO Rd/WR	Message			
Address does not satisfy any of the following conditions: 1. Within any configured Memory BAR 2. Outside of the memory range and prefetch memory range 3. the filter mask <code>CX_FLT_MASK_UR_OUTSIDE_BAR</code> bit is set, which indicates a special application requirement	UR	UR	N/A	UR	N/A	N/A	N/A	N/A
TLP header poison bit is set and the filter mask <code>CX_FLT_MASK_UR_POIS</code> bit is not set.	UR	UR	UR	UR	UR	N/A	N/A	N/A
MRd with lock and filter mask <code>CX_FLT_MASK_LOCKED_RD_A_S_UR</code> bit is set, which indicates that customer prefer to filter out the memory RD with lock.	UR	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CFG Request received and the filter mask <code>CX_FLT_MASK_RC_CFG_DISC</code> ARD is not set	N/A	N/A	UR	N/A	N/A	N/A	N/A	N/A
IO Request received and the filter mask <code>CX_FLT_MASK_RC_IO_DISCA</code> RD is not set	N/A	N/A	N/A	UR	N/A	N/A	N/A	N/A
Vendor MSG Type0 with filter mask <code>CX_FLT_MASK_VENMSG0_DR</code> OP bit not set.	N/A	N/A	N/A	N/A	UR	N/A	N/A	N/A
Not Valid Message for RC device	N/A	N/A	N/A	N/A	UR	N/A	N/A	N/A
TLP with ECRC error detected	CA	CA	CA	CA	N/A	N/A	N/A	N/A
Requester ID mis-match	N/A	N/A	N/A	N/A	N/A	UC	N/A	UC
Requester TAG mis-match	N/A	N/A	N/A	N/A	N/A	UC	N/A	UC

Table 2-4 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs: RC Mode

Filtering Rule	TLP Type					CPL with UR/CA status	CPL with CRS status	CPL with SU status
	MRd	MWr	CFG Rd/Wr	IO Rd/WR	Message			
TAG error (non-pad zero for reserved TAG bits)	N/A	N/A	N/A	N/A	N/A	UC	N/A	UC
Byte Count Mismatch (PCIe Gen2)	N/A	N/A	N/A	N/A	N/A	UC	N/A	UC
Completion received with status of UR.	N/A	N/A	N/A	N/A	N/A	UR	N/A	N/A
Completion received with status of CA.	N/A	N/A	N/A	N/A	N/A	CA	N/A	N/A
Completion received with CRS status and Completion is not a pending configuration request.	N/A	N/A	N/A	N/A	N/A	N/A	MLF	N/A

2.3.8.5 Routing of TLPs to Destination Interfaces (RC Mode)

The possible destinations of a posted or non-posted Request TLP are TRGT1 interface, TRGT0 interface and Core Discard. The possible destinations of a Completion TLP are RADM CPL interface, RADM Bypass interface, RADM TRGT1 interface, and Core Discard.

Because the core supports three types of queue architecture (single queue, multiple queue, segmented buffer queue architecture) and three buffer modes (bypass, cut-through, store-forward mode), a configuration of the core receive queue structure will affect the destination of a received TLP.

In general, a TLP type that is configured as bypass will be sent to either RADM Bypass or RADM CPL if it is a completion. A TLP type that is configured as a cut-through or store-forward will be sent to TRGT1 interface.

2.3.8.5.1 Routing Rules Defined for Request TLPs: RC Mode

Table 2-5 shows the applicability of routing rules for Request TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

Notation of routing results:

yes = destination is as specified in rule when conditions of rule are met

no = destination is not as specified in rule even when conditions of rule are met

N/A = routing rule has no affect because it does not apply to TLP type

Table 2-5 Application of Routing Rules Applied to Request TLPs: RC Mode

Routing Rule	TLP Type			CFG & IO Rd/WR	All messages except Vendor Message	Vendor Message Type0	Vendor Message Type1
	MRd	MWr					
The MEM_FUNC#_BAR#_TARGET_MAP parameter determines the destination when requests are filtered with successful status and requests are in BAR range.	yes	yes		no	no	no	no
The DEFAULT_TARGET parameter determines the destination when requests are filtered with UR/CA status.	yes	no	yes	yes	no	no	no
Core Drop is the destination when requests are filtered with UR/CA status.	no	yes	no	yes	yes	yes	yes
The DEFAULT_TARGET parameter determines the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	yes	no	yes	no	no	no	no
Core Drop is the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	no	yes	no	yes	yes	yes	yes
Core Drop is the destination when the filter mask CX_FLT_MASK_MSG_DROP bit is not set <u>and</u> the messages are filtered with successful status.	N/A	N/A	N/A	yes	no	no	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG0_DROP bit is not set <u>and</u> the Vendor0 messages are filtered with successful status.	N/A	N/A	N/A	no	yes	yes	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG1_DROP bit is not set <u>and</u> the Vendor1 messages are filtered with successful status.	N/A	N/A	N/A	no	no	no	yes
TRGT1 interface is the destination when the messages are filtered with successful status.	N/A	N/A	N/A	yes	yes	yes	yes
TRGT1 interface is the destination when the Memory Requests are filtered with successful status.	yes	yes	N/A	N/A	N/A	N/A	N/A

2.3.8.5.2 Routing Rules Defined for Completion TLPs (RC Mode)

Table 2-6 shows the applicability of routing rules for Completion TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

Notation of routing results:

yes = destination is as specified in rule when conditions of rule are met

no = destination is not as specified in rule even when conditions of rule are met

N/A = routing rule has no affect because it does not apply to TLP type

Table 2-6 Application of Routing Rules based on Completion TLPs: RC Mode

Routing Rule	TLP Type	Completion with UR or CA or CRS status	Completion with successful status
RADM CPL interface is the destination when completion is in either a single queue <u>or</u> multiple queue architecture.	yes Unexpected completion or completion with DLLP abort signal asserted or ECRC error asserted	yes	yes
Core Drop is the destination when completion is in store-forward buffer mode.	yes	no	no
RADM Bypass interface is the destination when completion is in by-pass mode of a segment buffer queue architecture.	yes	yes	yes
TRGT1 interface is the destination when completion is in cut-through mode of a segment buffer queue architecture.	yes	yes	yes
TRGT1 interface is the destination when completion is in store-forward mode of segment buffer queue architecture.	no	yes	yes

2.3.8.6 Filtering Rules Based on TLP Type Defined in PCIe Specification (SW)

The filtering rules for an incoming Request TLP are affected by the configuration parameters (compile-time options), I/O signals (run-time options), and register values (run-time options).

Completions are not filtered inside switch filter. This is under an assumption that switch will not generate a request locally within a switch. If embedded EP is involved in a switch application, there should be some modifications based on the requirements of the application.

Table 2-7 describes the filtering rules for Request TLPs and the results of the core's filter.

If a received TLP passes all of the filter rules, then it is considered to have no errors, and the TLP will be routed to the destination that is configured. Details on routing are provided in “[Routing of TLPs to Destination Interfaces \(SW\)](#)” and “[Routing Rules Defined for Request TLPs \(SW\)](#)”.

Notation of filter results:

UR = Unsupported request
 CA = Completion abort
 CRS = Completion retry request
 SU = Successful
 UC = Unexpected completion
 MLF = Malformed
 N/A = Filtering rule does not apply to TLP type

Table 2-7 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs: SW

Filtering Rule	TLP Type			
	Mem & IO Rd/Wr	CFG0 Rd/Wr	CFG1 Rd/Wr	Message TLP
PowerState is not in D0.	UR	SU	UR	SU
Address does not meet any of the following conditions: 1. Within any configured BAR of upstream port 2. Outside of the memory range and prefetch memory range of the downstream port. Or, inside of the memory range or prefetch memory range of the upstream port 3. the filter mask CX_FLT_MASK_UR_OUTSIDE_BAR bit is set which indicates a special application requirement	UR	N/A	N/A	N/A
TLP header poison bit is set and the filter mask CX_FLT_MASK_UR_POIS bit is not set for the TLPs that are locally towards switch.	UR	UR	SU	UR
The TLPs that is designated to TRGT0 and TLP DW length is greater than one (Message will never be directed to TRGT0).	CA	CA	SU	N/A
The function number of a completer ID within a CFG type0 request of upstream port does not match the function number of the receiver device and the filter mask CX_FLT_MASK_UR_FUNC_MISMATCH bit is not set.	N/A	UR	SU	N/A
Application requests the core filter to return CRS to CFG type0 request by asserting signal app_req_retry_en at upstream port.	N/A	CRS	SU	N/A
Not Valid Message depended on the SW upstream port or downstream port.	N/A	N/A	N/A	UR
Vendor MSG Type0 with filter mask CX_FLT_MASK_VENMSG0_DROP bit not set.	N/A	N/A	N/A	UR

Table 2-7 Result of Filtering Rules Applied to Request TLPs and Completion (CLP) TLPs: SW

Filtering Rule	TLP Type			
	Mem & IO Rd/Wr	CFG0 Rd/Wr	CFG1 Rd/Wr	Message TLP
The TLPs that is designated to switch and ECRC error detected.	CA	CA	CA	CA

2.3.8.7 Routing of TLPs to Destination Interfaces (SW)

The possible destinations of a posted or non-posted Request TLP are TRGT1 interface, TRGT0 interface and Core Discard. The possible destinations of a Completion TLP are RADM CPL interface, RADM Bypass interface, RADM TRGT1 interface, and Core Discard.

2.3.8.7.1 Routing Rules Defined for Request TLPs (SW)

Table 2-8 shows the applicability of routing rules for Request TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

Notation of routing results:

yes = destination is as specified in rule when conditions of rule are met

no = destination is not as specified in rule even when conditions of rule are met

N/A = routing rule has no affect because it does not apply to TLP type

Table 2-8 Application of Routing Rules Applied to Request TLPs: SW

Routing Rule	TLP Type				All messages except Vendor Message	Vendor Message Type0	Vendor Message Type1
	MRd & IORd & IOWr	MWr	CFG Rd & WR				
The MEM_FUNC#_BAR#_TARGET_MAP parameter determines the destination when requests are filtered with successful status.	yes	yes	no	no	no	no	no
The DEFAULT_TARGET parameter determines the destination when requests are filtered with UR/CA/CRS status.	yes	no	yes	no	no	no	no
Core Drop is the destination when requests are filtered with UR/CA/CRS status	no	yes	no	yes	yes	yes	yes
The TARGET_ABOVE_CONFIG parameter determines the destination when a configuration request is filtered with successful status <u>and</u> the configurations' register address is above the CONFIG_LIMIT parameter set.	N/A	N/A	yes	no	no	no	no

Table 2-8 Application of Routing Rules Applied to Request TLPs: SW

Routing Rule	TLP Type			All messages except Vendor Message	Vendor Message Type0	Vendor Message Type1
	MRd & IORd & IOWr	MWr	CFG Rd & WR			
TRGT0 interface is the destination when a configuration request is filtered with successful status <u>and</u> the configurations' register address is below the CONFIG_LIMIT parameter set.	N/A	N/A	yes	no	no	no
The DEFAULT_TARGET parameter determines the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	yes	no	yes	no	no	no
Core Drop is the destination when ECRC error is detected <u>and</u> the filter mask CX_FLT_MASK_ECRC_DISCARD bit is not set.	no	yes	no	yes	yes	yes
Core Drop is the destination when the filter mask CX_FLT_MASK_MSG_DROP bit is not set <u>and</u> the messages are filtered with successful status.	N/A	N/A	N/A	yes	no	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG0_DROP bit is not set <u>and</u> the Vendor0 messages are filtered with successful status.	N/A	N/A	N/A	no	yes	no
Core Drop is the destination when the filter mask CX_FLT_MASK_VENMSG1_DROP bit is not set <u>and</u> the Vendor1 messages are filtered with successful status.	N/A	N/A	N/A	no	no	yes
TRGT1 interface is the destination when the messages are filtered with successful status.	N/A	N/A	N/A	yes	yes	yes

2.3.8.8 Filtering Rules Not Defined in PCIe Specification

There are few filtering rules that are designed to provide enhanced filter support for certain applications.

- ❖ Core to handle the received posted or non-posted requests with zero byte length.
When a zero-byte request TLP is received, also called "flush" command, the core can drop the zero-byte request. This is designed to support some applications that can not handle a zero-byte request. Applications can dynamically program a bit in the filter mask CX_FLT_MASK_HANDLE_FLUSH bit to turn on/off this rule. If the core is programmed to handle the flush, it will be the completer to return completion status.
- ❖ Core to detect oversize read request and return UR for the read request.
Some applications may have a buffer limit and are not able to handle lengthy read requests. The core over-size read request detection rule can be turned on when an application can identify a maximum read request size that it can tolerant. This feature is enabled when DWC PCIe AHB or AXI bridge are enabled.

2.3.8.9 Error Indication Signals for Request TLPs Based on Filtering Rules (DM / RC / EP)

- ❖ DLLP-abort asserted as a result of one of two conditions:
 - (1) A data link layer error is detected and a retry from a remote device will occur.
 - (2) An unexpected completion or completion with ECRC error is detected in EP and RC devices. This condition is valid only when the application has configured the core with infinite credits. Because the completion buffer of the core or application has limited resources defined for expected completions, it is necessary to avoid overflowing the completion buffer by unexpected completions. Therefore DLLP abort is asserted to notify the core completion buffer (if completion is in store-forward mode) or application's completion buffer to rewind their buffer pointers when a completion with ECRC error or unexpected completion is detected.
- ❖ TLP abort consists of the malformed TLP detected from lower layer and the unexpected completion or a completion with an ECRC error for EP and RC devices, which notifies the application that there is a bad TLP received. This TLP is not expected to be recovered.
- ❖ ECRC error indicates the ECRC check failure of the TLP. When a TLP has an ECRC error detected, the header information is no longer valid.
- ❖ radm_trgt1_cpl_status indicates the suggested completion status based on the filter rules.

2.3.8.10 Error Indication Signals for Request TLPs Based on Filtering Rules (SW)

- ❖ DLLP-abort asserted as a result of the following conditions:
 - (1) A data link layer error is detected and a retry from a remote device will occur.
- ❖ ECRC error indicates the ECRC check failure of the TLP. When a TLP has an ECRC error detected, the header information is no longer valid.
- ❖ radm_trgt1_cpl_status indicates the suggested completion status based on the filter rules.
- ❖ ECRC is checked and pass through for SW device.

2.3.9 Receive Queuing (DM / RC / EP)

The DM, RC, and EP cores support three configurable queue architectures per VC: single queue, multiple queue, and segmented queue. Each queue architecture supports three buffering modes: bypass mode, cut-through mode, and store-and-forward mode. The buffering mode is selectable for each TLP type: posted, non-posted and completion. The configurability is dependent on the queuing architecture.

The single queue architecture has one header buffer and one data buffer; and the header and data buffers are used as a single FIFO for buffering all posted, non-posted and completion TLP.

The multiple queue architecture has a single header and data buffer per posted, non-posted and completion TLP type.

The segmented queue architecture has one header buffer and one data buffer, but these two buffers are segmented by posted, non-posted and completion TLP type (versus single queue architecture).

For all queuing modes, RAM modules are either instantiated inside the top-level module of the core or connected externally, which is configurable. See [External RAM Interface \(optional\)](#) for the external interface signals. External RAM may be preferred for customers doing RAM layout separately.

2.3.9.1 Queuing Architecture

2.3.9.1.1 Single-Buffer Receive Queue Per VC Configuration

The single-buffer queue architecture is designed for applications that want to have a FIFO buffering for all TLP types. The single queue architecture is the most simple and straight forward buffer architecture. This queue architecture is commonly used for endpoint devices.

In the single-buffer configuration:

- ❖ The Posted and Non-Posted Request cannot be bypassed.
- ❖ The Completion can be configured as is in bypass. The bypassed completion will be routed to the radm_cpl interface. The non-bypassed TLP will be routed to the RTRGT1 interface.
- ❖ The width of each memory is set automatically.
- ❖ The depth of each memory is set at hardware configuration time, and is either automatically calculated or user-specified. The memory depths are the same for all configured VCs.

2.3.9.1.2 Multiple-Buffer Receive Queue per VC Configuration

The multiple-buffer queue architecture is designed for applications that want to have the received TLPs queued per transaction type. Each Transaction type per VC is serviced in received order (like a FIFO). This is queue architecture not commonly used.

In the multiple-buffer configuration:

- ❖ The width of each memory is set automatically and independently.
- ❖ You can select the operating mode (bypass, cut-through, or store-and-forward) independently for each TLP type during hardware configuration.
- ❖ The depths of the Posted, Non-Posted, and Completion queue memories are set at hardware configuration time, either automatically calculated or user-specified. The memory depths for each TLP type are the same for all configured VCs.

2.3.9.1.3 Segmented-Buffer Receive Queue Configuration

The segmented-buffer queue architecture is designed for applications that want to enforce an ordering rule other than FIFO. This is the only queue architecture that can strictly adhere to the ordering rules of the PCI



Express Specification. This queue architecture is relatively large in design. The segmented-buffer configuration uses a single memory module pair (header and data) for all TLP types and all VCs.

In the segmented-buffer configuration:

- ❖ The memory width is set automatically.
- ❖ The memory is divided into segments for Posted, Non-Posted, and Completion queues for each VC. The depth of each segment is set during hardware configuration, and can later be controlled dynamically by writing to the Port Logic registers.
- ❖ The operating mode is selected (bypass, cut-through, or store-and-forward) independently for each TLP type of each VC during hardware configuration. The operating mode (per TLP type and VC) can be controlled dynamically by writing to the Port Logic registers. If a TLP type is configured to be cut-through or store-and-forward, then it will be routed to the RTGT interfaces. If a TLP type is configured to be bypassed, then it will be routed to the radm_bypass interface. Once a Posted Request is configured in bypass mode, the application should not expect to send a Posted-write to the ELBI interface.
- ❖ The number of advertised credits is selected independently for each TLP type and each VC during hardware configuration; and the number can be selected dynamically by writing to the Port Logic registers.
- ❖ The receive queue priority for VCs can be set to either strict priority (higher-numbered VCs have higher priority) or round robin during hardware configuration; and the priority at runtime can be set by writing to the Port Logic registers.
- ❖ The ordering rules for TLP types can be set during hardware configuration and at runtime by writing to the Port Logic registers. The choices are either strict priority (Posted first, Completion second, Non-Posted third) or priority determined according to ordering rules set forth in the *PCI Express 2.0 specification*.

2.3.9.2 Buffering Modes

2.3.9.2.1 Bypass Mode

Queues in bypass mode are completely bypassed.

In the case of aborted TLPs, the core asserts either dllp_abort or tlp_abort to the application. The application is responsible for rolling back any actions that were performed on behalf of the aborted TLP.

2.3.9.2.2 Cut-Through Mode

In cut-through mode, the queue presents data to the application as soon as the first data for a packet is placed into the queue.

In the case of aborted TLPs, the application is responsible for rolling back any actions that were performed on behalf of the aborted TLP. The dllp_abort and tlp_abort signals are presented to the application at the same time as the eot signal.

2.3.9.2.3 Store-and-Forward Mode

In store-and-forward mode, only valid TLPs are forwarded to the application logic. Therefore, no rollback functionality is required by the application. All TLP with dllp_abort or tlp_abort asserted will be dropped by the receive queues of the core.

Flow Control credits are returned by the queue as packets are read out (even when the TLP was aborted and not presented to the application).

2.3.9.3 Order Enforcement

The ordering of TLPs within the same VC depends on the queuing architecture as follows:

- ❖ In the single-buffer configuration, the single queue is self-ordering FIFO (first in first out).
- ❖ In the multiple-buffer configuration, strict ordering is used, whereby all TLPs received within a given VC are presented to the application in the exact order as received from the wire.
- ❖ In the segmented-buffer configuration, either PCIe ordering rules or strict priority ordering are provided.

The ordering of TLPs between VC depends on the queuing architecture as follows:

- ❖ In the single-buffer and multi-buffer configurations, strict VC priority is used, whereby the higher numbered VCs are given a higher priority.
- ❖ In segmented buffer mode, you may select either strict VC priority (same as for single- and multi-buffer) or round robin.



2.3.10 Receive Queuing (SW)

The Switch supports only the segmented queue architecture. The segmented queue architecture supports three buffering modes: bypass mode, cut-through mode, and store-and-forward mode. The buffering mode is selectable for each TLP type: posted, non-posted and completion. The configurability is dependent on the queuing architecture.

The segmented queue architecture has one header buffer and one data buffer, but these two buffers are segmented by posted, non-posted and completion TLP type (versus single queue architecture).

RAM modules for segmented buffer mode are either instantiated inside the top-level module of the core or connected externally, which is configurable. See “[External RAM Interface \(optional\)](#)” for the external interface signals. External RAM may be preferred for customers doing RAM layout separately.

2.3.10.1 Queuing Architecture

Only the segmented-buffer configuration is supported with Switch applications.

2.3.10.1.1 Segmented-Buffer Receive Queue Configuration

The segmented-buffer queue architecture is designed for applications that want to enforce an ordering rule other than FIFO. This is the only queue architecture that can strictly adhere to the ordering rules of the *PCI Express Specification*. This queue architecture is relatively large in design. The segmented-buffer configuration uses a single memory module pair (header and data) for all TLP types and all VCs.

In the segmented-buffer configuration:

- ❖ The memory width is set automatically.
- ❖ The memory is divided into segments for Posted, Non-Posted, and Completion queues for each VC. The depth of each segment is set during hardware configuration, and can later be controlled dynamically by writing to the Port Logic registers.
- ❖ The operating mode is selected (bypass, cut-through, or store-and-forward) independently for each TLP type of each VC during hardware configuration. The operating mode (per TLP type and VC) can be controlled dynamically by writing to the Port Logic registers. If a TLP type is configured to be cut-through or store-and-forward, then it will be routed to the RTGT interfaces. If a TLP type is configured to be bypassed, then it will be routed to the radm_bypass interface. Once a Posted Request is configured in bypass mode, the application should not expect to send a Posted-write to the ELBI interface.
- ❖ The number of advertised credits is selected independently for each TLP type and each VC during hardware configuration; and the number can be selected dynamically by writing to the Port Logic registers.
- ❖ The receive queue priority for VCs can be set to either strict priority (higher-numbered VCs have higher priority) or round robin during hardware configuration; and the priority at runtime can be set by writing to the Port Logic registers.
- ❖ The ordering rules for TLP types can be set during hardware configuration and at runtime by writing to the Port Logic registers. The choices are either strict priority (Posted first, Completion second, Non-Posted third) or priority determined according to ordering rules set forth in the *PCI Express Specification*.



2.3.10.2 Buffering Modes

2.3.10.2.1 Bypass Mode

Queues in bypass mode are completely bypassed.

In the case of aborted TLPs, the core asserts either dllp_abort or tlp_abort to the application. The application is responsible for rolling back any actions that were performed on behalf of the aborted TLP.

2.3.10.2.2 Cut-Through Mode

In cut-through mode, the queue presents data to the application as soon as the first data for a packet is placed into the queue.

In the case of aborted TLPs, the application is responsible for rolling back any actions that were performed on behalf of the aborted TLP. The dllp_abort and tlp_abort signals are presented to the application at the same time as the eot signal.

2.3.10.2.3 Store-and-Forward Mode

In store-and-forward mode, only valid TLPs are forwarded to the application logic. Therefore, no rollback functionality is required by the application. All TLP with dllp_abort or tlp_abort asserted will be dropped by the receive queues of the core.

Flow Control credits are returned by the queue as packets are read out (even when the TLP was aborted and not presented to the application).

2.3.10.3 Order Enforcement

The ordering of TLPs within the same VC is as follows:

- ❖ Either PCIe ordering rules or strict priority ordering are provided.

The ordering of TLPs between VC is as follows:

- ❖ You may select either strict VC priority or round robin.



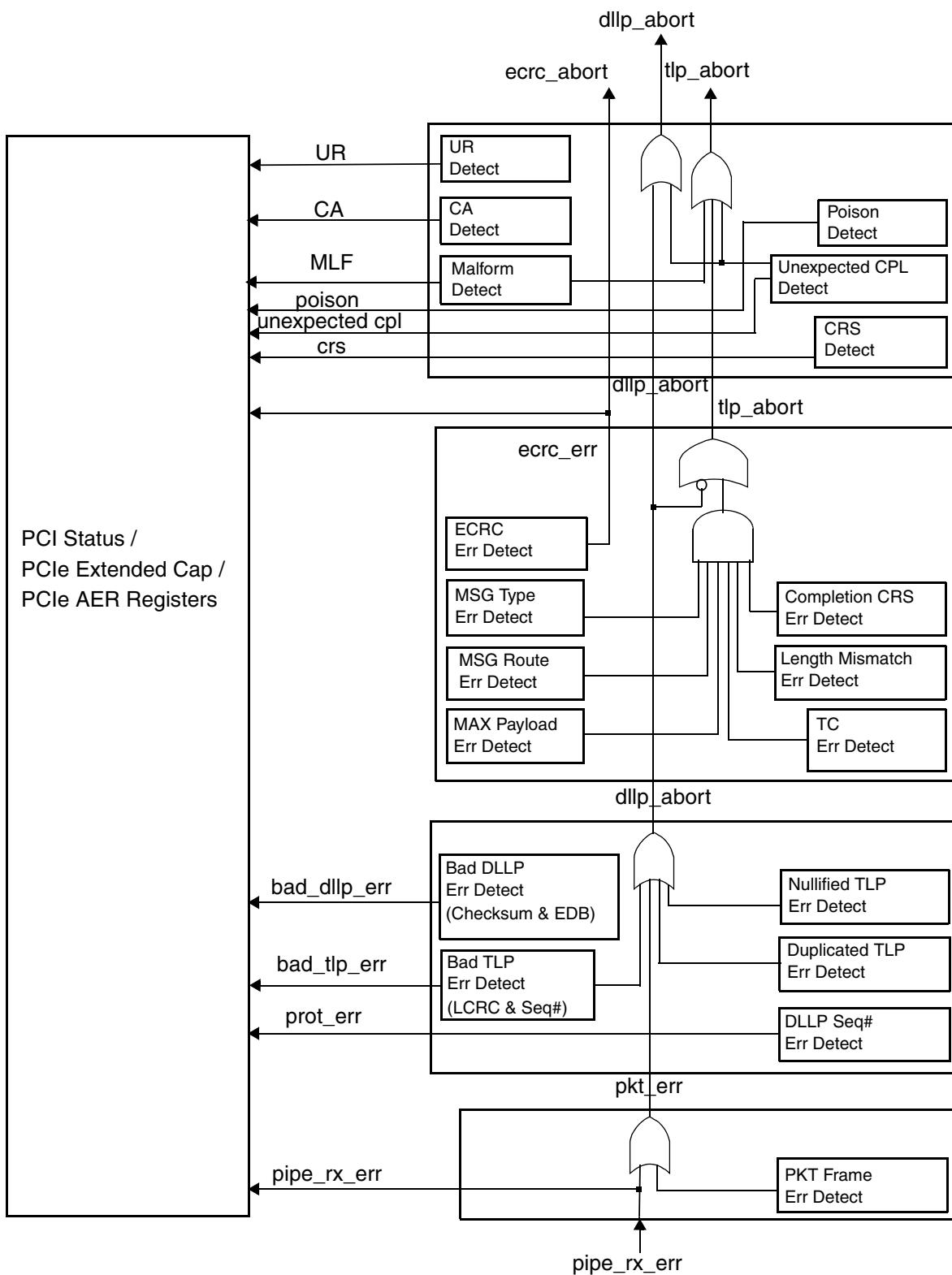
2.3.11 Error Handling for Received TLPs

Error handling within the core is designed primarily for the receive path. With the exception of parity error, errors are not inserted at transmission of the core. Applications have total control over the TLP that is formed in the transmit path.

This section describes receive error handling. In addition to the filtering rules described in “[Receive Filtering](#)” on page 60, the core implements Advanced Error Reporting and associated error message generation as defined in the *PCI Express Specification*, section 6.2.3.2.3 “Error Pollution.”

[Figure 2-1](#) provides a flow chart of the error detections as a packet received from a PCIe wire.



Figure 2-1 Flow Chart of Error Detections



The handling of detected TLP and DLLP abort conditions depends on the operating mode of the respective receive queue architecture and buffering mode.

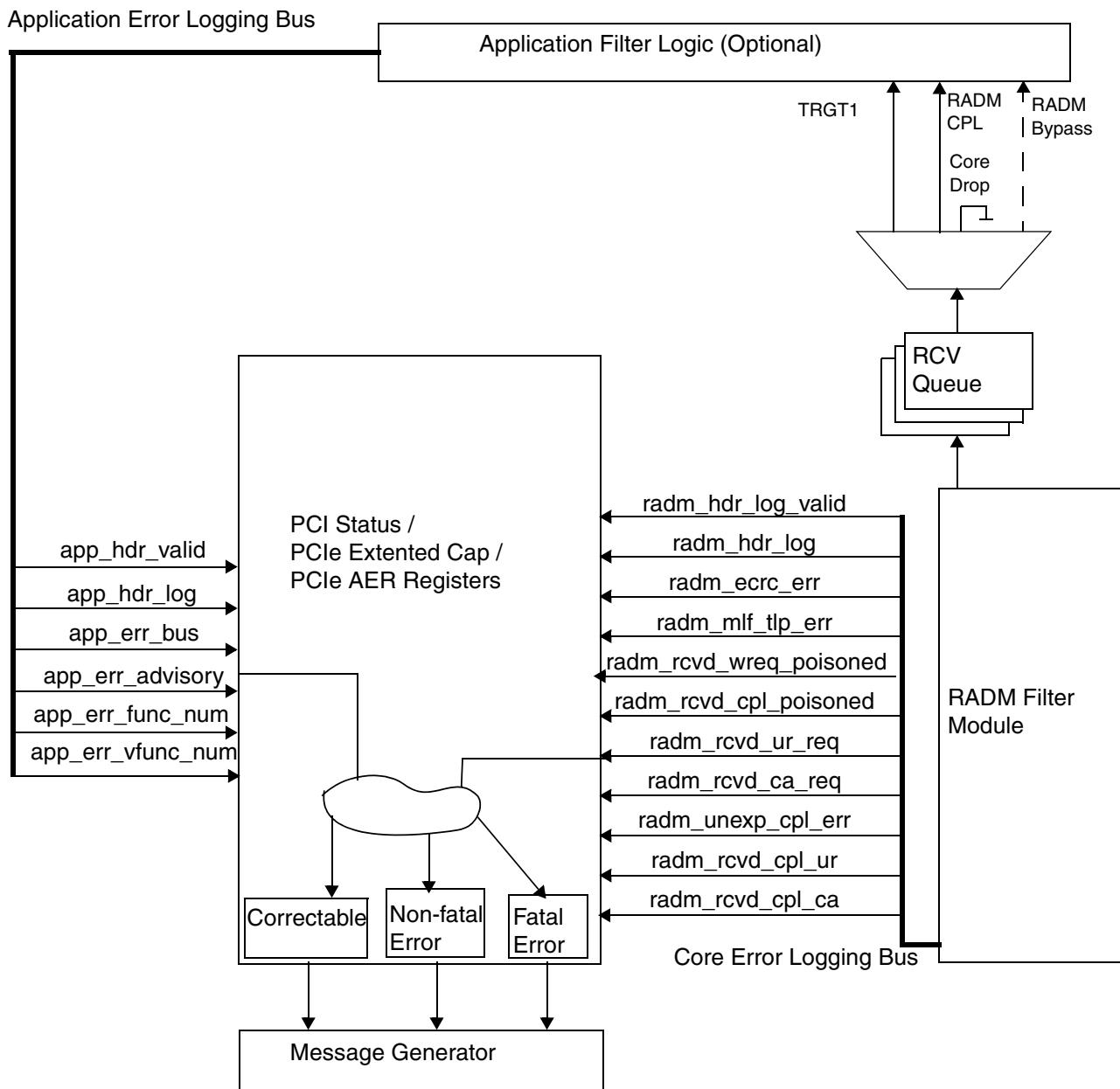
In store-and-forward mode, if a TLP abort or DLLP abort condition occurs, the core automatically drops the TLP. The TLP is never forwarded to the application.

In cut-through and bypass modes, transfer of the TLP to the application may begin before the TLP or DLLP abort condition is detected. Therefore, the core uses the TLP and DLLP abort signals to notify the application of the abort condition. For all transactions that reach the RTRGT1, RCPL or RBYPASS interfaces, a TLP abort and DLLP abort from the core signals the application to drop the packet and rewind its buffer pointer. The application's buffer should also be managed to prevent a jabber condition if the end delimiter is lost.

The application must roll back any action it has taken on behalf of the aborted TLP. In particular, certain DLLP abort conditions, such as a missing EOT delimiter, can cause the application's receive buffer to overflow when the receive queue is in bypass or cut-through mode. In store-and-forward mode, the core's internal receive buffer automatically prevents a buffer pointer wraparound from overwriting data in the buffer. In bypass or cut-through mode, the application must prevent buffer pointer wraparound.

Built into the core are all mandatory error detections, some optional error detections, and the error report mechanism based on the *PCI Express Specification*. The core also has an option for the application to turn off the filter rules and perform its own error checking.

[Figure 2-2](#) shows the flow for the application to return a PCIe error that it detected.

Figure 2-2 Application PCIe Error Report Mechanism

2.4 Interrupts

2.4.1 Interrupts (EP Mode)

In EP mode, the core supports the Legacy PCI INTx compatible interrupt emulation mechanism, the Message Signaled Interrupts (MSI), and the MSI-X. MSI-X generation is driven by the application through the MSI-X interface. A function can use either the Legacy PCI INTx mechanism or the MSI-X mechanism; a function cannot use both mechanisms. Different functions of the same device can use different mechanisms. For example, function 0 can use INTA while function 1 uses MSI and function 2 uses MSI-X.

2.4.1.1 Legacy PCI INTx Support

The core generates two Messages, Assert_INTx and Deassert_INTx, in response to assertion and the deassertion of the sys_int input. The Interrupt Pin register for each function determines whether the function uses INTA, INTB, INTC, or INTD.

In a single-function configuration, the core uses only INTA. An input signal, sys_int, is provided for the application to notify the core that an interrupt message should be sent. The sys_int signal is level-sensitive on a per-function basis. The rising edge transition of this signal triggers an interrupt assert message. The falling edge transition of this signal triggers an interrupt deassert message.

2.4.1.2 Message Signaled Interrupt (MSI) Support

MSI support is required for PCI Express devices. MSI-capable devices deliver interrupts by performing Memory Write transactions. The MSI is requested by application logic through the MSI interface; the core then generates the corresponding Memory Write. The optional MSI feature PVM (Per Vector Masking) is supported via a configuration selection. For details, see “[Message Signaled Interrupt \(MSI\) Interface \(DM / EP / SW\)](#)” on page [183](#).

2.4.1.3 MSI-X Support

The MSI-X capability structure is implemented in the CDM as part of the capabilities link list. The application must configure the values for Table and PBA Offset and BIR registers.

System software loads the MSI-X address and data values into the MSI-X Table and PBA structures, which must be implemented outside the core, by executing the required Memory Write Requests. For applications using the RTRGT1 interface, the RADM filter must be configured to route incoming Requests for the Table and PBA BARs to the RTRGT0 or RTRGT1 interface. For applications that do not use the RTRGT1 interface, no RADM filter configuration is needed. Requests to access the Table and PBA appear on the ELBI.

To request the core to send an MSI-X Message, the application uses the same request protocol as it does for MSI (using input ports that are shared between MSI and MSI-X), but also supplies the MSI-X address and data on dedicated MSI-X address and data input ports. The core then generates the corresponding Memory Write.

For details, see “[MSI-X Interface \(optional\) \(DM / EP / SW\)](#)” on page [188](#).

2.4.2 Interrupts (RC Mode)

In RC mode, the core accepts Assert_INTERRUPTx and Deassert_INTERRUPTx Messages from the downstream component and provides the decoded output signals on the SII (for example, radm_inta_asserted and radm_inta_deasserted).

For interrupts the downstream component transmits through the MSI-X mechanism, the core passes the received Memory Write to the application on the RTRGT1 interface.

2.4.2.1 PCI Express Hot-Plug Logic Interrupt and Wakeup

In RC mode, the Hot-Plug logic supports generation of Hot-Plug interrupts on the following Hot-Plug events:

- ❖ Power Fault Detected
- ❖ MRL Sensor Changed
- ❖ Presence Detect Changed
- ❖ Command Completed
- ❖ Attention Button Pressed
- ❖ Electromechanical Interlock Status Changed
- ❖ Data Link Layer State Changed

When MSI or MSI-X mode is enabled, the core notifies the RC application of Hot-Plug events using the hp_msi output. When INTx interrupt mode is enabled, the core notifies the application of Hot-Plug events using the hp_int output.

If PME is enabled, the The Hot-Plug logic generates a Hot-Plug wake-up signal on hp_pme, triggered by the above Hot-Plug events. The RC Core does not check if the PM state is D1, D2, or D3_{hot}. It is up to the application to check the value on pm_dstate to make sure the device is in D1, D2, or D3_{hot} upon receiving of hp_pme notification.



Note A command completion event does not trigger a wake event, even if the system is in a sleep state or if the device is in device state D1, D2, or D3_{hot}.

2.4.3 Interrupts (SW)

For an upstream Switch port, the core supports the Legacy PCI INTx compatible interrupt emulation mechanism, the Message Signaled Interrupts (MSI), and the MSI-X. MSI-X generation is driven by the application through the MSI-X interface. A function can use either the Legacy PCI INTx mechanism or the MSI-X mechanism; a function cannot use both mechanisms. Different functions of the same device can use different mechanisms. For example, function 0 can use INTA while function 1 uses MSI and function 2 uses MSI-X.

For a downstream Switch port, the core accepts Assert_INTx and Deassert_INTx Messages from the downstream component and provides the decoded output signals on the SII (for example, radm_inta_asserted and radm_inta_deasserted).

For interrupts the downstream component transmits through the MSI-X mechanism, the core passes the received Memory Write to the application on the RTRGT1 interface.

2.4.3.1 Legacy PCI INTx Support

The core generates two Messages, Assert_INTx and Deassert_INTx, in response to assertion and the deassertion of the sys_int input. The Interrupt Pin register for each function determines whether the function uses INTA, INTB, INTC, or INTD.

In a single-function configuration, the core uses only INTA. An input signal, sys_int, is provided for the application to notify the core that an interrupt message should be sent. The sys_int signal is level-sensitive on a per-function basis. The rising edge transition of this signal triggers an interrupt assert message. The falling edge transition of this signal triggers an interrupt deassert message.

As an upstream Switch Port, and with legacy interrupts enabled, the core transmits Assert_INTx and Deassert_INTx Messages to the upstream device under any of the following conditions:

- ❖ Switch application logic asserts/deasserts the sys_int input, in which case the core generates the corresponding Assert_INTA or Deassert_INTA Messages to the upstream device.
- ❖ The Switch application logic asserts/deasserts any of the following input signals to indicate that the application logic's Virtual Interrupt wires have changed state: dp_inta, dp_intb, dp_intc, and dp_intd. To determine when to assert/deassert dp_inta, dp_intb, dp_intc, or dp_intd, the Switch application logic decodes/combines the following output signals from the downstream Switch Ports:
 - ◆ radm_inta_asserted/radm_inta_deasserted
 - ◆ radm_intb_asserted/radm_intb_deasserted
 - ◆ radm_intc_asserted/radm_intc_deasserted
 - ◆ radm_intd_asserted/radm_intd_deasserted
- ❖ Any of the following Hot-Plug events occurs and the associated interrupt is enabled in the Slot Control register:
 - ◆ Power Fault Detected
 - ◆ MRL Sensor Changed
 - ◆ Presence Detect Changed
 - ◆ Command Completed
 - ◆ Attention Button Pressed

2.4.3.2 Message Signaled Interrupt (MSI) Support

MSI support is required for PCI Express devices. MSI-capable devices deliver interrupts by performing Memory Write transactions. The MSI is requested by application logic through the MSI interface; the core then generates the corresponding Memory Write. As an upstream Switch Port, the core generates MSI Messages in response to requests from the application on the MSI interface. For details, see “[Message Signaled Interrupt \(MSI\) Interface \(DM / EP / SW\)](#)” on page 183.

2.4.3.3 MSI-X Support

The MSI-X capability structure is implemented in the CDM as part of the capabilities link list. The application must configure the values for Table and PBA Offset and BIR registers.

System software loads the MSI-X address and data values into the MSI-X Table and PBA structures, which must be implemented outside the core, by executing the required Memory Write Requests. For applications using the RTRGT1 interface, the RADM filter must be configured to route incoming Requests for the Table and PBA BARs to the RTRGT0 or RTRGT1 interface. For applications that do not use the RTRGT1 interface, no RADM filter configuration is needed. Requests to access the Table and PBA appear on the ELBI.

To request the core to send an MSI-X Message, the application uses the same request protocol as it does for MSI (using input ports that are shared between MSI and MSI-X), but also supplies the MSI-X address and data on dedicated MSI-X address and data input ports. The core then generates the corresponding Memory Write.

For details, see “[MSI-X Interface \(optional\) \(DM / EP / SW\)](#)” on page 188.

2.4.3.4 PCI Express Hot-Plug Logic Interrupt and Wakeup

For a downstream Switch Port, the Hot-Plug logic supports generation of Hot-Plug interrupts on the following Hot-Plug events:

- ❖ Power Fault Detected
- ❖ MRL Sensor Changed
- ❖ Presence Detect Changed
- ❖ Command Completed
- ❖ Attention Button Pressed
- ❖ Electromechanical Interlock Status Changed
- ❖ Data Link Layer State Changed

When MSI or MSI-X mode is enabled, the core notifies the RC application of Hot-Plug events using the `hp_msi` output. When INTx interrupt mode is enabled, the core notifies the application of Hot-Plug events using the `hp_int` output.

If PME is enabled, the The Hot-Plug logic generates a Hot-Plug wake-up signal on `hp_pme`, triggered by the above Hot-Plug events. The RC Core does not check if the PM state is D1, D2, or D3_{hot}. It is up to the application to check the value on `pm_dstate` to make sure the device is in D1, D2, or D3_{hot} upon receiving of `hp_pme` notification.



Note A command completion event does not trigger a wake event, even if the system is in a sleep state or if the device is in device state D1, D2, or D3_{hot}.



2.5 Flow Control

PCI Express implements a differentiated, credit-based Flow Control system to prevent overflows at the receiver. In contrast to the simple XON/XOFF type flow control of the Ethernet protocol, the PCI Express Flow Control system requires that the consumer of data advertise the available buffer space for each type and priority of traffic. The Flow Control mechanism is divided into two phases: the initialization phase and the update phase. The core automatically performs both of these phases with minimal support required from the application.

The Flow Control for VC0 must be initialized following Link initialization, but prior to sending normal traffic. This initialization is performed by the Flow Control Initialization state machine. The initialization process involves exchanging information with the Link partner about the size of the receiver's buffers for each type of packet data: Posted, Non-Posted, and Completion. Header and payload buffers for each of these types are tracked and reported independently. Flow Control must be initialized for Virtual Channel 0 (VC0) before the Data Link Layer's link state moves into the DL_ACTIVE state, and normal traffic can begin flowing. Additional VCs (if any) are initialized following this, intermingled with the regular traffic already flowing on VC0. Initialization of other VCs begins when the VCs are enabled. The VC0 traffic has priority over non-VC0 flow control initialization.

The core provides a configurable solution to choose the number of credits to advertise per type and per VC. It can be configured to support multiple VCs as well as infinite credits. The core performs all required Flow Control protocol handshakes. The core currently provides a solution in which the application does not have to deal with Flow Control updates and checks. By default, the RADM is responsible for returning Flow Control credits as data is read out of the RADM queuing structure. Additionally, the core provides optional signals to enable the application to handle Flow Control returns in an application-specific manner (see ["System Information Interface \(SII\)" on page 200](#)).

The core does not return Flow Control credits for packets that have Data Link Layer errors.

2.6 Address Translation

Address translation is used for mapping different address ranges to different memory spaces comprehended by the application. A typical example will map the AMBA memory space to PCI memory space when the application has the PCIe-to-AMBA bridge in place. Other examples could include switch applications or IOV applications. The core provides only a mechanism to allow the application to take control over address translation and the core will perform the actual swap of the address and other fields of a request.

The application can get the DWC PCIe core's address translation hooks by setting the ADDR_TRANSLATION_SUPPORT_EN parameter. The translation is performed by the application, which is able to customize its address mapping.

Address translation is accomplished in two parts. The first is provided by the core which swaps the address of an inbound or outbound transaction. The second is provided by application translation logic which does the address range comparison and calculates the new translated address.

The core offers the address swapping function for both transmit and receive paths through a group of signals based on a defined timing as shown in [Figure 3-28](#).

Please refer to “[Address Translation \(optional\)](#)” on page [305](#) for the detailed signal descriptions.

Please refer to Chapter 3 of the User Manual for details of an example address translation unit included in the reference RTL design.



Note This address translation is for a proprietary local address translation implementation and is not related to the PCI-SIG ATS specification support.

2.6.1 Address Translation for the Transmit Path

The application's address translation logic determines a type, function number and address based on the address of the outbound request (signal x_addr_out). The DWC core swaps the address/type/function provided by the application only when an outbound transaction is a memory read or write. The memory lock bit of a read will be kept as-is. The application can translate a memory transaction into the IO or CFG space of an outbound transaction by changing the TYPE TLP header field.

2.6.2 Address Translation for the Receive Path

The address, function number, type, and BAR number of an inbound request are presented to the application's address translation logic. The application translation logic must generate a translated address for the core resulting in an inbound request address at the core's application interface (TRGT1 interface). The signal table for the address translation interface is provided in [Table 3-30](#).

Applications can notify the core of error conditions. There are two bits of error status defined at the receive address translation interface. The error status results in a return completion of UR, CA, or others depending on whether the core is configured to form the return completions.

Both received IO and Configuration requests can be translated into a memory request by changing the TYPE TLP header field.

Received Messages and completions will never be translated.

There is a strict timing relationship between the core's output address versus the application returned translated address. All translations must be finished within one core-clk (clock) cycle.

The core provides output ports containing the address and the function number. Address translation should be performed each cycle, based on the outputs provided by the core. See [Figure 3-28](#) for the address translation timing diagram.

2.6.3 PCI-SIG Address Translation Services (ATS)

ATS is a complete specification that describes an end-to-end protocol for offloading translation resources in the platform.

The Synopsys implementation of ATS is to solely allow the application to use/set the Address Type (AT) field of the TLP header.

The actual usage of these bits, in implementation of an Address Translation Cache (ATC) or Translation Agent, is outside the scope of the core's implementation of ATS.

The parameter for enabling access to the AT header field is `CX_ATS_ENABLE.

Individual access to the AT field in either the inbound (RX) or outbound (TX) direction can be further enabled/disabled via the `ATS_RX_ENABLE and `ATS_TX_ENABLE parameters.

Setting these parameters makes the client*_tlp_ats ("Transmit Client0 Interface (XALI0) Signals"), radm_bypass_ats ("RCPL Interface Signals"), and radm_trgt1_ats signals ("RTRGT1 Interface Signals") available for the application to use. See the relevant IO sections for more details.



2.7 Peer-to-Peer Support (P2P)

In certain multi-port PCI Express applications, such as a PCIe Switch or multi-port Root Complex, it is useful to allow TLPs to flow from one port to another instead of strictly up or down the PCI hierarchy, from root to endpoint or endpoint to root. The PCI Express specification describes this capability in terms of routing peer-to-peer transactions between ports.

Though identified by the PCI Express specification, the mechanism and support of this feature is considered to be outside the scope of the base specification. The Synopsys DesignWare PCI Express RC and SW ports provide support for this mechanism.

To enable the peer-to-peer feature, the CX_P2P_ENABLE parameter must be set. This may be done via the coreConsultant GUI interface or via a set_configuration_parameter command in the configuration batch file.

Enabling peer-to-peer support modifies the operation of the core in the following ways:

1. Includes additional transmit client interface signals (for example, client0_tlp_p2p_en) to distinguish between peer-to-peer and local traffic.
2. On transmit, the core distinguishes between locally generated traffic and peer-to-peer traffic, and adds only locally generated requests to the completion lookup table (LUT).
3. On receive, only completions corresponding to requests originated by the device will be subject to the normal completion LUT processing. Completions of peer-to-peer requests will not reference the completion LUT.
4. On transmit, the core preserves the original requester or completer IDs for peer-to-peer traffic while automatically adding the appropriate ID for locally generated traffic.
5. All TLP header fields are preserved in both transmit and receive directions; that is, all reserved fields are presented to the receive interfaces and accepted via the client interfaces.
6. The core will optionally append ECRC fields to only those TLPs that are locally generated, passing peer-to-peer traffic along without modification.
7. The core never strips ECRC from packets when peer-to-peer support is enabled.

The following limitations apply with respect to the Peer-to-peer feature:

- ❖ Segmented buffer must be selected for the RADM queue mode.
- ❖ The core does not support Target LUT functionality in conjunction with peer-to-peer support.
- ❖ Stripping of ECRC on TLP reception is not supported in conjunction with peer-to-peer support.
- ❖ The core does not split transmit packets. The application must handle such feature if the peer-to-peer packet exceeds the max payload limitation.



2.8 PCI Express 2.0 Features (Gen2)

The DesignWare Cores for PCI Express 2.0 currently support all of the non-optional PCI Express 2.0 features defined in the *PCI Express 2.0 specification*.

A PIPE signal, mac_phy_rate, is used to negotiate the Link data rate. The core drives mac_phy_rate to indicate the negotiated Link data rate. For mac_phy_rate, a value of zero indicates 2.5 Gbps, and a value of one indicates 5.0 Gbps. The MAC changes the rate signal and waits for a pulse on the phy_mac_phystatus signal to confirm that the PHY has accepted the requested rate. The DWC PCIe cores support two different mechanisms of achieving the PCI Express 2.0 rate, Dynamic Frequency and Dynamic Width.

When supporting Gen2 Dynamic Frequency, the core operates at either 125 MHz or 250 MHz at the Gen1 rate. When operating at the Gen2 rate, the core's clock frequency is changed to 250 MHz or 500 MHz. When supporting Gen2 Dynamic Width, the clock frequency of the core remains constant and the number of valid symbols to the PHY changes. If operating at the Gen1 rate, the core currently supports 1 out of 2 symbols, or 2 out of 4 symbols. If operating at the Gen2 rate, the core currently supports 2 out of 2 symbols, or 4 out of 4 symbols. Signals, registers, register changes, and other changes in support of Gen2 are noted in the respective chapters in this databook.

The PCIe core uses core_clk to sample the phy_mac_phystatus signal during speed changes. The PHY generates phy_mac_phystatus based on pipe_clk. Therefore, the following restrictions are placed on DWC_PCIE external logic when the PHY is Gen2 Dynamic Frequency and the PCIe core is Gen2 Dynamic Width (i.e. when core_clk is not equivalent to pipe_clk).

1. ensure that core_clk is toggling when the PHY returns phy_mac_phystatus for a speed change OR
2. the external logic must hold phy_mac_phystatus (for the speed change) until core_clk toggles again

For other configurations, the core_clk is equivalent to the pipe_clk, and this restriction is not needed.

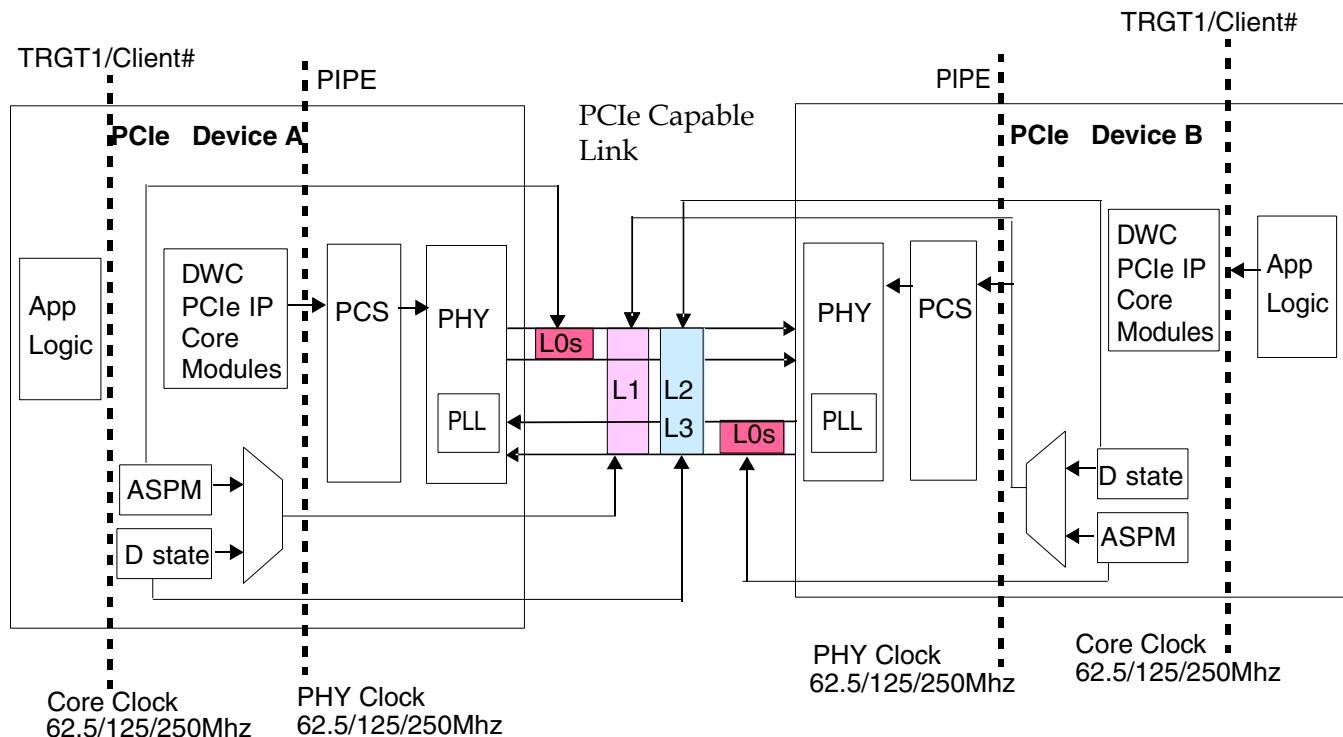
2.9 Power Management

There are two types of power management operations:

- ❖ Software controlled PCI power management operations
- ❖ Active state power management operation (ASPM) for PCIe device only

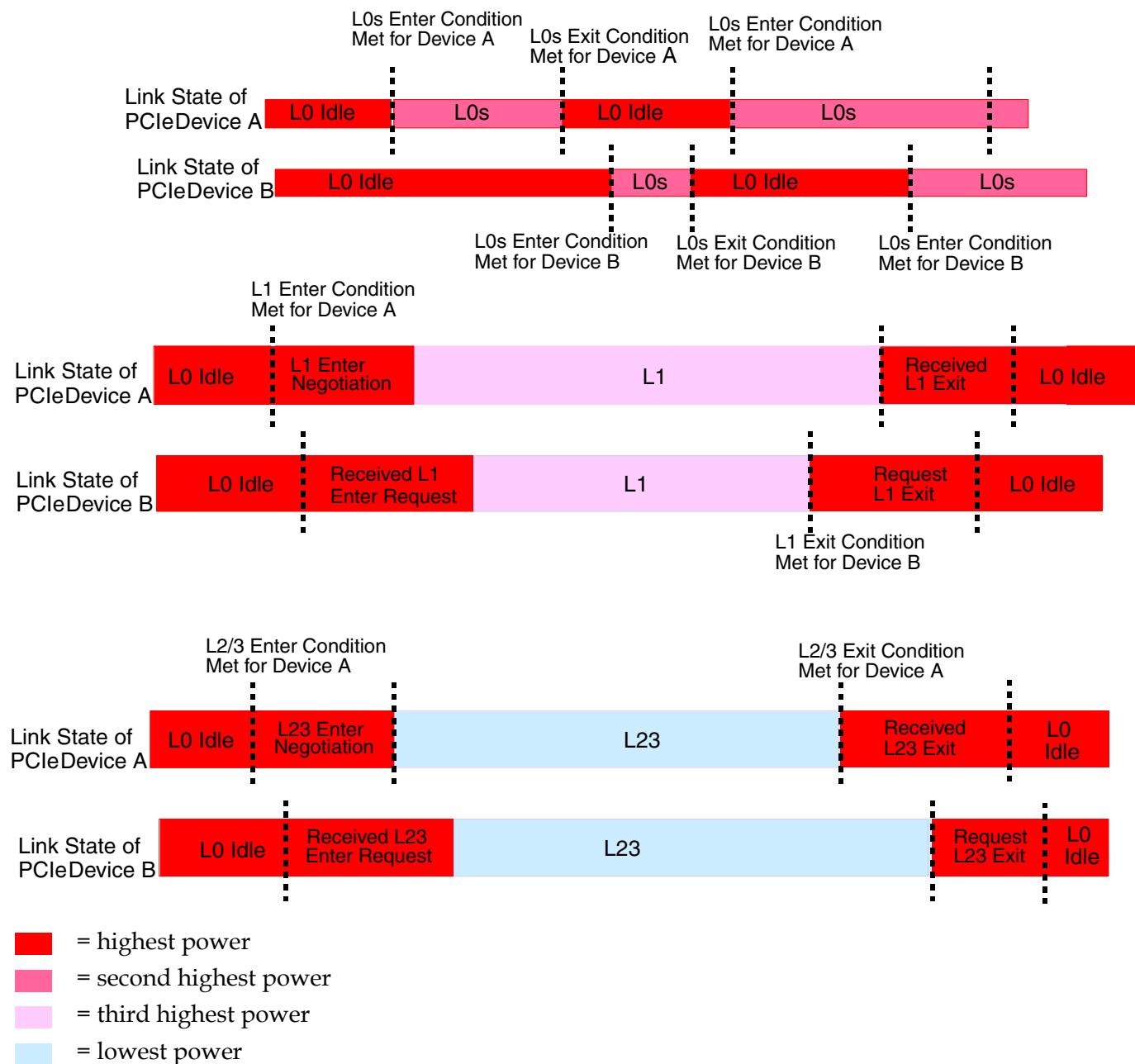
[Figure 2-3](#) shows the capable link state and its control conditions.

Figure 2-3 Power Management - Capable Link State and Control Conditions



The L0s link state is controlled by the ASPM L0s enter condition met state. The L1 link state is controlled either by the ASPM L1 enter condition met state, or by the D-state (D1, D2, or D3) of the PCIe device. The D-state of the PCIe device is programmable by software. The L2/L3 ready state is controlled by D-state and power turn-off event.

The power saving of links in lower power states is greater as the link state numbers get larger. [Figure 2-4](#) shows the links states of PCIe devices and the relationships of power down states between link partners.

Figure 2-4 Relationships of Power Down States between Link Partners

2.9.1 L0s Power Down

L0s is a low power state enabled by Active State Power Management (ASPM). ASPM enabled devices can only control L0s entrance of the transmitter. The receiver L0s is controlled by the remote devices.

2.9.1.1 L0s Enter Conditions (All Conditions Met)

1. ASPM L0s is enabled.
2. L0s enter conditions defined by *PCI Express Specification* for a duration of time and there is no higher stage of power down requested.
3. The timeout value is controlled by the DEFAULT_L0S_ENTR_LATENCY parameter.

2.9.1.2 L0s Exit Conditions (Any Condition Met)

1. Any DLLP or TLP pending to be sent.
2. L1 enter condition met.
3. PCIe link partner request to enter into link recovery.

2.9.2 L1 Power Down

L1 is a power down state enabled either by ASPM or by the software controlled D1, D2 or D3 state (which is programmed by the system power management unit). L1 state is a bi-directional link power down state. Both link partners must negotiate to go to L1 state.

2.9.2.1 L1 Enter Conditions Due to ASPM (All Conditions Met)

There are three scenarios that result in the L1 state:

- ❖ Scenario 1: L1 Idle Timeout From L0s
 1. ASPM L1 and L0s are enabled.
 2. Link state is in L0s for both transmitter and receiver of the link, and Port Logic Ack Frequency Register bit 30 is set to 0 (default setting) OR Link state is in L0s of transmitter and Port Logic Ack Frequency Register bit 30 is set to 1.
 3. L1 enter conditions defined by PCIe spec for a duration of time and there is no higher stage of power down requested.
 4. The timeout value is controlled by the DEFAULT_L1_ENTR_LATENCY parameter.
- ❖ Scenario 2: L1 Idle Timeout from L0
 5. ASPM L1 is enabled and L0s is not enabled.
 6. Link state is in L0.
 7. L1 enter conditions defined by PCIe spec for duration of time, and there is no higher stage of power down requested.
 8. The timeout value is controlled by the DEFAULT_L1_ENTR_LATENCY parameter.
- ❖ Scenario 3: Application Controlled
 9. ASPM L1 is enabled.
 10. Application request to enter L1 by asserting signal app_req_entr_l1.
 11. L1 enter conditions defined by PCIe spec is met.



The app_req_entr_l1 is a pulse. The core latches the command and makes sure that L1 has been entered.

2.9.2.2 L1 Enter Conditions Due to D1/D2/D3 States (All Conditions Met)

1. All functions that are programmed to D1, D2 or D3 states.
2. Always enter L1 when L2/L3 PM turn-off negotiation has not yet been done.

2.9.2.3 L1 Exit Conditions (Any Condition Met)

1. Software requests a higher stage of power down.
2. Any DLLP or TLP pending to be sent.
3. Application requesting exit of L1 by asserting signal app_req_exit_l1.
4. Link partner requesting exit of L1.

Once L1 has exited, another L1 entry will not be initiated for 10us if the enter L1 condition is due to ASPM. If the enter L1 condition is due to lower power D-state, the core will enter L1 again after a wait time of



cfg_cpl_sent_count cycles defined in PL register. This wait time to ensures the exit conditions have been served.

2.9.3 L2/L3 Power Down

The core has control over the L2 or L3 ready link state. After the L2/L3 ready is entered, the downstream device will begin preparation for the power and clock removal. After main power has been removed, the link will transition to L2 if Vaux is provided, or it will transition to L3 if no Vaux is provided. L2/L3 ready is a bi-directional link power down state.

2.9.3.1 L2/L3 Enter Conditions (All Conditions Met)

1. PME_Turn_Off/Pme_To_Ack handshake has been completed at any of D0,D1,D2,D3 states.
2. Application is ready to be turned off by asserting signal app_ready_entr_l23.

2.9.3.2 L2/L3 Exit Conditions (Any Condition Met)

1. Device is programmed with capability to support PME and application requests wakeup by asserting the apps_pm_xmt_pme signal or by triggering a native hot-plug event when D-state is in D1, D2 or D3.
2. Link partner requesting exit of L2/L3.

The core supports beacon signaling by asserting signal pm_phy_beacongen or wake when a wake-up event is initiated by a PCIe device.

2.10 Single Root I/O Virtualization (SR-IOV) (EP / DM in EP mode)

2.10.1 Overview

The Synopsys DW_pcie implements revision 1.0 of the *Single Root I/O Virtualization and Sharing Specification (SR-IOV)* which defines extensions to enable multiple System Images (SI) running on a single Root to share PCI Express hardware resources. The parameter for enabling SR-IOV is CX_SRIOV_ENABLE. Additional information on the affected IO can be found in the Signal Description section of most interfaces in “[Signal Descriptions](#)”.

To support the SR-IOV functionality, the following features are also supported:

- ❖ Function Level Reset (FLR)
- ❖ Alternate Routing ID Interpretation (ARI)

2.10.2 Features and Limitations

- ❖ Implicates ARI and FLR functionality
- ❖ All VFs support
 - ◆ Error Handling
 - ◆ MSI-X Capability
 - ◆ PCI Express Capability
- ❖ Supports up to
 - ◆ Total of 8 PFs (CX_NFUNC)
 - ◆ 256 VFs (CX_MAX_VF_n) per PF, where n is {7..0}. This value sets TotalVFs and InitialVFs in the PF SR-IOV Capability Structure.
 - ◆ Number of VFs per PF (CX_MAX_VF_n) must be a power of two
- ❖ Total of VFs (all VFs for all PFs) is 256 for ASIC technologies. (CX_NVFUNC = SUM {CX_MAX_VF_n})
- ❖ Total of VFs (all VFs for all PFs) is 16 for FPGA technologies. (CX_NVFUNC = SUM {CX_MAX_VF_n})
- ❖ Supports separate VF Stride and VF Offset values (per PF) for both ARI Capable and non-ARI Capable Hierarchies. Selection is performed by host software via the 'ARI Capable Hierarchy' bit of the Control register in the PF SR-IOV Capability Structure.
- ❖ VF Stride per PF must be
 - ◆ A power of two.
 - ◆ Greater than or equal to the number of PFs.
- ❖ VF Offset must be
 - ◆ A power of two.
 - ◆ Greater than or equal to the number of PFs.
- ❖ TRGT_CPL_LUT in SR-IOV not supported
- ❖ VF Migration not supported
- ❖ Optional PM capability (per VF) not yet supported.
- ❖ Optional AER capability (per VF) not yet supported.
- ❖ Optional MSI capability (per VF) not yet supported.

2.10.2.1 Backward Compatibility issues with some non-ARI-aware Hypothetical Software

When the core is configured for SRIOV, the behavior becomes that of an ARI device, which is consistent with the ARI specification. This implies the following:

- ❖ For routing IDs, requester IDs, and completer IDs associated with the ARI device, the traditional 5-bit device number and 3-bit function number fields are interpreted as a single 8-bit field, which is used to address up to 256 functions.
- ❖ Instead of requiring "voting" logic between the different functions, the maximum payload size settings are managed solely by the settings in Function 0.



The two items above may impact backward compatibility with some non-ARI-aware hypothetical software. However, they should not impact compatibility with non-ARI aware software that (a) uses Device Number = 0 in all configuration requests targeting the ARI device and (b) configures all functions within the ARI device with the same maximum payload size, or configures Function 0 with the maximum payload size among all functions.

2.10.2.2 Features controlled by the ARI Capable Hierarchy bit

The ARI capable hierarchy bit in the SRIOV Extended Capability structure is used by the core to determine, among the two available values, which one to use for VF Stride and First VF offset. This behavior is consistent with the SRIOV specification,

2.10.3 Function Level Reset (FLR) (EP / DM in EP mode)

Function Level Reset is a mechanism to allow specific functions to be reset without affecting other functions or the link as a whole. FLR applies to both physical functions (PFs) and virtual functions (VFs). The reason for supporting both PFs and VFs is because resetting a single function of a device without impacting the other active functions provides necessary isolation and security.

Function Level Reset (FLR) is an optional feature for non-SR-IOV Endpoints, but mandatory for SR-IOV Endpoints. FLR is enabled by the CX_FLR_ENABLE parameter, and can be enabled independent of SR-IOV.

For more information on FLR I/O and FLR operation details, see [Function Level Reset \(FLR\) \(DM / EP\)](#).

2.10.3.1 Features and Limitations

- ❖ FLR must be enabled for SR-IOV operation.
- ❖ FLR implements per-function resets for PFs and VFs.
- ❖ The driver should put the function into a quiescent state before initiating an FLR.
- ❖ The core will return Unsupported Request (UR) completions for requests to functions in FLR.

2.10.4 Alternate Routing ID Interpretation (ARI) (EP / DM in EP mode)

ARI is required to support a large number of Physical Functions (PFs) or Virtual Functions (VFs) in SR-IOV. With the ARI capability, 256 functions may be implemented per bus number. The DWC_pcie EP core supports both ARI Capable and non-ARI Capable Hierarchies. The device automatically selects which stride and which offset to advertise depending on the setting of ARI Capable Hierarchy bit in the Control register of the PF0 SR-IOV Capability Structure. The determination of the VF Number depends on offset and stride settings now advertised by the device. In the general case these are per-PF settings (i.e., different PFs may have different settings). The stride and offset values may also depend on the ARI Capable Hierarchy bit being set or not by the System, and how the system configures NumVFs.

2.10.4.1 Features and Limitations

- ❖ ARI must be enabled for SR-IOV operation.
- ❖ ARI is not supported in a non-SR-IOV environment.
- ❖ The physical functions (ARIDs) are sequential and not sparse/non-sequential.
- ❖ Once a device supports ARI (even if ARI is not enabled in the system), the following occurs:
 - ◆ Max payload size is determined solely by settings in Function 0.
 - ◆ Clock Power Management is the same for all functions.
 - ◆ ASPM Control is determined solely by settings in Function 0.
 - ◆ Common Clock Configuration is determined solely by settings in Function 0.

2.11 Completion Timeout Ranges

Timeout ranges are supported as defined in the *PCI Express 2.0 Specification*. The Device Capabilities 2 Register (Offset 24h) shows support for all ranges. The Device Control 2 Register (Offset 28h) will have a reset value equal to the default value in the spec: "0000b Default range: 50 us to 50 ms". If the default value is used then the timeout will be in "Range B: 0101b: 16ms to 55ms." This range was chosen for the default because the *PCI Express 2.0 Specification* states "It is strongly recommended that the Completion Timeout mechanism not expire in less than 10 ms." Below is a table of the spec values versus the PCI Express core values for the ranges.

Table 2-9 PCIe Core Completion Timeout Ranges versus PCI Express 2.0 Specification

Range	Encoding	Spec Minimum	Spec Maximum	PCIe Core Minimum	PCIe Core Maximum
Default	0000b	50μs	50ms	28ms	44ms
A	0001b	50μs	100μs	65μs	99μss
A	0010b	1ms	10ms	4.1ms	6.2ms
B	0101b	16ms	55ms	28ms	44ms
B	0110b	65ms	210ms	86ms	131ms
C	1001b	260ms	900ms	260ms	390ms
C	1010b	1s	3.5s	1.8s	2.8s
D	1101b	4s	13s	5.4s	8.2s
D	1110b	17s	64s	38s	58s

3

Signal Descriptions

3.1 Top-Level I/O Diagrams

Figures 3-1 through Figures 3-6 illustrate the top-level I/O signals for all of the PCIe IP cores. These signals are grouped according to interface. The signal interface names in these figures are cross-referenced to the corresponding signal description tables that follow. (For example, click on “Clock and Reset Signals” in Figures 3-1 to hyper-link to Section 5-14, “Clock and Reset Signals,” for a detailed description of these signals.)

The signal interfaces groups are as follows:

- ❖ Transmit Client Interfaces
- ❖ Receive Completion Interface (RCPL) (DM / RC / EP)
- ❖ Bypass Interface
- ❖ Receive Target 1 Interface (RTRGT1) (optional)
- ❖ External Local Bus Interface (ELBI) (DM / EP / SW)
- ❖ Data Bus Interface (DBI)
- ❖ Message Signaled Interrupt (MSI) Interface (DM / EP / SW)
- ❖ MSI-X Interface (optional) (DM / EP / SW)
- ❖ Vital Product Data (VPD) Support (optional)
- ❖ Power Budgeting (optional)
- ❖ Vendor Message Interface (VMI)
- ❖ System Information Interface (SII)
- ❖ PHY Interface (PIPE)
- ❖ Synopsys PHY Interface
- ❖ Clock and Reset
- ❖ External RAM Interface (optional)
- ❖ Address Translation (optional)
- ❖ Peer-to-Peer (optional)
- ❖ Function Level Reset (FLR) (DM / EP)



Note The signal descriptions include the fields below. Read the description of these fields prior to reviewing the signal descriptions so the intention of the information is clear.

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. If a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the in-active state (opposite of the active state).
- ❖ **Registered:** Indicates whether or not the signal is registered directly at the core boundary. 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 core. A value of “Optionally” means that the signal is not registered at the core boundary by default, but you can choose to insert an input/output register by selecting the corresponding configuration option.
- ❖ **Synchronous to:** Indicates which clock (core_clk or aux_clk) the signal is synchronous to or if the signal is asynchronous.
- ❖ **Device Type:** The device (IP core) in which this function/signal is used.
- ❖ **Populated by:** Name of `define that populates this signal (for example, `TRGT1_POPULATE). In some cases, the `define implies that the signal is optional (for example, `CX_SRIOV_ENABLE).
- ❖ **Valid when:** Assertion or deassertion of signal(s) that validates the signal being described (for example, data is valid when data_dv is asserted).



Note In some cases, the `define in the “Populated by” implies that the signal is optional (for example, `CX_SRIOV_ENABLE). All signals named *vfun* are valid only when the SR-IOV feature is enabled by `CX_SRIOV_ENABLE.



Note For any signal that is not used, the default pull-up/pulldown definition is simply the opposite of the Active State. For example, for the client0_tlp_hv signal the active state is high, so if this pin is not used, it should be pulled down.

 **Note**

Some of the IOs have widths defined by NFUNC_WD and NVFUNC_NUM_WD.

NFUNC_WD is the width needed to communicate the number of physical functions, the value of which is set by the CX_NFUNC_WD parameter as follows:

If SR-IOV is not enabled, NFUNC_WD = CX_NFUNC_WD = 3

If SR-IOV is enabled, NFUNC_WD = CX_NFUNC_WD = \log_2 CX_NFUNC, where CX_NFUNC is the number of PF's and is in the range 1 to 8.

NVFUNC_NUM_WD is the width needed to communicate the total number of virtual functions, the value of which is set by the CX_NVFUNC_NUM_WD parameter as follows:

NVFUNC_NUM_WD = CX_NVFUNC_NUM_WD = \log_2 (CX_NVFUNC + 1),

where CX_NVFUNC = CX_MAX_VF_0 + CX_MAX_VF_1 + ..CX_MAX_VF_7,

where CX_MAX_VF_N is the maximum number of VFs supported for this PF (N=0-7)

 **Note**

In this release, SR-IOV is only available for the EP product. In the next release, it will be available for the DM product (in EP mode).

In the signal descriptions for signals of the name *vfun*; please read EP only instead of DM (EP mode) / EP in the Device row.

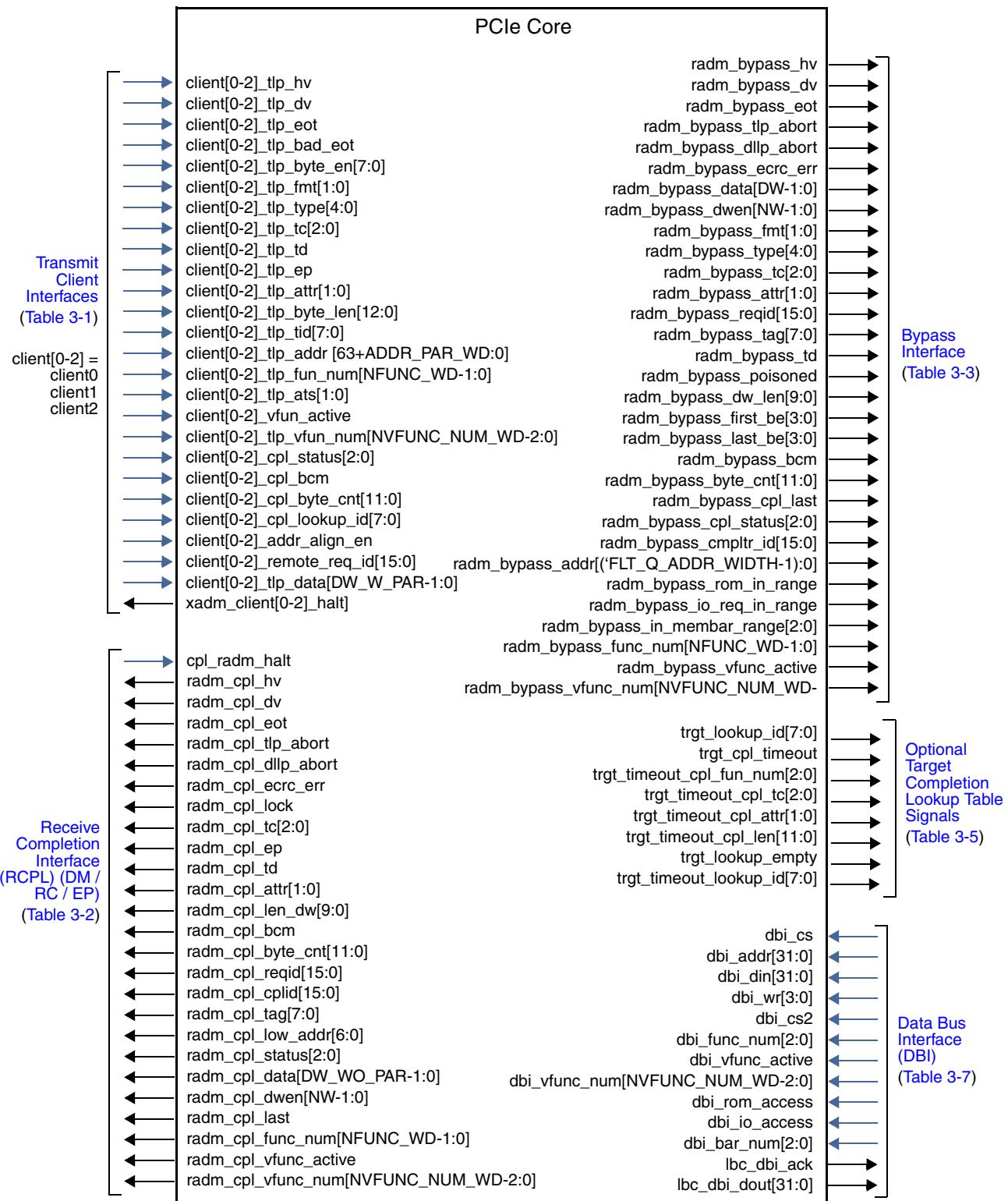
Figure 3-1 Top-Level I/O Diagram (1 of 6)

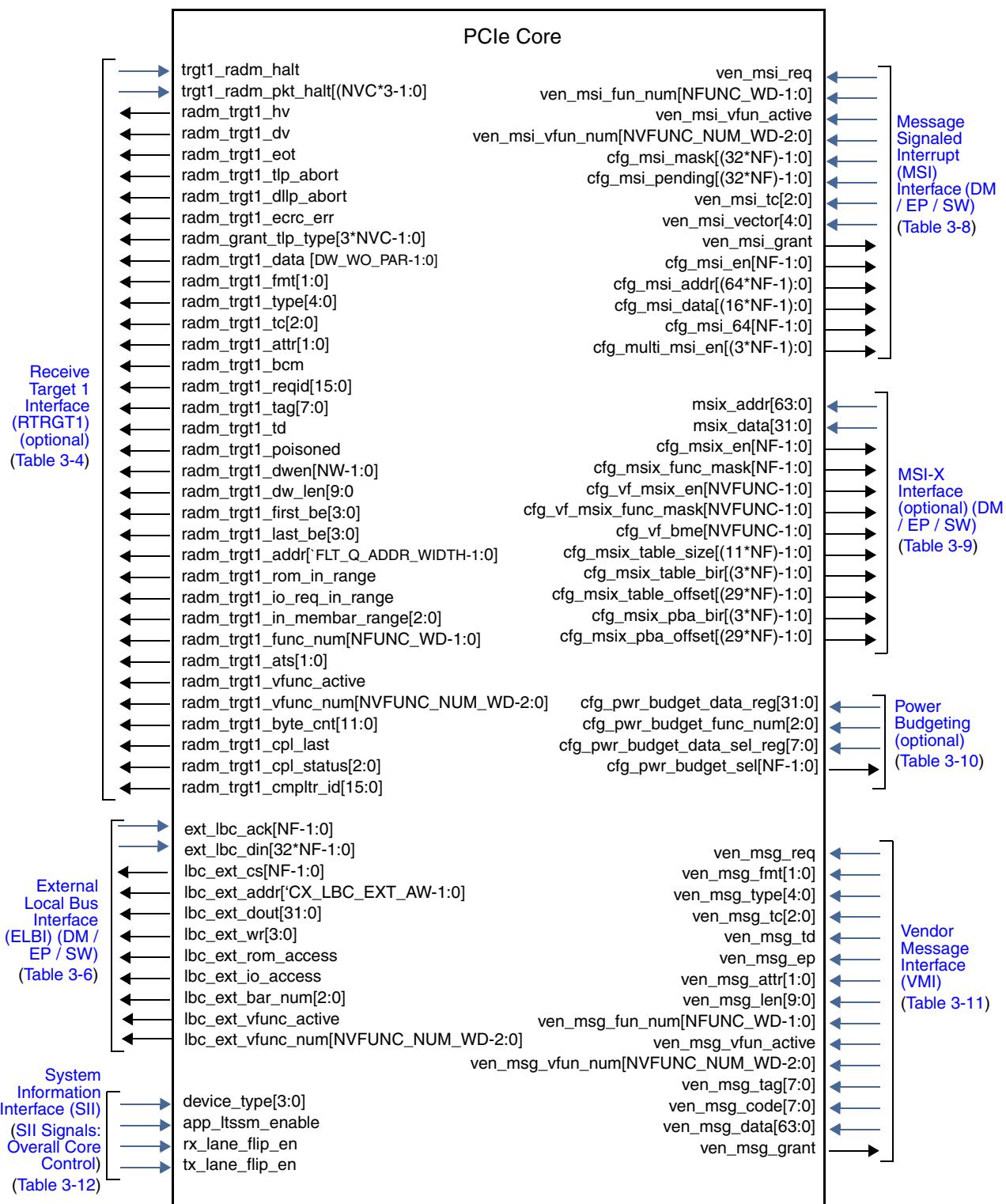
Figure 3-2 Top-Level I/O Diagram (2 of 6)

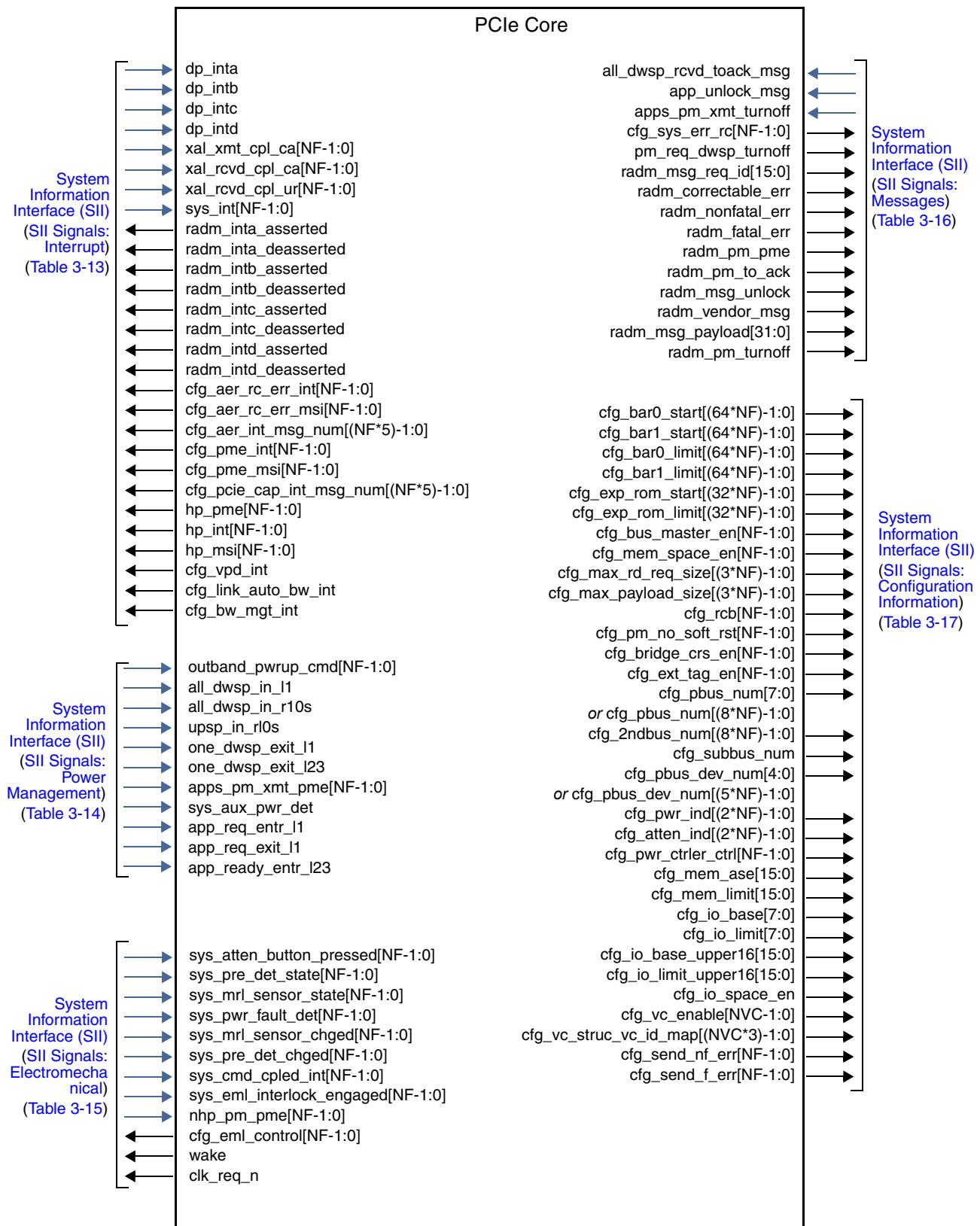
Figure 3-3 Top-Level I/O Diagram (3 of 6)

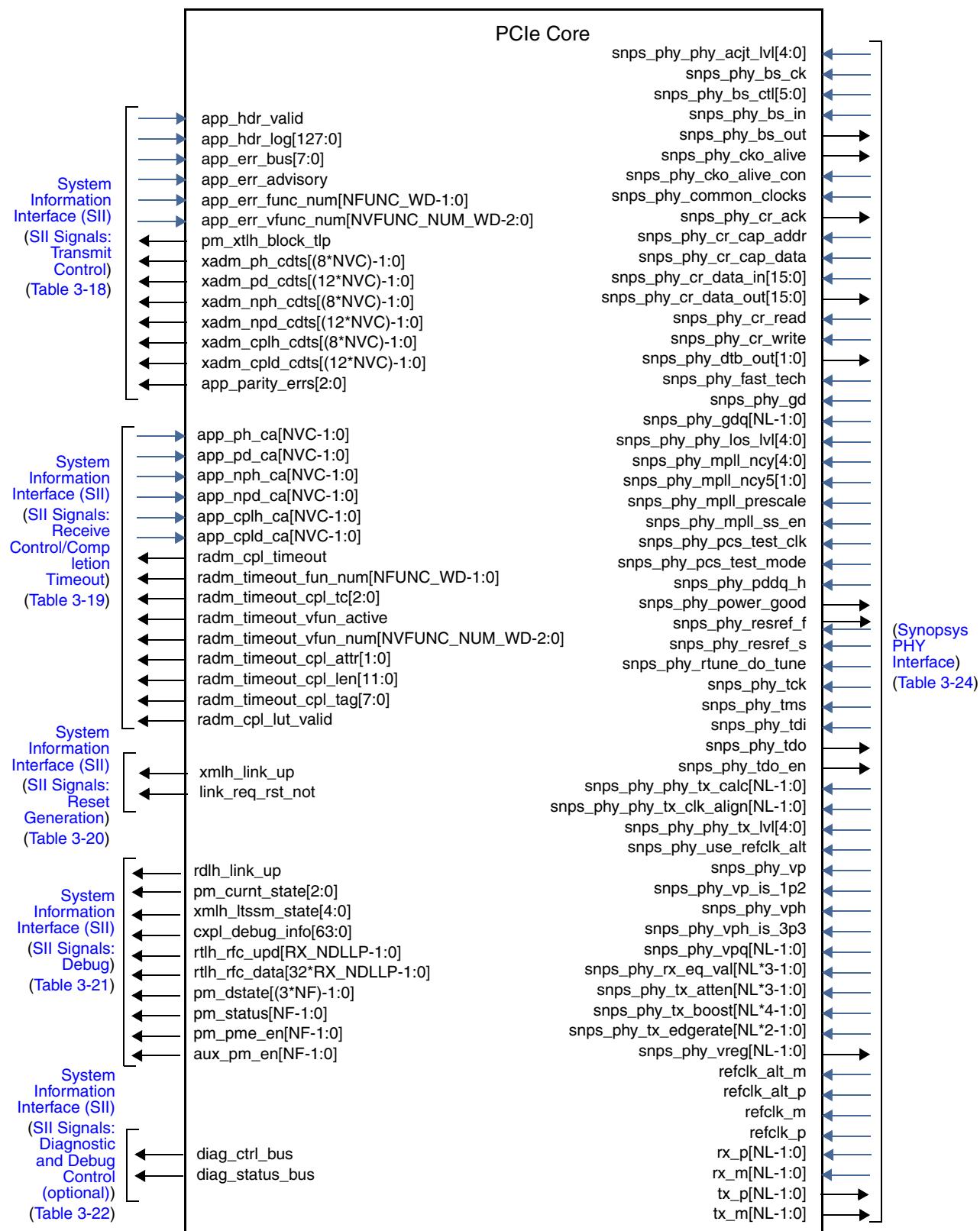
Figure 3-4 Top-Level I/O Diagram (4 of 6)

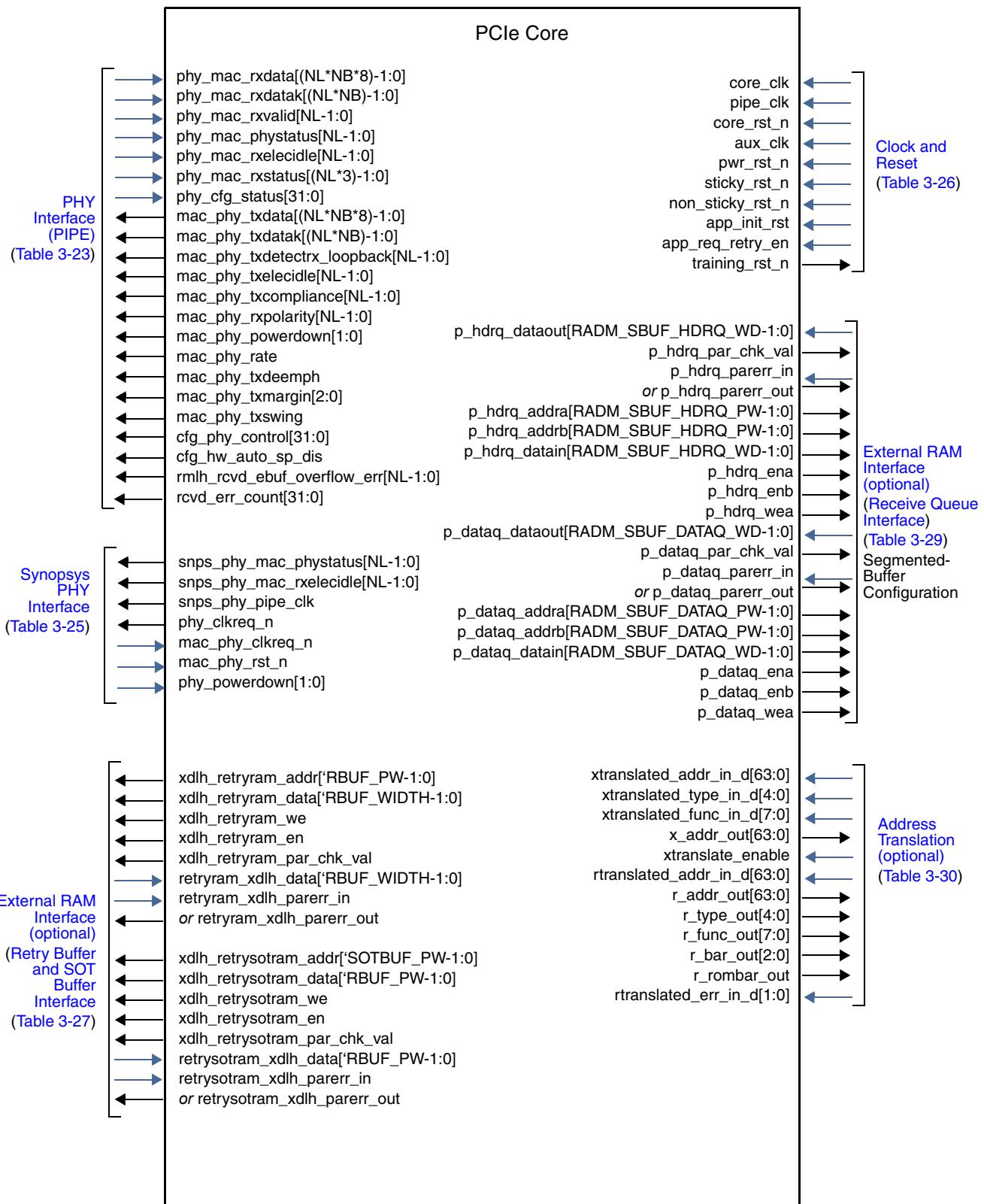
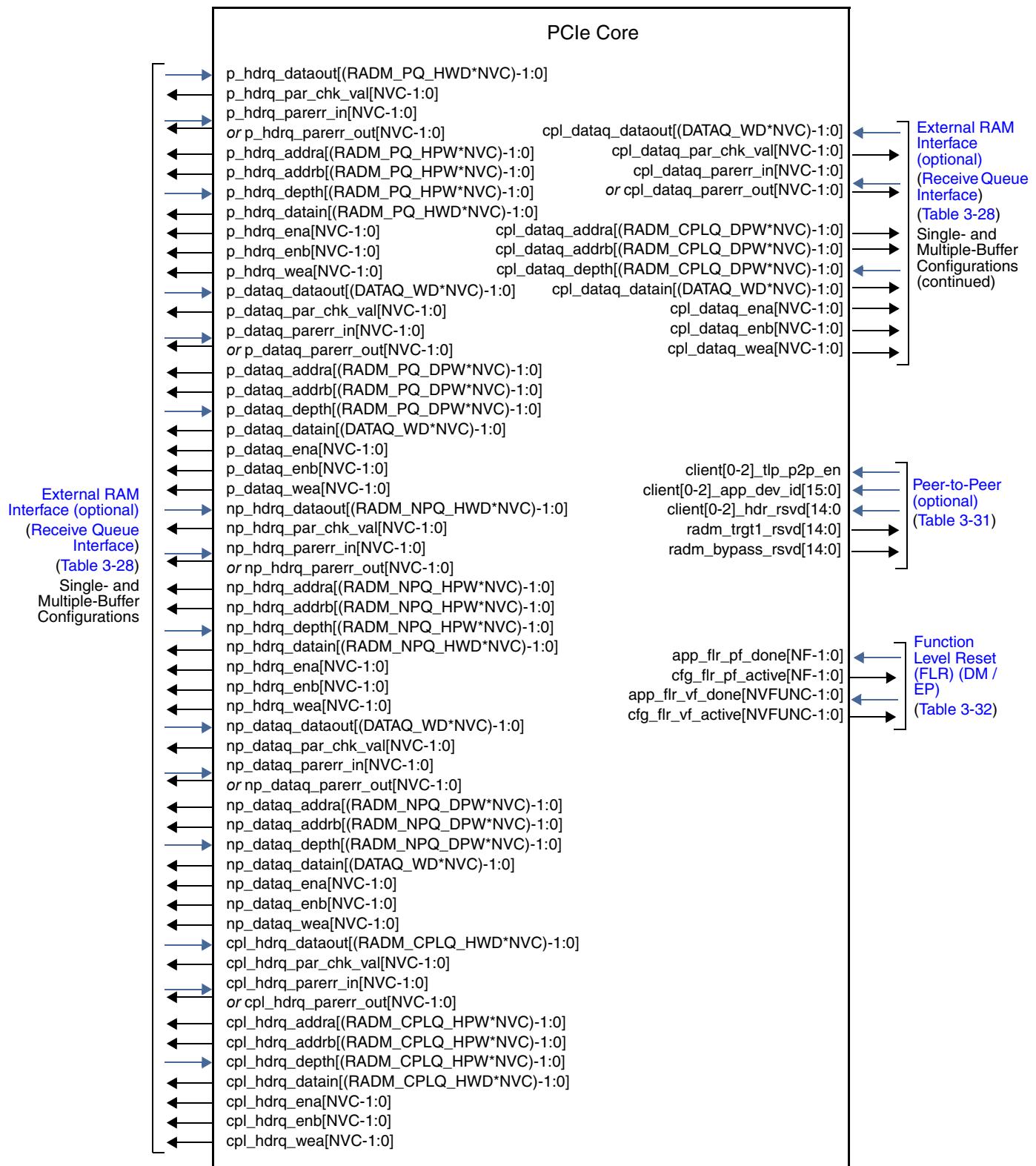
Figure 3-5 Top-Level I/O Diagram (5 of 6)

Figure 3-6 Top-Level I/O Diagram (6 of 6)

Transonic Client Interfaces

3.2 Transmit Client Interfaces

XALI0 and XALI1 are two identical client interfaces connected to two clients at the application side. Each client can be either a Requester or a Completer.

The XALI2 interface (if implemented) is identical to XALI0 and XALI1. Requests on XALI0, XALI1, and XALI2 can be asserted in the same clock cycle. The core automatically arbitrates requests received from the XALI0, XALI1, and XALI2 interfaces.

3.2.1 Transmit Client Interface Protocol Rules

The transmit client provides information on the header bus and payload data on the data bus. The core does not have internal queues for transmit TLPs. Once the application starts a transaction transfer request, it must continue to stream data unless the core halts the client interface. The protocol rules for each of the transmit client interfaces (using the client0 signals as the example) are:

- ❖ For transactions with an associated payload, the header valid (client0_tlp_hv) and data valid (client0_tlp_dv) signals must be asserted in the same cycle.
- ❖ The end of transaction indication signal (client0_tlp_eot) must be asserted during the last cycle of data valid or header valid when no payload is present.
- ❖ When the xadm_client0_halt signal is asserted, the application must maintain the information on the header bus and data bus.
- ❖ The client0_tlp_bad_eot signal is used to abort a transfer that is in progress. It must be asserted on the same cycle as client0_tlp_eot.
- ❖ Not all header information has to be driven dynamically. Some header input signals can be hardwired if suitable for the application.
- ❖ To send a vendor-defined Message through the transmit client interface, the application asserts the Message code on client0_tlp_byte_en, and the last 2 DWORDs of the header (vendor-defined) through client0_tlp_addr[63:0], where client0_tlp_addr[63:32] is the 4th DWORD of the header. If there is data to send with the vendor-defined Message, the application sends the data through client0_tlp_data, as with other TLP data.
- ❖ If you choose the `TRGT_CPL_LUT_EN configuration option, then the application does not need to provide certain TLP header information when transmitting Completions because the core stores and tracks the information in an internal target Completion lookup table. To transmit a Completion, the application must first store the lookup ID from the incoming Request (provided on trgt_lookup_id) and then provide the same lookup ID on client0_cpl_lookup_id when transmitting the Completion (see [Table 3-5](#) for signal descriptions). When using the target Completion lookup table feature, the application does not need to supply the following header information for Completions: client0_remote_req_id, client0_cpl_byte_cnt, client0_tlp_attr, client0_tlp_addr, client0_tlp_tid.



Note When address alignment is enabled, application logic must make sure that the length of the TLP is less than the maximum payload supported. For example, if maximum payload is set to 128 bytes (32 DWORDs) and the application sends a request of 128 bytes but with a non-DWORD-aligned address the core will send the request with a DWORD-aligned address with Length = 33 DWORDs. This will be detected as a Malformed TLP on the destination.

3.2.2 Transmit Client0 Interface (XALI0) Signals

[Table 3-1](#) defines the XALI0 interface signals.

Table 3-1 XALI0 Interface Signals

Signal	Input/O utput	Description
client0_tlp_hv	I	<p>Function: Indicates that the TLP header data on the TLP Header ports is valid. The header data must remain valid until client0_tlp_hv is deasserted.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: Not validated by another signal</p>
client0_tlp_dv	I	<p>Function: Indicates that the TLP payload data on client0_tlp_data is valid. The application must not assert client0_tlp_dv for TLPs that do not include a payload, such Memory Read Requests.</p> <p>The application must present the first payload data on the same cycle with the TLP header.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: Not validated by another signal</p>
client0_tlp_eot	I	<p>Function: Indicates the end of TLP payload data for a TLP that includes a payload. For a TLP that does not include a payload (such as a Memory Read Request), the application must assert and deassert client0_tlp_eot at the same time as client0_tlp_hv.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: Either client0_tlp_hv or client0_tlp_dv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_bad_eot	I	<p>Function: Indicates that the current TLP must be nullified. To nullify a TLP, the application must assert client0_tlp_bad_eot in the same cycle as client_tlp_eot. When client0_tlp_bad_eot is asserted, the core nullifies the TLP (inverts the LCRC and inserts EDB as the final framing symbol in the transmitted TLP). The application can use client0_tlp_bad_eot to nullify a TLP either because of a known error condition or to abort the current transaction if the current transaction is being blocked (for example, due to lack of FC credits):</p> <ul style="list-style-type: none"> • To cancel a request that has not been granted (xadm_client0_halt has never been de-asserted for the transfer), the CLIENT_PULLBACK feature from coreConsultant must be selected. In addition, the following signals must be driven as follows. Assert client0_tlp_bad_eot and client0_tlp_eot for one clock cycle while xadm_client0_halt is asserted (indicating that the request has not been granted), then de-assert client0_tlp_hv, client0_tlp_dv, client0_tlp_bad_eot, and client0_tlp_eot to terminate the request. Without the CLIENT_PULLBACK feature, the application has to send the full packet and mark the EOT and BAD_EOT at the end. With the CLIENT_PULLBACK feature, the application can mark EOT and BAD_EOT when it realizes that it needs to nullify the TLP, without having to present the full packet. • To cancel a request that is in progress (xadm_client0_halt has been asserted for the transfer), assert client0_tlp_bad_eot and client0_tlp_eot for one clock cycle while xadm_client0_halt is deasserted (indicating that the request has been granted), then deassert client0_tlp_hv, client0_tlp_dv, client0_tlp_bad_eot, and client0_tlp_eot to terminate the request. In this case, the core nullifies the TLP as described above. • If the TLP is a one-cycle request (client0_tlp_hv and client0_tlp_eot is asserted on the same cycle) and client0_tlp_bad_eot is also asserted, then this request is ignored. Sending a nullified TLP is not very useful because the link partner has to ignore it anyway. The application should realize that this request is nullified (in this case ignored) and de-assert everything (_hv, _eot, bad_eot, etc). <p>Active State: High Registered: No Synchronous to: core_clk Device Type: All Populated by: Always populated Valid when: client0_tlp_eot is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_byte_en[7:0]	I	<p>Function: Byte enables for the first and last DWORD of the TLP:</p> <ul style="list-style-type: none"> [3:0]: Byte enables for the first DWORD [7:4]: Byte enables for the last DWORD <p>A TLP can have “holes” between bytes if the TLP is less than 8 bytes (2 DWORDs) long, according to the <i>PCI Express 2.0 specification</i> (not all valid bytes need to be contiguous). However, a TLP is not allowed to have holes between bytes if the TLP is more than 8 bytes.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: Not valid when client0_addr_align_en is asserted</p>
client0_tlp_fmt[1:0]	I	<p>Function: The value to be used for the Format field of the TLP header:</p> <ul style="list-style-type: none"> 00b: 3 DWORD header, no data 01b: 4 DWORD header, no data 10b: 3 DWORD header, with data 11b: 4 DWORD header, with data <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_type[4:0]	I	<p>Function: The value to be used for the Type field of the TLP header, as defined in the <i>PCI Express 2.0 specification</i>.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_tc[2:0]	I	<p>Function: The value to be used for the Traffic Class (TC) field of the TLP header (000b–111b).</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_td	I	<p>Function: TLP Digest bit. It is expected that this signal is tied to 0 by the application. When device is RC, EP or DM, it is expected that there is no ECRC added at the client interface from the application logic and this signal must be tied to 0 by the application. If the application does not drive this signal to 0, the transmit behavior is undefined. It is the core's responsibility to add ECRC and set the TD bit of the header when cfg_ecrc_gen_en is set. When device is a switch (SW), it is expected that ECRC is determined by the application logic. The core will not add ECRC to transactions from client interfaces of a switch device. The TD bit should be set by the application logic corresponding to the transaction ECRC field in the switch device.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_ep	I	<p>Function: When asserted, the core sets the Poisoned TLP (EP) bit in the TLP header.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_attr[1:0]	I	<p>Function: Attributes field for the TLP header</p> <ul style="list-style-type: none"> • Bit 0: No Snoop • Bit 1: Relaxed Ordering <p>For the Client0 interface, if you select the `TRGT_CPL_LUT_EN configuration option, the core does not use client0_tlp_attr for Completions. Instead, the core takes the attr value from the target Completion lookup table entry specified on the client0_cpl_lookup_id input.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_byte_len[12:0]	I	<p>Function: The number of bytes in the TLP (not including ECRC):</p> <ul style="list-style-type: none"> • 0000h: 0 bytes • up to • 1000h: 4096 bytes <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_tid[7:0]	I	<p>Function: Tag field for the TLP header. The application must provide a valid Tag on client0_tlp_tid for all Posted and Non-Posted Requests.</p> <p>The value of the `CX_MAX_TAG configuration option determines the maximum number of outstanding Client Requests.</p> <p>The application is expected to manage the Tag for every Posted and Non-Posted Request.</p> <p>For Completions, the client0_tlp_tid value must be the Tag from the corresponding Request.</p> <p>If you select the `TRGT_CPL_LUT_EN configuration option, the core does not use client0_tlp_tid for Completions. Instead, the core takes the Tag value from the target Completion lookup table entry.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_addr[63+ADDR_PAR_WD:0]	I	<p>Function: The Address field for the TLP header with optional parity bits. The TLP address can be a 64- or 32-bit address. When sending Completions, the core uses client0_tlp_addr[6:0] for the Lower Address field of the Completion TLP.</p> <p>When sending a Configuration Request in RC mode, client0_tlp_addr must contain the following information:</p> <ul style="list-style-type: none"> • [31:24]: Bus Number • [23:19]: Device Number • [18:16]: Function Number • [11:8]: Extended Register Number • [7:2]: Register Number <p>The base width of client0_tlp_addr is 64. The number of extra parity bits depends on the value you select for the Client Data/Address Bus Parity Protection configuration option ('CX_CLIENT_PAR_MODE').</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_fun_num[NFUNC_WD-1:0]	I	<p>Function: Indicates from which physical function (PF) the request is coming. The core uses client0_tlp_fun_num to form the Requester ID for Requests and the Completer ID for Completions.</p> <p>For a single-function device, set client0_tlp_fun_num to 000b.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_ats[1:0]	I	<p>Function: AT field in the TLP header.</p> <p>Active State: NA</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not DM, RC, SW)</p> <p>Populated by: 'ATS_TX_ENABLE'</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_vfun_active	I	<p>Function: Indicates that the TLP request is coming from a VF (virtual function).</p> <p>state 0: No VF is active and client0_tlp_vfun_num is invalid. A PF is valid and identified by client0_tlp_fun_num.</p> <p>state 1: A VF is active and is identified by client0_tlp_vfun_num.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not DM, RC, SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_tlp_vfun_num[NVFUN_C_NUM_WD-2:0]	I	<p>Function: Indicates from which virtual function (VF) the request is coming. The core uses this signal to form the Requester ID for Requests and the Completer ID for Completions.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC, SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: client0_tlp_hv and client0_tlp_vfun_active are asserted</p>
client0_cpl_status[2:0]	I	<p>Function: The value to be used in the Completion Status field of a Completion TLP header:</p> <ul style="list-style-type: none"> • 000b: Successful Completion • 001b: Unsupported Request • 010b: Configuration Request Retry Status • 100b: Completer Abort <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted. Only used when XALI0 is connected to a Completer.</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_cpl_bcm	I	<p>Function: Not used for PCI Express Endpoints or Root Ports. The client0_cpl_bcm input is only used for PCI-X bridges and must be hardwired to 0 for Endpoint and Root Port applications.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: PCI-X Bridge, must be hardwired to 0 for Endpoint and Root Port applications.</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_cpl_byte_cnt[11:0]	I	<p>Function: The value to be used for the Byte Count field of a Memory Read Completion TLP. The value on client0_cpl_byte_cnt indicates the number of bytes remaining to be delivered for the Request, as defined in the <i>PCI Express 2.0 specification</i>:</p> <ul style="list-style-type: none"> • 001h: 1 byte • FFFh: 4095 bytes • 000h: 4096 bytes <p>For the Client0 interface, if you select the `TRGT_CPL_LUT_EN configuration option, the core does not use client0_cpl_byte_cnt for Completions. Instead, the core takes the current Byte Count value from the target Completion lookup table entry specified on the client0_cpl_lookup_id input.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: PCI-X Bridge, must be hardwired to 0 for Endpoint and Root Port applications.</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted. Only used when XALI0 is connected to a Completer.</p>
client0_cpl_lookup_id[7:0]	I	<p>Function: The target Completion table lookup ID for a Completion TLP, which must match the trgt_lookup_id value for the corresponding Request. When using the optional target Completion lookup table feature, the core uses the Byte Count, Tag, Attributes, Requester ID, and Address values stored in the target Completion lookup table for the Completion TLP header instead of the values of the corresponding input signals.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_addr_align_en	I	<p>Function: When client0_addr_align_en is asserted, the core performs address alignment and generates the first and last byte enables based on the address and number of bytes of the TLP requested from the client interface.</p> <p>When client0_addr_align_en is deasserted, the core takes the first and last byte enables from client0_tlp_byte_en and does not address-align on the data.</p> <p>To support the PCI Express checkerboard option to the application, deassert client0_addr_align_en. Otherwise, assert client0_addr_align_en to enable address alignment.</p> <p>Note: If you do not select the Enable ECRC Support (`CX_ECRC_ENABLE) option, and the application is generating the ECRC, do not assert client0_addr_align_en for TLPs that contain application-generated ECRC. If the TLP from the application does not contain ECRC, then the application may assert client0_addr_align_en to enable address alignment.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: client0_tlp_hv is asserted</p>
client0_remote_req_id[15:0]	I	<p>Function: The Requester ID to be used in the TLP header of a Completion TLP. The value must match the Requester ID of the corresponding incoming Request:</p> <ul style="list-style-type: none"> • [15:8]: Bus number • [7:3]: Device number • [2:0]: Function number <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-1 XALI0 Interface Signals (Continued)

Signal	Input/O utput	Description
client0_tlp_data [DW_W_PAR-1:0]	I	<p>Function: The data payload for the TLP. The base width of client0_tlp_data is the application datapath width, which is automatically determined according to the selected operating frequency and number of Lanes.</p> <ul style="list-style-type: none"> For a 32-bit core, the base width is 32 (client0_tlp_data[31:0]) For a 64-bit core, the base width is 64 (client0_tlp_data[63:0]) For a 128-bit core, the base width is 128 (client0_tlp_data[127:0]) <p>The number of extra parity bits depends on the value you select for the Client Data/Address Bus Parity Protection configuration option (~CX_CLIENT_PAR_MODE).</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: client0_tlp_dv is asserted</p>
xadm_client0_halt	O	<p>Function: The core asserts xadm_client0_halt when it is not ready to process a transaction. The header and payload input for Client0 must remain unchanged while xadm_client0_halt is asserted.</p> <p>The conditions that cause the core to assert xadm_client0_halt include:</p> <ul style="list-style-type: none"> The XADM is busy servicing one of the other transmit client interfaces. There are not enough Flow Control credits available to send the requested TLP type (Posted, Non-Posted, or Completion). The LTSSM is not ready. A replay is in progress. ECRC or LCRC insertion is in progress. <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always populated</p> <p>Valid when: Not validated by another signal</p>

3.2.3 Transmit Client1 Interface (XALI1) Signals

The XALI1 interface is identical to the XALI0 interface, except that the signal names contain the string “client1” instead of “client0”. Otherwise, the functions of the XALI1 signals are the same as the XALI0 signals described in [Table 3-1](#).

Application logic can assert input signals on both XALI0 and XALI1 on the same cycle; the core will arbitrate between them.

3.2.4 Transmit Client2 Interface (XALI2) Signals (optional)

The XALI2 interface is present only if `CLIENT2_POPULATED is defined. If present, the XALI2 interface is identical to the XALI0 interface, except that the signal names contain the string “client2” instead of “client0”. Otherwise, the functions of the XALI2 signals are the same as the XALI0 signals described in [Table 3-1](#).

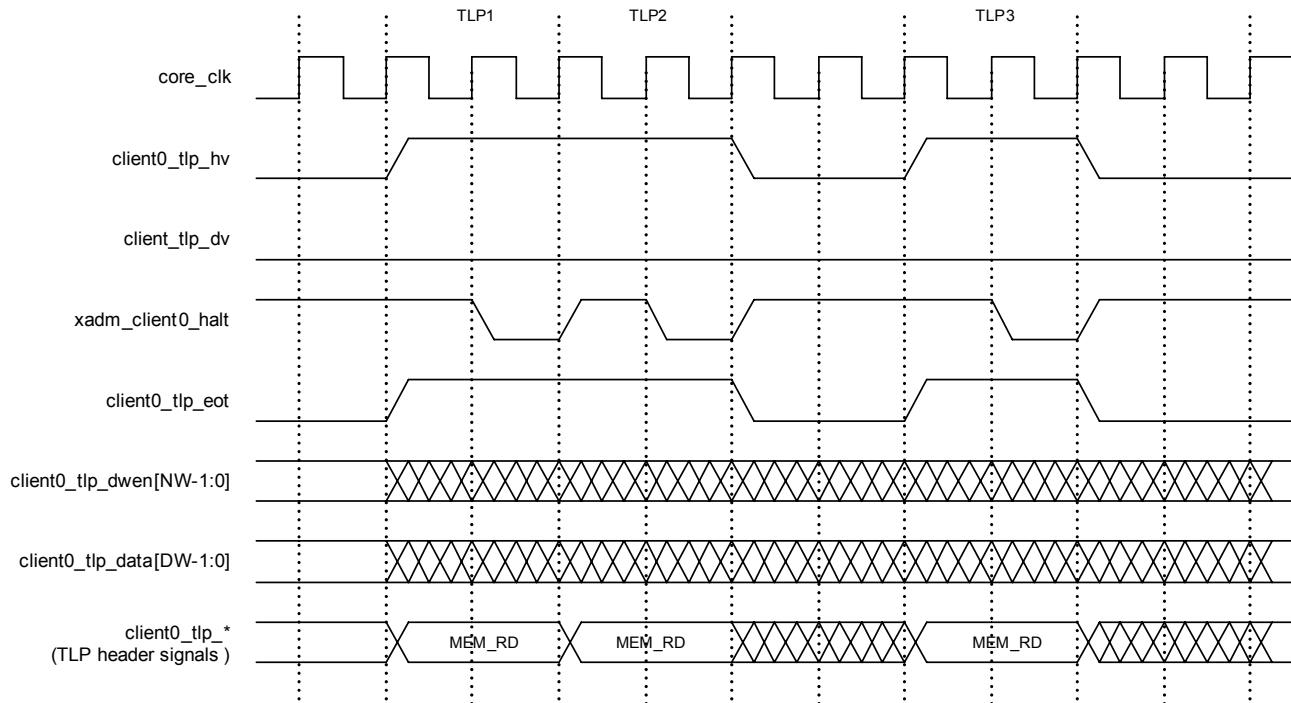
Application logic can assert input signals on XALI0, XALI1, and/or XALI2 on the same cycle; the core will arbitrate between them.

3.2.5 Example Transmit Client Transactions

The following diagrams illustrate example transactions on the transmit client interfaces:

- ❖ [Figure 3-7](#) depicts 3 Memory Read (MRd) Request TLPs. `client0_tlp_eot` and `client0_tlp_hv` remain asserted for two back-to-back MRd TLPs. The TLP header inputs remain valid until `xadm_client0_halt` is deasserted. `xadm_client0_halt` is asserted when the XADM is not ready to process any transactions.
- ❖ [Figure 3-8](#) depicts a Memory Write (MWr) Request TLP. The TLP header and payload inputs remain unchanged when `xadm_client0_halt` is asserted.
- ❖ [Figure 3-9](#) depicts back-to-back MRd and MWr transactions. The `client0_tlp_eot` and `client0_tlp_hv` signals remain asserted across multiple TLPs.
- ❖ [Figure 3-10](#) depicts a Completion TLP requested by Client0.

Figure 3-7 Client0 Transaction: MRd Request TLPs



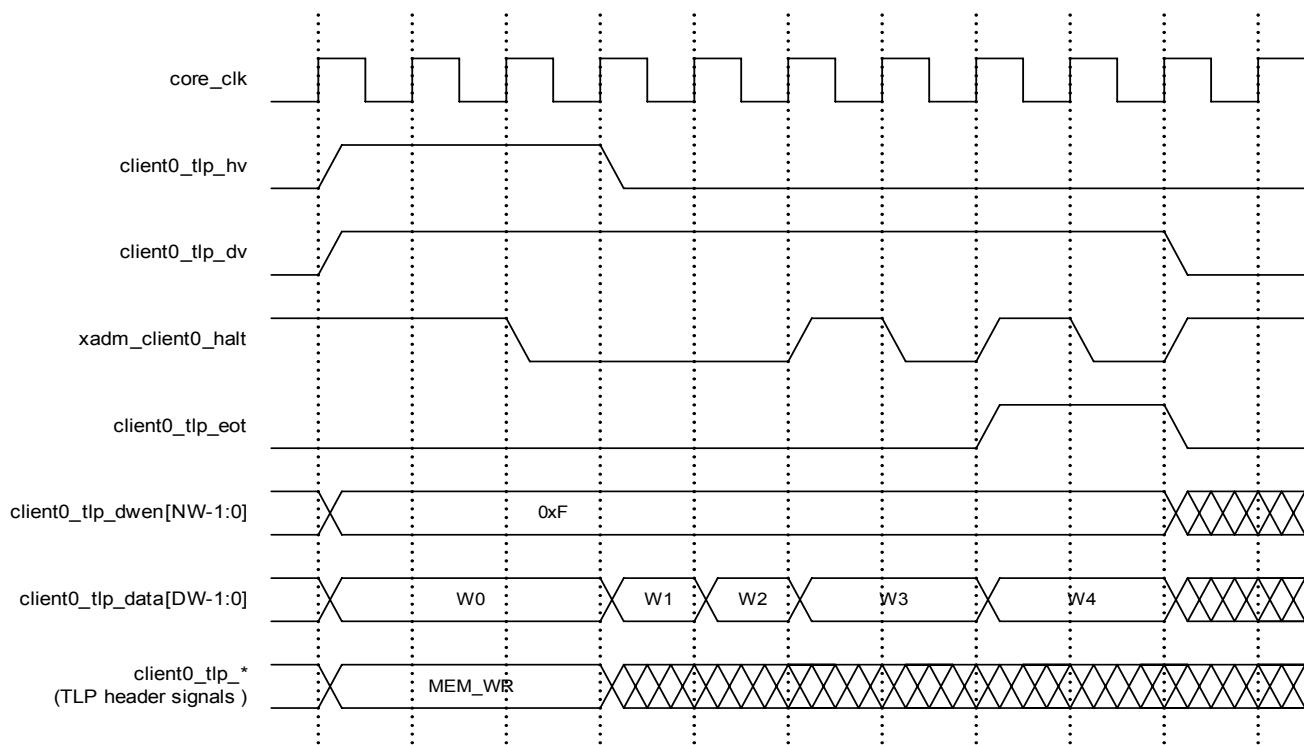
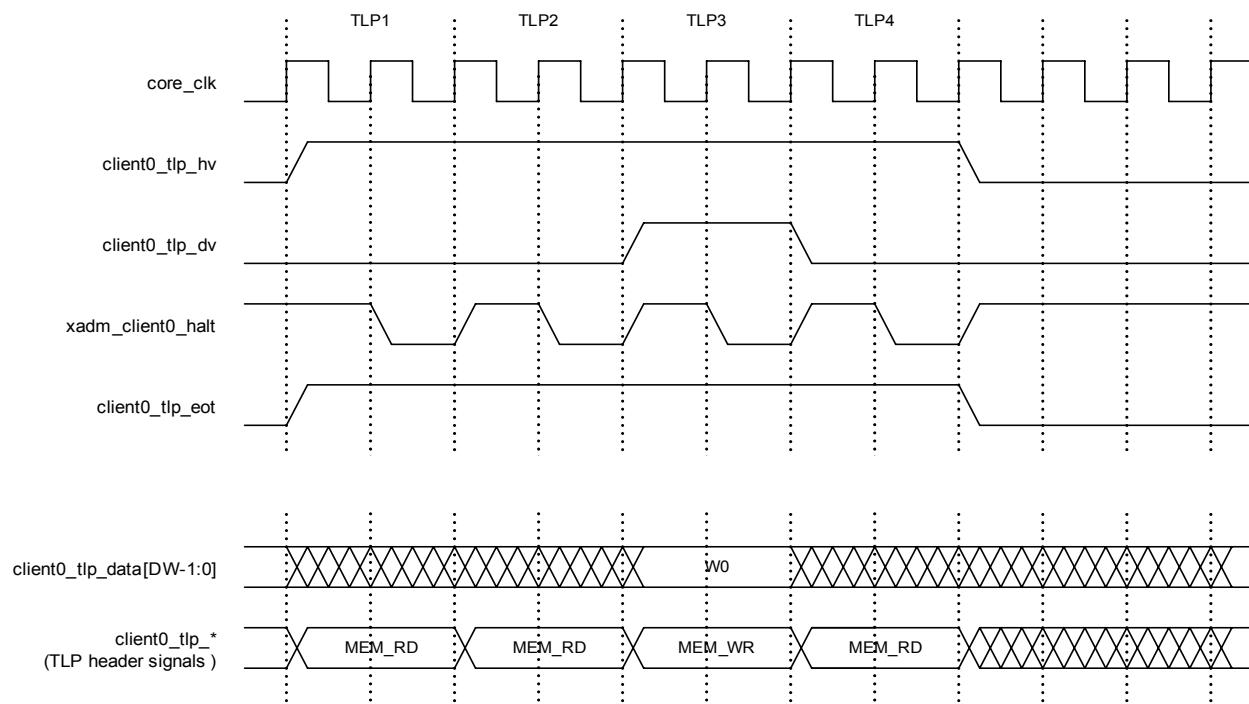
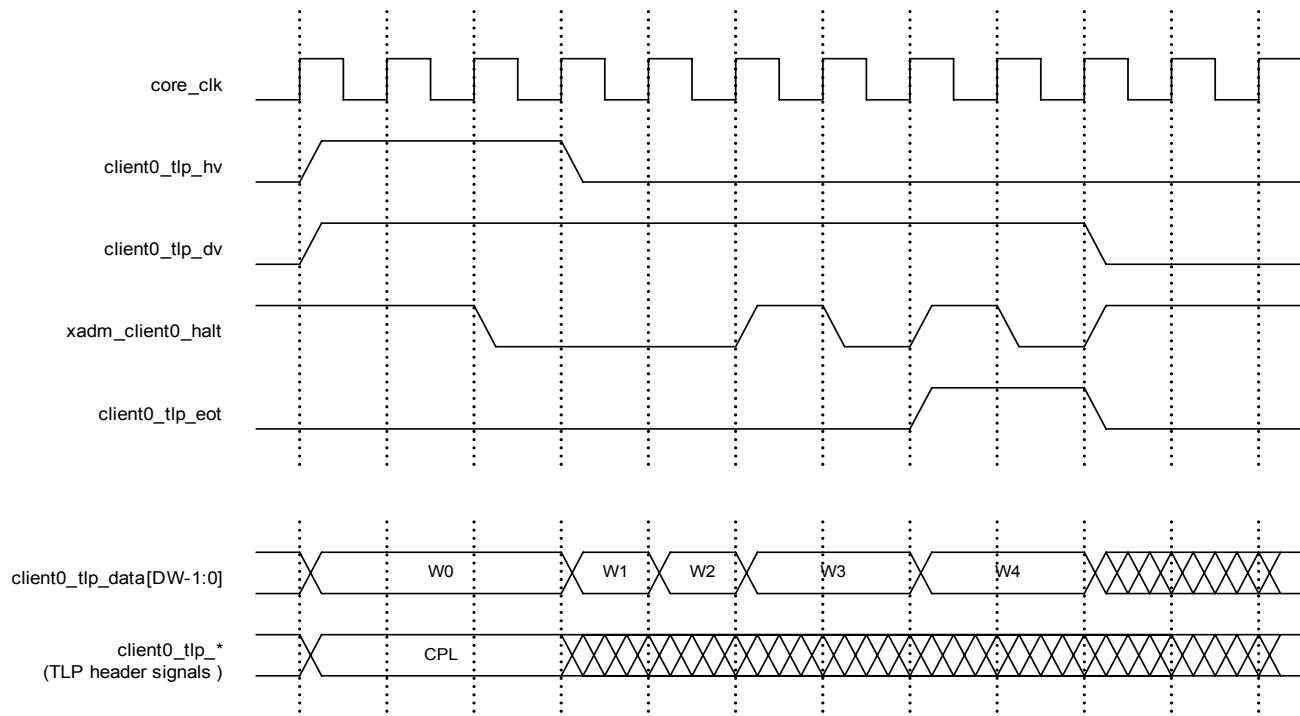
**Figure 3-8 Client0 Transaction: MWr Request TLP****Figure 3-9 Client0 Transaction: Back-to-Back MRd and MWr TLPs**

Figure 3-10 Client0 Transaction: Completion TLP



3.2.6 Transmit Address Alignment

For the transmission of Memory and I/O Requests from the client interface(s), the core provides an address alignment feature so the application can have per-packet control of how the address is aligned on the transmit client interface. This feature is only available if you choose the Enable Address Alignment (^GLOB_ADDR_ALIGN_EN) configuration option.

If you choose the Enable Address Alignment option, then dynamic (runtime) control of transmit client address alignment on a per-packet basis is through the client0/1/2_addr_align_en signal.

- ❖ When address alignment is disabled for a packet (clientN_addr_align_en = 0), the application is responsible for presenting the Request to the client interface in the format in which the packet will be transmitted. This includes setting the appropriate byte enable values and presenting any included data as it will appear in the transmitted TLP.
- ❖ When address alignment is enabled (clientN_addr_align_en = 1), the core uses the lower 2 bits of the clientN_tlp_addr bus to determine the appropriate byte enable values and the alignment of any data included for the transmitted TLP. Also, when the core performs address alignment on a transmitted TLP, the transmitted TLP will always have contiguous byte enables.

3.2.6.1 Address Alignment Disabled

When address alignment is disabled, the transmit client interface essentially becomes a DWORD interface for that packet. In this case, the application must provide the TLP header exactly as it is to be inserted in the outgoing TLP. Using Client0 as the example, the requirements are:

- ❖ client0_tlp_addr[1:0]: Must be 0; the address must be DWORD-aligned.
- ❖ client0_tlp_byte_en[7:0]: The byte enables are placed directly into the transmitted TLP. Bits [3:0] form the First BE field; bits [7:4] form the Last BE Field. Non-contiguous byte enables can be transmitted in this way.

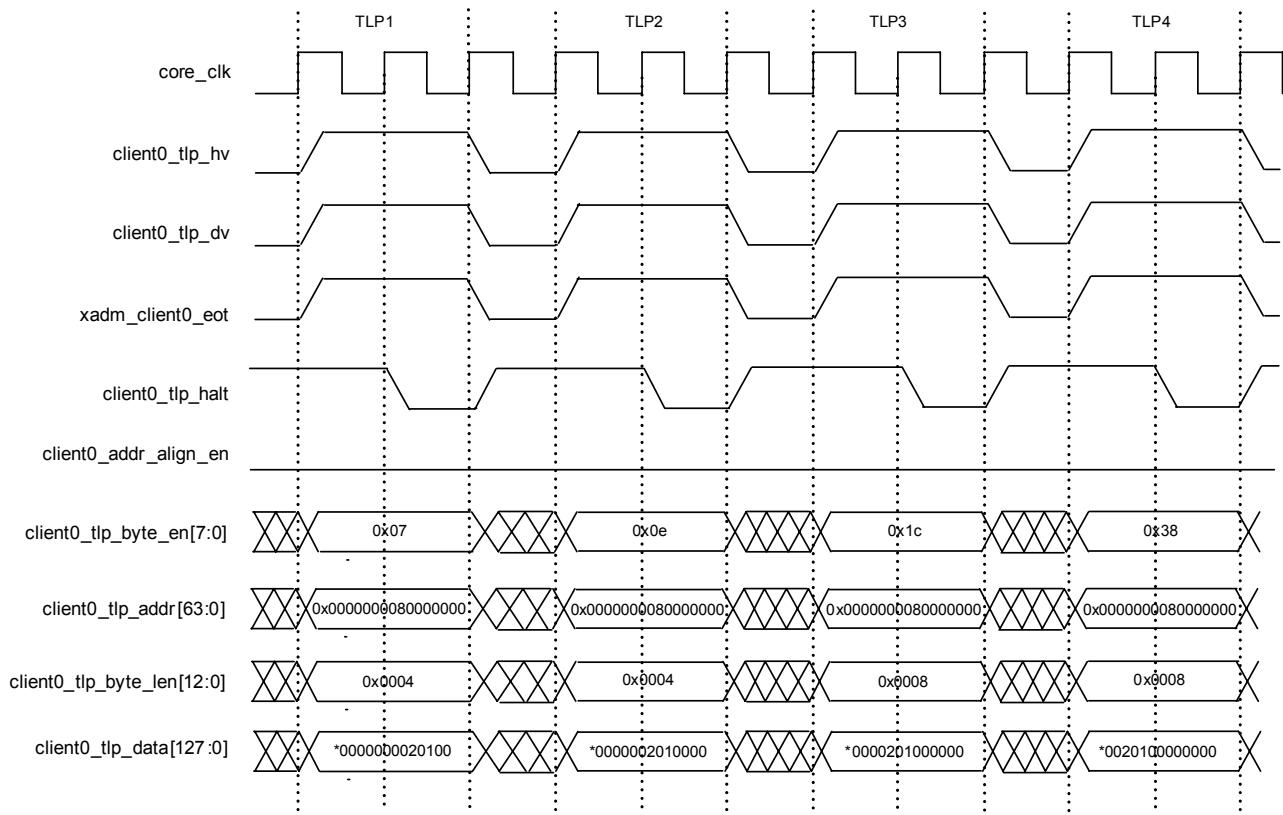


Note According to the *PCI Express 2.0 specification*, only certain TLP lengths can have non-contiguous byte enables.

- ❖ client0_tlp_byte_len[12:0]: Bits [1:0] must be set to 0. The byte length value must be rounded up to the next DWORD boundary. For example, if the Request is 5 bytes long, the value of client0_tlp_byte_len[12:0] must be 8.
- ❖ client0_tlp_data: The core transmits the data just as it is presented on client0_tlp_data. The application is responsible for shifting the data up according to the byte address.

For Read Requests, the client0_tlp_data bus has no requirements.

Figure 3-11 shows example waveforms for the transmission of four 3-byte Memory Write Requests with incrementing byte addresses starting from 0x80000000 and ending at 0x80000003. Note that the first enabled byte is shifted up for each Request with increasing address. Also, when the address plus length crosses a DWORD boundary (third TLP in Figure 3-11), the byte length goes from 4 directly to 8.

Figure 3-11 Client0 Transaction: Address Alignment Disabled

3.2.6.2 Address Alignment Enabled

When address alignment is enabled for a packet, the client interface becomes more of a byte interface. The formatting information that the application provides is as follows:

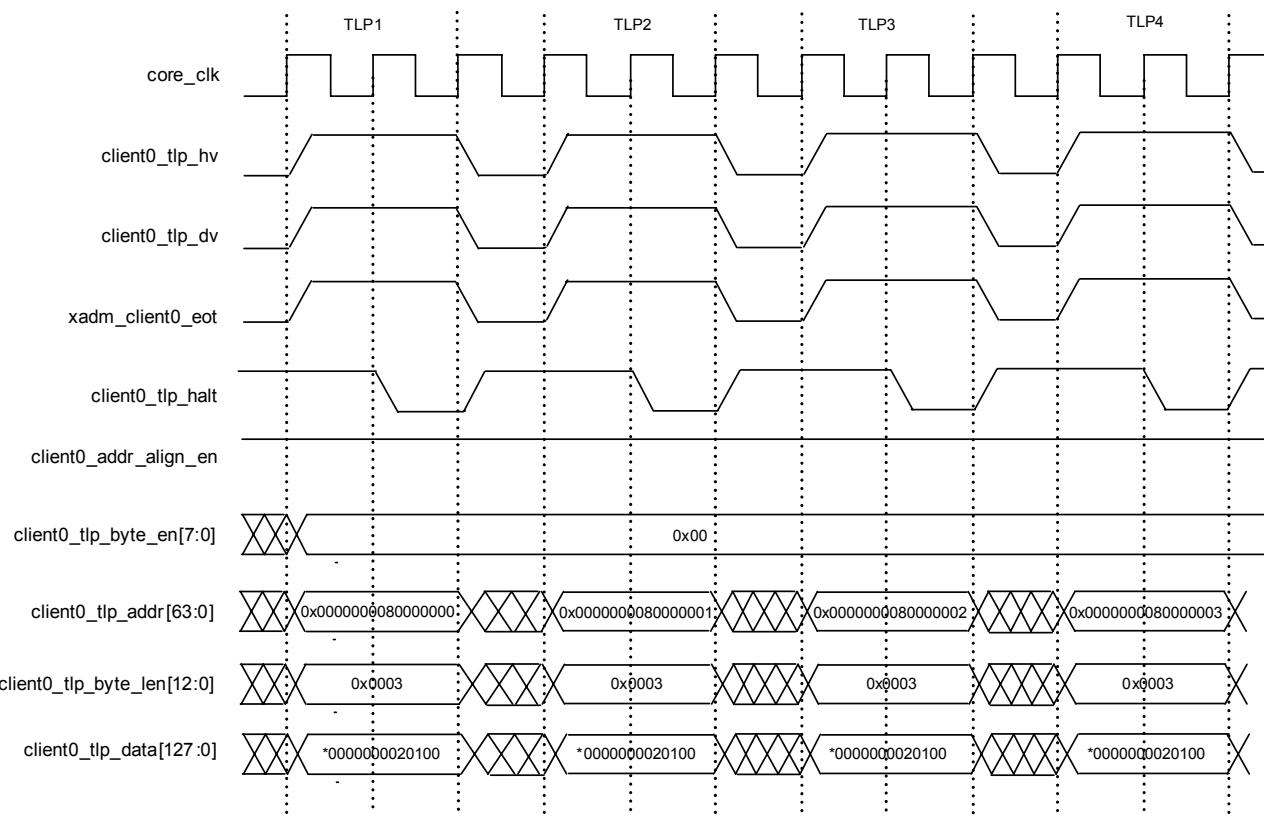
- ❖ **client0_tlp_addr**: The address is the full byte address of the first enabled byte of the Request. That is, the Request must be byte-aligned.
- ❖ **client0_tlp_byte_en**: Should be zero; not used for a packet with address alignment enabled.
- ❖ **client0_tlp_byte_len**: Must be the full byte length of the Request.
- ❖ **client0_tlp_data**: The first enabled byte of the Request must be on bits [7:0] of this data bus. The core up-shifts the data in the transmitted TLP based on the lower 2 bits of the byte address.

Based on the address and byte length, the core determines the First and Last BE fields of the TLP. For Read Requests, the **client0_tlp_data** signal has no requirements. For Write Requests, the core determines how many DWORDs of data payload will be transmitted in the associated TLP. [Figure 3-12](#) shows the same sequence of Memory Write Requests as [Figure 3-11](#), except that address alignment is enabled. The data vector does not change based on the byte address.



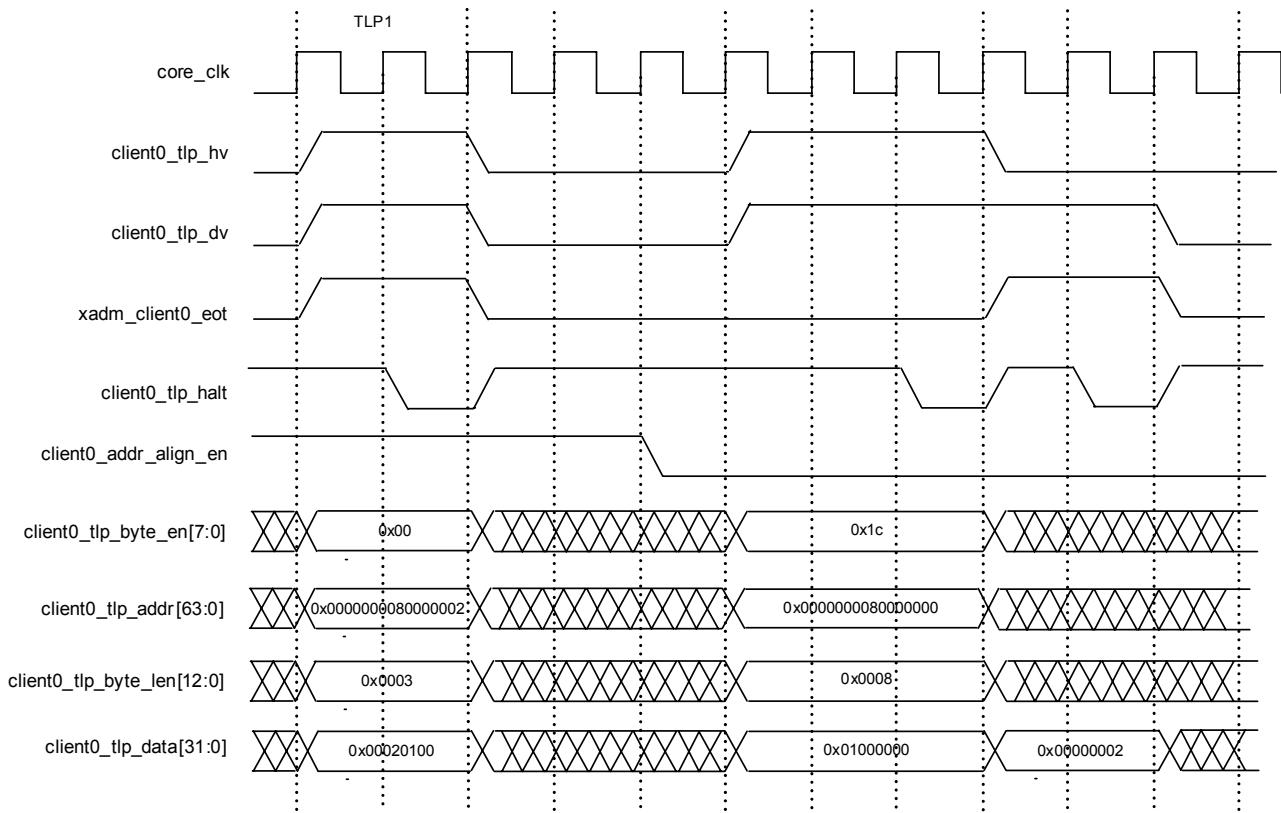
Note When address alignment is enabled, application logic must make sure that the transmitted TLP sent has a Length value less than the maximum payload supported. For example, if maximum payload is set to 128 bytes (32 DWORDs) and the application sends a request of 128 bytes but with a non-DWORD-aligned address the core will send the request with a DWORD-aligned address with Length = 33 DWORDs. This will be detected as a Malformed TLP at the destination.

Figure 3-12 Client0 Transaction: Address Alignment Enabled



3.2.6.3 Address Alignment and Data Bus Cycles

For a Write Request, the number of clock cycles on the application data bus can be greater if address alignment is disabled for that Request. This occurs when the data payload of the Write Request crosses the boundary of the data vector size. [Figure 3-13](#) shows two identical Memory Write Requests on a 32-bit transmit data bus. The byte address is 0x80000002 and the byte length is 3. The first Request has address alignment enabled and requires only one cycle on the transmit client interface. The second Request has address alignment disabled and requires two clock cycles on the transmit client interface. This has no effect on aggregate bandwidth; the only difference is where the TLP formatting is performed, by the core (first Request) or by the application (second Request).

Figure 3-13 Client0 Transaction: Address Alignment and Data Bus Cycles

3.2.6.4 Address Alignment and ECRC

If the application transmits Requests across the client interface with ECRC included, the `clientN_addr_align_en` signal must be zero for those TLPs that contain ECRC (TD bit = 1). This is because there is no way for the core to know the value of the bytes below the first enabled bytes. If the ECRC was computed over these unknown bytes, the validity of the ECRC in the transmitted TLP cannot be guaranteed if `clientN_addr_align_en` is high for that TLP.



3.3 Receive Completion Interface (RCPL) (DM / RC / EP)

The RCPL is used as a Completion interface when the core receive queues are in the multiple-buffer or single-buffer-per-VC configuration. The RCPL forwards to the application Completions that are received in response to Non-Posted Requests initiated by the transmit clients. The RADM module inside the core decodes the header of the received Completion before sending it to the application.

When the core receive queues are in the segmented-buffer configuration, the RCPL is replaced by the Bypass interface. For details about the Bypass interface, see “[Bypass Interface](#)” on page [145](#).

3.3.1 RCPL Protocol Rules

The RCPL protocol rules are:

- ❖ `radm_cpl_hv` indicates the cycle when the header information of the Completion is valid.
- ❖ `radm_cpl_dv` indicates the cycle(s) when the payload data is valid.
- ❖ A Completion begins with the assertion of `radm_cpl_hv` and ends with the assertion of `radm_cpl_eot`.
- ❖ `radm_cpl_hv` and `radm_cpl_dv` may or may not be asserted in the same cycle.
- ❖ `radm_cpl_tlp_abort` indicates that the currently received Completion has a malformed TLP error or that the Completion lookup check failed.
- ❖ `radm_cpl_dllp_abort` indicates that the currently received Completion has a Data Link Layer error or PHY error.
- ❖ Both aborts are asserted only when `radm_cpl_eot` is asserted.
- ❖ The application is not required to use all Completion header information outputs. It is up to application to use the information that it needs.



Note `radm_cpl_dllp_abort` and `radm_cpl_tlp_abort` are intended for use with the Completion queue configured in either bypass or cut-through mode so that the core can notify the application of errors detected as the TLP is received.



Note The application cannot halt TLPs on this interface when the Completion queue is configured in bypass mode. The application must provide enough buffer space to accept Completions.

3.3.2 RCPL Interface Signals

[Table 3-2](#) defines the RCPL interface signals.

Table 3-2 RCPL Interface Signals

Signal	Input/ Output	Description
cpl_radm_halt	I	<p>Function: Halts the Completion queue; must be '0' when the Completion queue is in bypass mode.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_cpl_hv	O	<p>Function: Indicates that the decoded header information is valid. The core asserts radm_cpl_hv for one clock cycle. For an Endpoint, the application must capture the header information as soon as the radm_cpl_hv is asserted because the Completion receive queue is usually bypassed due to the fact that the Endpoint typically advertises infinite credits.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_cpl_dv	O	<p>Function: Indicates valid received Completion payload data on radm_cpl_data.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_cpl_eot	O	<p>Function: Indicates the last cycle of valid payload data on radm_cpl_data for the received Completion TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: Not validated by another signal</p>

Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_tlp_abort	O	<p>Function: Indicates to the application to drop this Completion due to a malformed TLP or Completion lookup failure.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_eot is asserted. Not applicable in store-and-forward mode.</p>
radm_cpl_dllp_abort	O	<p>Function: Indicates to the application to drop this Completion due to a Data Link Layer error.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_eot is asserted. Output is not applicable in store-and-forward mode.</p>
radm_cpl_ecrc_err	O	<p>Function: Indicates an ECRC error in the received Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_eot is asserted. Not valid when Completion queue is in store and forward mode.</p>
radm_cpl_lock	O	<p>Function: Indicates that the received Completion is for a Locked Memory Read.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: Downstream ports in DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>

Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_tc[2:0]	O	<p>Function: Traffic Class (TC) field from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: Downstream ports in DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_ep	O	<p>Function: Poisoned (EP) bit from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_td	O	<p>Function: TLP Digest (TD) bit from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_attr[1:0]	O	<p>Function: Attributes field from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_len_dw[9:0]	O	<p>Function: Length field from the received Completion header, indicates length of data payload in DWORDS</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>

Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_bcm	O	<p>Function: Byte Count Modified (BCM) bit from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_byte_cnt[11:0]	O	<p>Function: Byte Count field from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_reqid[15:0]	O	<p>Function: Requester ID field from the received Completion header. The Requester ID on radm_cpl_reqid is the Requester ID sent by the initiating client.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_cplid[15:0]	O	<p>Function: Completer ID field from the received Completion header. The Completer ID on radm_cpl_cplid is the Completer ID sent by the Completer from the upstream component.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_tag[7:0]	O	<p>Function: Tag field from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>

Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_low_addr[6:0]	O	<p>Function: Lower Address field from the received Completion header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_status[2:0]	O	<p>Function: Completion Status field from the received Completion header</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_data [DW_WO_PAR-1:0]	O	<p>Function: The payload data from the received Completion TLP. The data on radm_cpl_data is in little Endian format. The first received byte is on radm_cpl_data[7:0].</p> <p>The width of radm_cpl_data depends on the data width of the core:</p> <ul style="list-style-type: none"> • For a 32-bit core, DW_WO_PAR = 32 • For a 64-bit core, DW_WO_PAR = 64 • For a 128-bit core, DW_WO_PAR = 128 <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: single/multiple buffer configuration</p> <p>Valid when: radm_cpl_dv is asserted.</p>

Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_dwen[NW-1:0]	O	<p>Function: The Data Bus DWORD Enables identify the location of the last DWORD of the TLP on the data bus (radm_cpl_data). The width and usage of radm_cpl_dwen depend on the data width of the corecore:</p> <ul style="list-style-type: none"> • For a 32-bit core, radm_cpl_dwen is not used. • For a 64-bit core, NW = 2. radm_cpl_dwen[1:0] have the following meaning: <ul style="list-style-type: none"> - 01b: The last DWORD is at radm_cpl_data[31:0]. - 11b: The last DWORD is at radm_cpl_data[63:32]. • For a 128-bit core, NW = 4. radm_cpl_dwen[3:0] have the following meaning: <ul style="list-style-type: none"> - 0001b: The last DWORD is at radm_cpl_data[31:0]. - 0011b: The last DWORD is at radm_cpl_data[63:32]. - 0111b: The last DWORD is at radm_cpl_data[95:64]. - 1111b: The last DWORD is at radm_cpl_data[127:96]. <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/RC (not SW) Populated by: single/multiple buffer configuration Valid when: radm_cpl_dv is asserted. Valid for 64 and 128-bit cores only.</p>
radm_cpl_last	O	<p>Function: Indicates the last Completion TLP of a split Completion transaction. For non-split Completion transactions, the core asserts radm_cpl_last for each Completion TLP. In either case, the core asserts radm_cpl_last in the same cycle as radm_cpl_hv.</p> <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/RC (not SW) Populated by: single/multiple buffer configuration Valid when: radm_cpl_hv is asserted</p>
radm_cpl_func_num [NFUNC_WD-1:0]	O	<p>Function: Physical function number. The core generates this signal from the Requester ID for Requests and the Completer ID for Completions.</p> <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM in EP mode / EP (not RC/SW) Populated by: single/multiple buffer configuration, 'CX_SRIOV_ENABLE' Valid when: radm_cpl_hv is asserted</p>

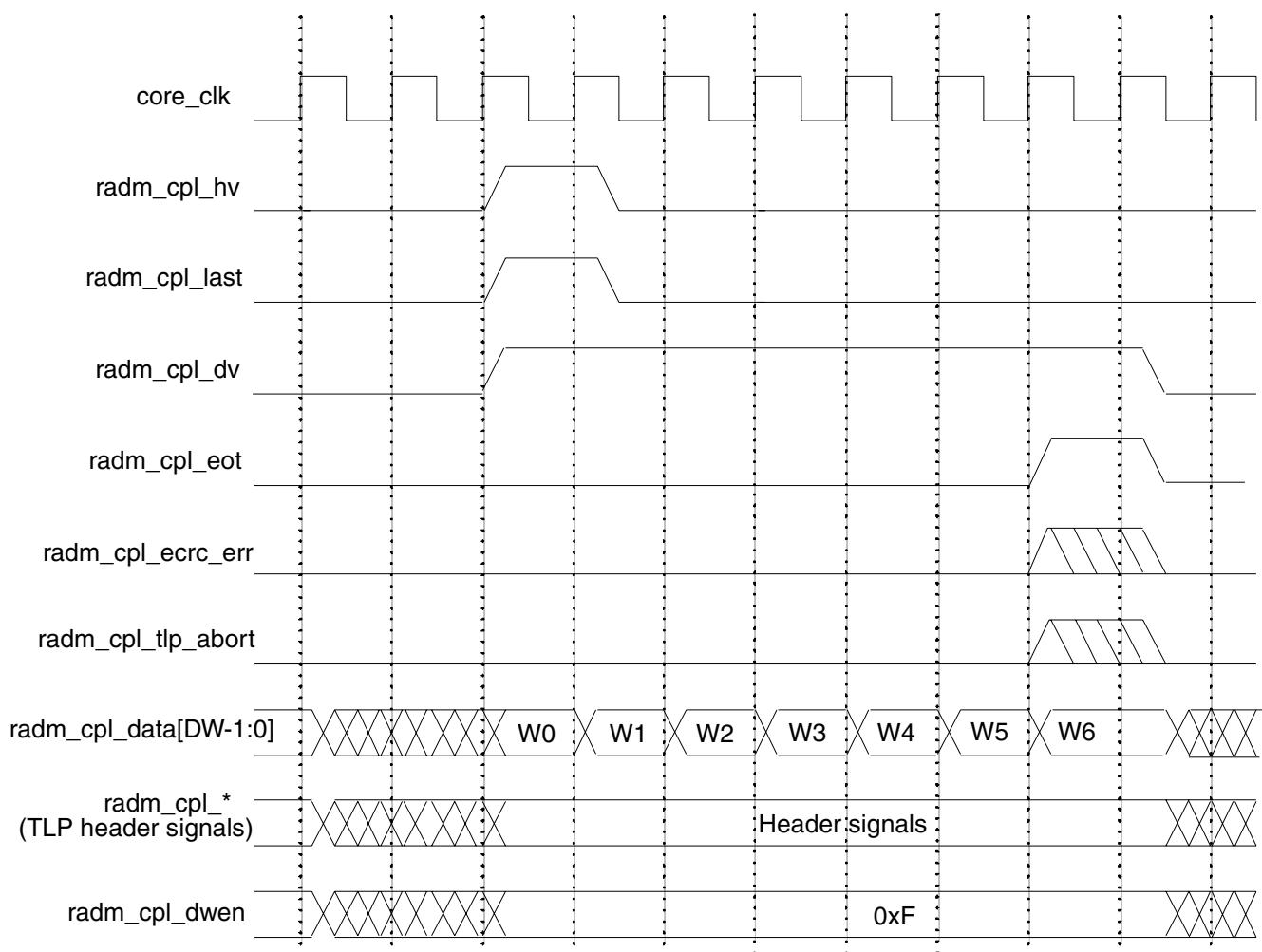
Table 3-2 RCPL Interface Signals (Continued)

Signal	Input/O utput	Description
radm_cpl_vfunc_active	O	<p>Function: TLP virtual function number valid. Indicates that there is a VF valid for this TLP.</p> <p>state 0: No VF is active and radm_cpl_vfunc_num is invalid. A PF is valid and identified by radm_cpl_func_num.</p> <p>state 1: A VF is active and is identified by radm_cpl_vfunc_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: single/multiple buffer configuration, 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_cpl_hv is asserted</p>
radm_cpl_vfunc_num [NVFUNC_NUM_WD-2:0]	O	<p>Function: Virtual function number. The core generates this signal from the Requester ID for Requests and the Completer ID for Completions.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: single/multiple buffer configuration, 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_cpl_hv and radm_cpl_vfunc_active are asserted</p>

3.3.3 Example RCPL Transactions

The following diagrams illustrate example transactions on the RCPL interface:

- ❖ [Figure 3-14](#) shows an example of a received Completion at the RCPL interface. For Endpoint applications, it is assumed that there is always buffering room at the receive end when the client sends a Memory Read Request upstream because Endpoints must advertise infinite Completion credits. Assertion of `radm_cpl_ecrc_err` indicates that the Completion TLP has a bad ECRC. Assertion of `radm_cpl_tlp_abort` indicates that the Completion TLP is malformed.
- ❖ [Figure 3-15](#) is another example of a Completion being received. However, in this example, the Lane rate is slower than the bandwidth supported at the RCPL interface. In this case `radm_cpl_data` is only valid when `radm_cpl_dv` is asserted.
- ❖ [Figure 3-16](#) shows an example in which a Completion timeout is detected. According to the *PCI Express 2.0 specification*, a transaction that requires a Completion must be completed within a specified period of time. If this time period passes without a Completion being received, then a Completion timeout occurs. The RCPL passes information about these transactions to the client.
- ❖ [Figure 3-17](#) shows an example of a split Completion. In this example, the core sends a Memory Read Request of 17 DWORDs. The target responds with a split Completion, where the first Completion TLP is 8 bytes long and the second (last) Completion TLP is 9 bytes long.

Figure 3-14 RCPL Transaction: Received Completion With Streaming Data

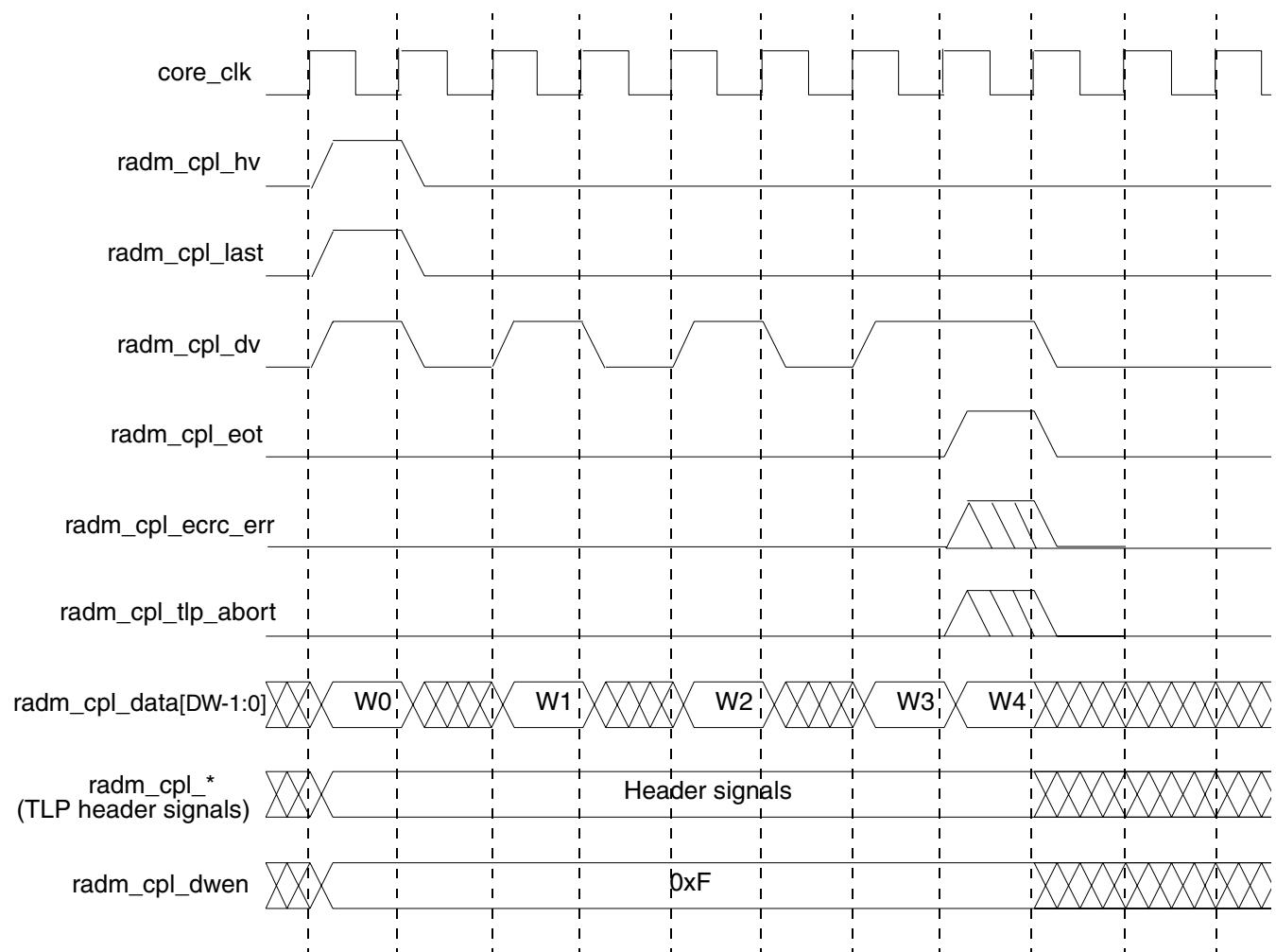
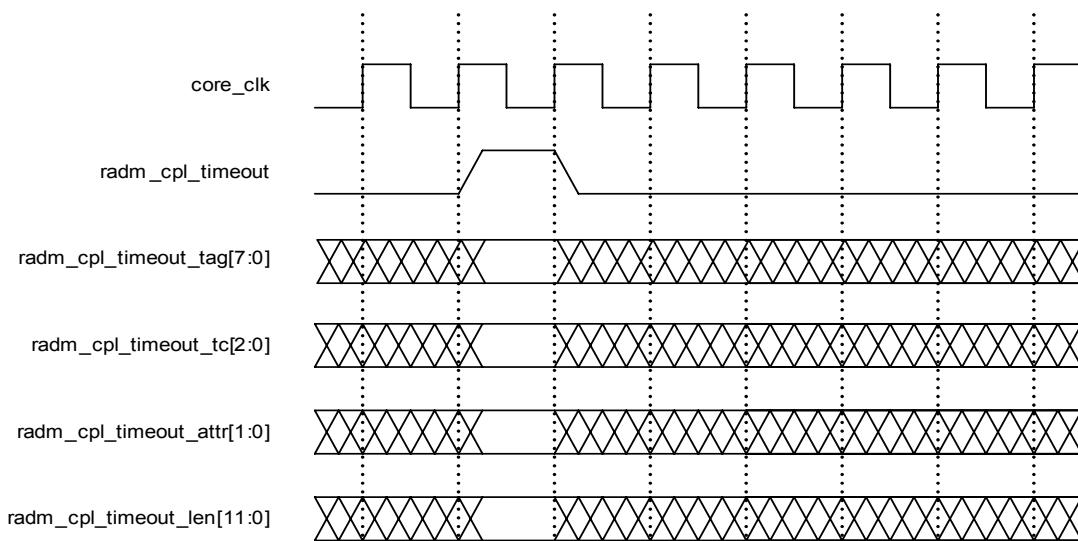
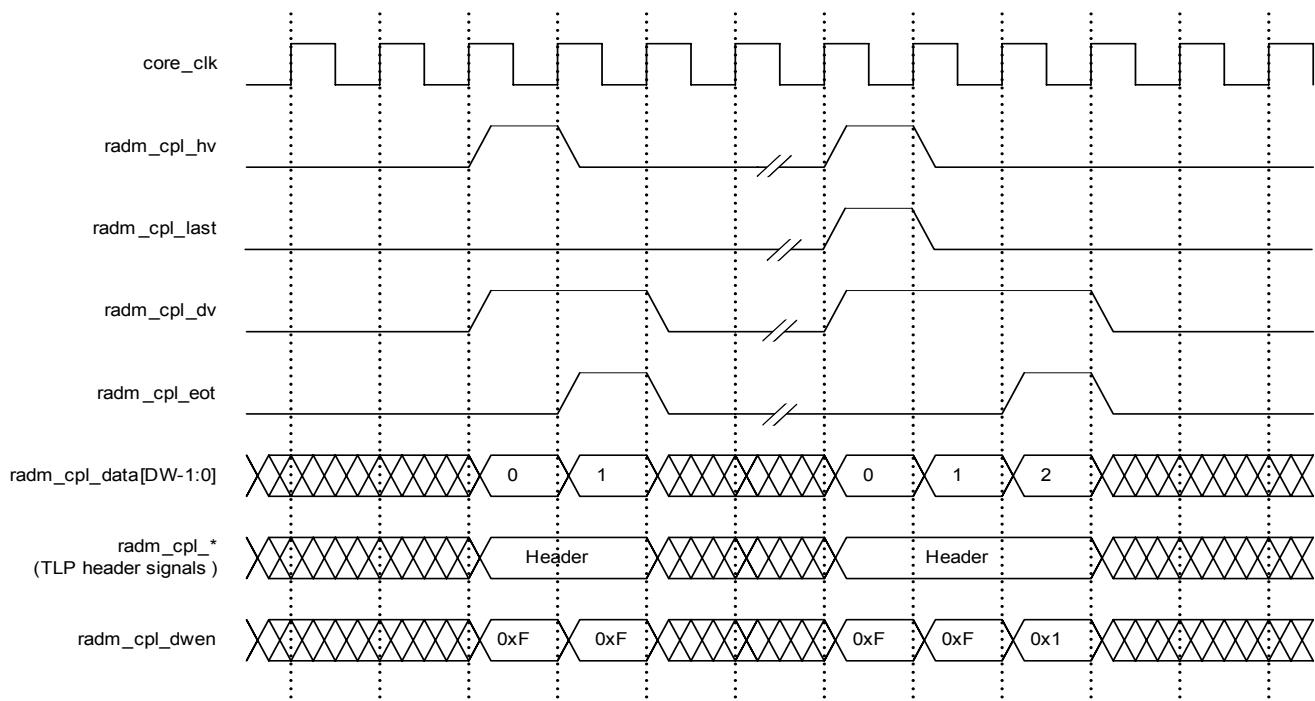
**Figure 3-15 RCLP Transaction: Received Completion With Slower Lane Rate**

Figure 3-16 RCPL Transaction: Completion Timeout**Figure 3-17 RCPL Transaction: Split Completion**



3.4 Bypass Interface

The Bypass interface replaces the RCPL when the core receive queues are in the segmented-buffer configuration. It is called bypass interface because the segmented buffer mode allows Posted, Non-Posted, or Completions in bypass mode. The core uses the Bypass interface to forward received TLPs to the application for all receive queues that are configured in bypass mode.

3.4.1 Bypass Interface Protocol Rules

The Bypass interface protocol rules are:

- ❖ `radm_bypass_hv` indicates the cycle when the header information of the TLP is valid.
- ❖ `radm_bypass_dv` indicates the cycle(s) when the payload data is valid.
- ❖ A transfer begins with the assertion of `radm_bypass_hv` and ends with the assertion of `radm_bypass_eot`.
- ❖ `radm_bypass_hv` and `radm_bypass_dv` may or may not be asserted in the same cycle.
- ❖ `radm_bypass_tlp_abort` indicates that the currently received TLP has a malformed TLP error or Completion lookup checking failed.
- ❖ `radm_bypass_dllp_abort` indicates that the currently received TLP has a Data Link Layer error or PHY error.
- ❖ Both aborts are asserted only when `radm_bypass_eot` is asserted.



Note The application cannot halt TLPs on this interface when the receive queue is configured in bypass mode. The application must provide enough buffer space to store receive transactions.

3.4.2 Bypass Interface Signals

[Table 3-3](#) defines the Bypass interface signals.

Table 3-3 Bypass Interface Signals

Signal	Input/O utput	Description
radm_bypass_hv	O	<p>Function: Indicates that the header information on radm_bypass_hdr is valid.</p> <p>The core asserts radm_bypass_hv for one clock cycle. For bypassed queues, the application must capture the header information as soon as the radm_bypass_hv is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_bypass_dv	O	<p>Function: Indicates valid received payload data on radm_bypass_data.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_bypass_eot	O	<p>Function: Indicates the last cycle of valid payload data on radm_bypass_data for the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: Not validated by another signal</p>
radm_bypass_tlp_abort	O	<p>Function: Indicates to the application to drop the TLP due to a malformed TLP or Completion lookup failure.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_eot is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_dllp_abort	O	<p>Function: Indicates to the application to drop the TLP due to a Data Link Layer error.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_eot is asserted</p>
radm_bypass_ecrc_err	O	<p>Function: Indicates an ECRC error in the received TLP. The core asserts radm_bypass_ecrc_err in the same cycle as radm_bypass_eot if an ECRC error is detected in the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_eot is asserted</p>
radm_bypass_data[DW-1:0]	O	<p>Function: The payload data from the received TLP. The data on radm_bypass_data is in little Endian format. The first received byte is on radm_bypass_data[7:0].</p> <p>The width of radm_bypass_data depends on which version of the core you are using:</p> <ul style="list-style-type: none"> • For a 32-bit core, DW = 32 • For a 64-bit core, DW = 64 • For a 128-bit core, DW = 128 <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_dv is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_dwen[NW-1:0]	O	<p>Function: The data bus DWORD enables identify the location of the last DWORD of the TLP on the data bus (radm_bypass_data). The width and usage of radm_bypass_dwen depend on which version of the core you are using:</p> <ul style="list-style-type: none"> For a 32-bit core, radm_bypass_dwen is not used. For a 64-bit core, NW = 2. radm_bypass_dwen[1:0] have the following meaning: <ul style="list-style-type: none"> 01b: The last DWORD is at radm_bypass_data[31:0]. 11b: The last DWORD is at radm_bypass_data[63:32]. For a 128-bit core, NW = 4. radm_bypass_dwen[3:0] have the following meaning: <ul style="list-style-type: none"> 0001b: The last DWORD is at radm_bypass_data[31:0]. 0011b: The last DWORD is at radm_bypass_data[63:32]. 0111b: The last DWORD is at radm_bypass_data[95:64]. 1111b: The last DWORD is at radm_bypass_data[127:96]. <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: All Populated by: segment buffer configuration Valid when: radm_bypass_dv is asserted</p>
radm_bypass_fmt[1:0]	O	<p>Function: The Format field from the received TLP header.</p> <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: All Populated by: segment buffer configuration Valid when: radm_bypass_hv is asserted</p>
radm_bypass_type[4:0]	O	<p>Function: The Type field from the received TLP header.</p> <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: All Populated by: segment buffer configuration Valid when: radm_bypass_hv is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_tc[2:0]	O	<p>Function: The Traffic Class (TC) field from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_attr[1:0]	O	<p>Function: The Attributes field from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_reqid[15:0]	O	<p>Function: The Requester ID from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_tag[7:0]	O	<p>Function: The Tag field from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_td	O	<p>Function: The TLP Digest (TD) bit from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_poisoned	O	<p>Function: The Poisoned (EP) bit from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_dw_len[9:0]	O	<p>Function: The Length field (length of TLP in DWORDs) from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_first_be[3:0]	O	<p>Function: The First DWORD Byte Enable field from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_last_be[3:0]	O	<p>Function: The Last DWORD Byte Enable field from the received TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_bcm	O	<p>Function: Byte Count Modified (BCM) bit from the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>

**Table 3-3 Bypass Interface Signals (Continued)**

Signal	Input/O utput	Description
radm_bypass_byte_cnt[11:0]	O	<p>Function: Byte Count field from the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_cpl_last	O	<p>Function: Indicates the last Completion TLP of a split Completion transaction. For non-split Completion transactions, the core asserts radm_bypass_cpl_last for each Completion TLP. In either case, the core asserts radm_bypass_cpl_last in the same cycle as radm_bypass_hv.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_cpl_status[2:0]	O	<p>Function: Completion Status field from the header of a received Completion TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_cmpltr_id[15:0]	O	<p>Function: Completer ID field from the header of a received Completion TLP. The Completer ID on radm_bypass_cmpltr_id is the Completer ID sent by the Completer from the upstream component.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>



Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_addr [(`FLT_Q_ADDR_WIDTH-1):0]	O	<p>Function: The Address from the received TLP header. The width of radm_bypass_addr (`FLT_Q_ADDR_WIDTH) is a configuration option.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_rom_in_range	O	<p>Function: Indicates that the target address in the received Request TLP is in the range defined for the expansion ROM.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_io_req_in_range	O	<p>Function: Indicates that the target address in the received Request TLP is in the I/O address range.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: radm_bypass_hv is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_in_membar_range[2:0]	O	<p>Function: Indicates which of the configured BARs contains the target address in the received Request TLP:</p> <ul style="list-style-type: none"> • 000b: Target address is in the range defined for BAR 0. • 001b: Target address is in the range defined for BAR 1. • 010b: Target address is in the range defined for BAR 2. • 011b: Target address is in the range defined for BAR 3. • 100b: Target address is in the range defined for BAR 4. • 101b: Target address is in the range defined for BAR 5. • 110b: Reserved • 111b: Target address is in not any defined BAR. <p>If <code>radm_bypass_rom_in_range = 0</code> and <code>radm_bypass_io_req_in_range = 0</code>, then <code>radm_bypass_in_membar_range</code> also indicates that the BAR associated with the target address is a memory BAR.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: <code>radm_bypass_hv</code> is asserted</p>
radm_bypass_ats[1:0]	O	<p>Function: AT field in the TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: segmented buffer configuration, 'ATS_RX_ENABLE'</p> <p>Valid when: <code>radm_bypass_hv</code> is asserted</p>
radm_bypass_func_num [NFUNC_WD-1:0]	O	<p>Function: The function number of the incoming TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: segment buffer configuration</p> <p>Valid when: <code>radm_bypass_hv</code> is asserted</p>

Table 3-3 Bypass Interface Signals (Continued)

Signal	Input/O utput	Description
radm_bypass_vfunc_active	O	<p>Function: Indicates that the received TLP was targeted to a virtual function (VF).</p> <p>state 0: No VF is active and radm_bypass_vfunc_num is invalid. A PF is valid and identified by radm_bypass_func_num.</p> <p>state 1: A VF is active and is identified by radm_bypass_vfunc_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: segmented buffer configuration, 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_bypass_hv is asserted</p>
radm_bypass_vfunc_num [NVFUNC_NUM_WD-2:0]	O	<p>Function: Indicated which virtual function (VF) was targeted by the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: segmented buffer configuration, 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_bypass_hv and radm_bypass_vfunc_active are asserted</p>



3.5 Receive Target 1 Interface (RTRGT1) (optional)

The RTRGT1 interface is an optional external interface from the core to the application client. The core uses RTRGT1 to pass incoming Requests to the application after the Requests have been filtered and routed to RTRGT1 as described in “[Receive Filtering](#)” on page 60.

When RTRGT1 is used as the default target, the application client is required to check the memory BAR range and poisoned payload bit as well as other requirements of the Completion handling mechanism, as specified in the *PCI Express 2.0 specification*.

In the segmented-buffer configuration, the core passes the Requests that pass filtering to the application on the Bypass interface instead of RTRGT1 for all queues that are operating in bypass mode.

3.5.1 RTRGT1 Protocol Rules

RTRGT1 protocol rules are:

- ❖ **EP mode:** the RTRGT1 interface is not used if the application, as a target, can only serve register Read/Write Requests, and these Requests are routed to RTRGT0 for register access on the ELBI.
- ❖ `radm_trgt1_tlp_abort` and `radm_trgt1_dllp_abort` are never asserted when the target Request queue is in store-and-forward mode.
- ❖ All header information is valid in the cycle when `radm_trgt1_hv` is asserted.



3.5.2 RTRGT1 Interface Signals

[Table 3-4](#) defines the RTRGT1 interface signals. The signals listed in [Table 3-4](#) are only present on the core if the parameter `TRGT1_POPULATE is defined.

Table 3-4 RTRGT1 Interface Signals

Signal	Input/O utput	Description
trgt1_radm_halt	I	<p>Function: Flow control input signal. When trgt1_radm_halt is asserted, the core stops streaming the next valid data from the queue inside the RADM. The data bus (radm_trgt1_data) remains unchanged while trgt1_radm_halt is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: Not validated by another signal</p>
trgt1_radm_pkt_halt[(NVC*3)-1:0]	I	<p>Function: Halts the transfer of packets from individual queues. There is one bit of trgt1_radm_pkt_halt for each TLP type for each configured VC:</p> <ul style="list-style-type: none"> • 0: Halt Posted TLPs for VC0 • 1: Halt Non-Posted TLPs for VC0 • 2: Halt Completion TLPs for VC0 • 3: Halt Posted TLPs for VC1 • . • . • up to the number of configured VCs. <p>Note: Refer to “Example RTRGT1 Transaction,” for the detailed description and timing diagrams.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: The TLP type is in not in bypass mode</p>
radm_trgt1_hv	O	<p>unction: Indicates that the header information on the RTRGT1 header information outputs is valid.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: Not validated by another signal</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_dv	O	<p>Function: Indicates that the received data (Posted and Non-Posted) is valid on radm_trgt1_data.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: Not validated by another signal</p>
radm_trgt1_eot	O	<p>Function: Indicates the last cycle of valid data for the received TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: Not validated by another signal</p>
radm_trgt1_tlp_abort	O	<p>Function: Indicates that the received TLP is malformed. The application can abort the TLP when radm_trgt1_tlp_abort is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_eot is asserted. Not applicable in store-and-forward mode.</p>
radm_trgt1_dllp_abort	O	<p>Function: Indicates that the receiver received a Data Link Layer error, asserted in the same cycle as radm_trgt1_eot. The application can abort the TLP when radm_trgt1_dllp_abort is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_eot is asserted. Not applicable in store-and-forward mode.</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_ecrc_err	O	<p>Function: Indicates that the received TLP has an ECRC error, asserted in the same cycle as radm_trgt1_eot.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_eot is asserted. Not applicable in store-and-forward mode or when ECRC errors are masked.</p>
radm_grant_tlp_type[3*NVC-1:0]	O	<p>Function: Indicates that a particular VC and type transaction has been granted to output from the receive queue. There is one bit for each TLP type for each configured VC:</p> <ul style="list-style-type: none"> • 0: Grant Posted TLPs for VC0 • 1: Grant Non-Posted TLPs for VC0 • 2: Grant Completion TLPs for VC0 • 3: Grant Posted TLPs for VC1 • . • . • up to the number of configured VCs. <p>This grant signal is a pulse. It is also a clocked output. It is used together with trgt1_radm_pkt_halt to control the amount of transactions that the core's output queue will output (see Figure 3-21 for an example).</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: The TLP type is not in bypass mode</p>
radm_trgt1_data[DW_WO_PAR-1:0]	O	<p>Function: Received TLP Request payload data (Posted and Non-Posted) from the upstream component to the application client.</p> <p>The width of radm_trgt1_data depends on the data width of the core.</p> <ul style="list-style-type: none"> • For a 32-bit core, DW_WO_PAR = 32 • For a 64-bit core, DW_WO_PAR = 64 • For a 128-bit core, DW_WO_PAR = 128 <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_dv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_fmt[1:0]	O	<p>Function: The Format field from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_type[4:0]	O	<p>Function: The Type field from the received Posted or Non-Posted Request TLP header, valid when radm_trgt1_hv is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_tc[2:0]	O	<p>Function: The Traffic Class (TC) field from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_attr[1:0]	O	<p>Function: The Attributes field from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_bcm	O	<p>Function: Byte Count Modified (BCM) bit from the header of a received Completion TLP.</p> <p>The BCM bit is not applicable for an Endpoint device. However, the core does provide the value of the BCM bit on radm_trgt1_bcm.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted and the completion is not in bypass mode</p>
radm_trgt1_reqid[15:0]	O	<p>Function: The Requester ID from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_tag[7:0]	O	<p>Function: The Tag field from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_td	O	<p>Function: The TLP Digest (TD) bit from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_poisoned	O	<p>Function: The Poisoned (EP) bit from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_dwen[NW-1:0]	O	<p>Function: Identifies the location of the last DWORD of the TLP on the data bus (radm_trgt1_data). The width and usage of radm_trgt1_dwen depend on which version of the core you are using:</p> <ul style="list-style-type: none"> For a 32-bit core, radm_trgt1_dwen is not used. For a 64-bit core, NW = 2. radm_trgt1_dwen[1:0] have the following meaning: <ul style="list-style-type: none"> 01b: The last DWORD is at radm_trgt1_data[31:0]. 11b: The last DWORD is at radm_trgt1_data[63:32]. For a 128-bit core, NW = 4. radm_trgt1_dwen[3:0] have the following meaning: <ul style="list-style-type: none"> 0001b: The last DWORD is at radm_trgt1_data[31:0]. 0011b: The last DWORD is at radm_trgt1_data[63:32]. 0111b: The last DWORD is at radm_trgt1_data[95:64]. 1111b: The last DWORD is at radm_trgt1_data[127:96]. <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_dv is asserted. Valid for 64 and 128 bit cores.</p>
radm_trgt1_dw_len[9:0]	O	<p>Function: The Length field (length of TLP in DWORDs) from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/ Output	Description
radm_trgt1_first_be[3:0]	O	<p>Function: The First DWORD Byte Enable field from the received Posted or Non-Posted Request TLP header. When a TLP is a message, radm_trgt1_first_be is overlaid with message_code[3:0]</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_last_be[3:0]	O	<p>Function: The Last DWORD Byte Enable field from the received Posted or Non-Posted Request TLP header. When a TLP is a message, radm_trgt1_last_be is overlaid with message_code[7:4]</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_addr [`FLT_Q_ADDR_WIDTH-1:0]	O	<p>Function: The Address from the received Posted or Non-Posted Request TLP header.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_rom_in_range	O	<p>Function: Indicates that the target address in the received Request TLP is in the range defined for the expansion ROM.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/ Output	Description
radm_trgt1_io_req_in_range	O	<p>Function: Indicates that the target address in the received Request TLP is in the I/O address range.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_in_membar_range[2:0]	O	<p>Function: Indicates which of the configured BARs contains the target address in the received Request TLP:</p> <ul style="list-style-type: none"> • 000b: Target address is in the range defined for BAR 0. • 001b: Target address is in the range defined for BAR 1. • 010b: Target address is in the range defined for BAR 2. • 011b: Target address is in the range defined for BAR 3. • 100b: Target address is in the range defined for BAR 4. • 101b: Target address is in the range defined for BAR 5. • 110b: Reserved • 111b: Target address is in not any defined BAR. <p>If radm_trgt1_rom_in_range = 0 and radm_trgt1_io_req_in_range = 0, then radm_trgt1_in_membar_range also indicates that the BAR associated with the target address is a memory BAR.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_func_num[NFUNC_WD-1:0]	O	<p>Function: The function number of the incoming Posted or Non-Posted Request TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/ Output	Description
radm_trgt1_ats[1:0]	O	<p>Function: AT field in the TLP header.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: `TRGT1_POPULATE and 'ATS_RX_ENABLE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_vfunc_active	O	<p>Function: Indicates that the received request matched a virtual function (VF) BAR.</p> <p>state 0: No VF is active and radm_trgt1_vfunc_num is invalid.</p> <p>A PF is valid and identified by radm_trgt1_func_num.</p> <p>state 1: A VF is active and is identified by radm_trgt1_vfunc_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: `TRGT1_POPULATE and 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_trgt1_vfunc_num [NVFUNC_NUM_WD-2:0]	O	<p>Function: Indicates which virtual function (VF) was targeted by the received request.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode / EP (not RC/SW)</p> <p>Populated by: `TRGT1_POPULATE and 'CX_SRIOV_ENABLE</p> <p>Valid when: radm_trgt1_hv and radm_trgt1_vfunc_active are asserted</p>
radm_trgt1_byte_cnt[11:0]	O	<p>Function: Byte Count field from the header of a received Completion TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted and completion is not in bypass mode</p>

Table 3-4 RTRGT1 Interface Signals (Continued)

Signal	Input/O utput	Description
radm_trgt1_cpl_last	O	<p>Function: Indicates the last Completion TLP of a split Completion transaction. For non-split Completion transactions, the core asserts radm_trgt1_cpl_last for each Completion TLP. In either case, the core asserts radm_trgt1_cpl_last in the same cycle as radm_trgt1_hv.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted and completion is not in bypass mode</p>
radm_trgt1_cpl_status[2:0]	O	<p>Function:</p> <ul style="list-style-type: none"> If the received TLP is a completion: Completion Status field from the header of a received Completion TLP. If the received TLP is a request: What the completion status for this request should be, according to internal filtering rules. <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted and completion is not in bypass mode</p>
radm_trgt1_cmplt_id[15:0]	O	<p>Function: Completer ID field from the header of a received Completion TLP. The Completer ID on radm_trgt1_cmplt_id is the Completer ID sent by the Completer from the upstream component.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `TRGT1_POPULATE</p> <p>Valid when: radm_trgt1_hv is asserted and completion is not in bypass mode</p>

3.5.3 Target Completion LUT (optional) (DM / RC / EP)

[Table 3-5](#) defines the optional target Completion lookup signals. The signals listed in [Table 3-5](#) are only present on the core if the parameter `TRGT_CPL_LUT_EN is defined.

If you select the `TRGT_CPL_LUT_EN configuration option, the trgt_lookup_id signal plus several target Completion timeout-related signals exist. The application must save the value on trgt_lookup_id and assert that value on client0_cpl_lookup_id when transmitting a Completion for the incoming Request.

Table 3-5 Optional Target Completion Lookup Table Signals

Signal	Input/O utput	Description
trgt_lookup_id[7:0]	O	<p>Function: The target Completion table lookup ID for the incoming Request TLP. When using the optional target Completion lookup table feature, the application must save the lookup ID and assert the same lookup ID on client0_cpl_lookup_id when generating a Completion for the Request.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: Not validated by another signal</p>
trgt_cpl_timeout	O	<p>Function: Indicates that the application has not generated a Completion for an incoming Request within the required time interval. Information about the timed out Completion is available on the trgt_timeout_* outputs listed below.</p> <p>When a Completion timeout occurs, the core clears the corresponding entry from the Completion lookup table.</p> <p>The default Completion timeout value is approximately 10 ms. It can be modified by changing the values of `CX_CPL_BASE_TIMER_TW, `CX_TIMEOUT_GRANULARITY, and `CX_CPL_BASE_TIMER_VALUE in adm_defs.vh.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: Not validated by another signal</p>
trgt_timeout_cpl_fun_num[2:0]	O	<p>Function: The function number of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: trgt_cpl_timeout is asserted</p>

Table 3-5 Optional Target Completion Lookup Table Signals (Continued)

Signal	Input/O utput	Description
trgt_timeout_cpl_tc[2:0]	O	<p>Function: The TC of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: trgt_cpl_timeout is asserted and completion is in store_forward segmented buffer mode</p>
trgt_timeout_cpl_attr[1:0]	O	<p>Function: The Attributes value of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: trgt_cpl_timeout is asserted</p>
trgt_timeout_cpl_len[11:0]	O	<p>Function: The Length of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: trgt_cpl_timeout is asserted</p>
trgt_lookup_empty	O	<p>Function: This signal enables the application to stop reading the received non-posted requests from the queue. When this signal is asserted with radm_trgt1_hv, it indicates that the target completion lookup table is not full. There is no valid TAG available. The application can use this signal to determine a back-pressure mechanism to the core so that there is no overflow condition when the number of outstanding non-posted transactions received is more than expected. This signal will only be valid when target completion lookup is enabled.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-5 Optional Target Completion Lookup Table Signals (Continued)

Signal	Input/O utput	Description
trgt_timeout_lookup_id[7:0]	O	<p>Function: The target Completion table lookup ID of the timed out Completion</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/RC (not SW)</p> <p>Populated by: `TRGT_CPL_LUT_EN</p> <p>Valid when: trgt_cpl_timeout is asserted</p>

3.5.4 Example RTRGT1 Transaction

Figure 3-18 shows an example when a Memory Write (MWr) TLP from upstream is received at the RTRGT1 interface. The assertion of `radm_trgt1_ecrc_err` indicates that the MWr TLP has a bad ECRC; the assertion of `radm_trgt1_abort` indicates that the MWr TLP is malformed.

Figure 3-18 RTRGT1 Transaction: MWr Request TLP

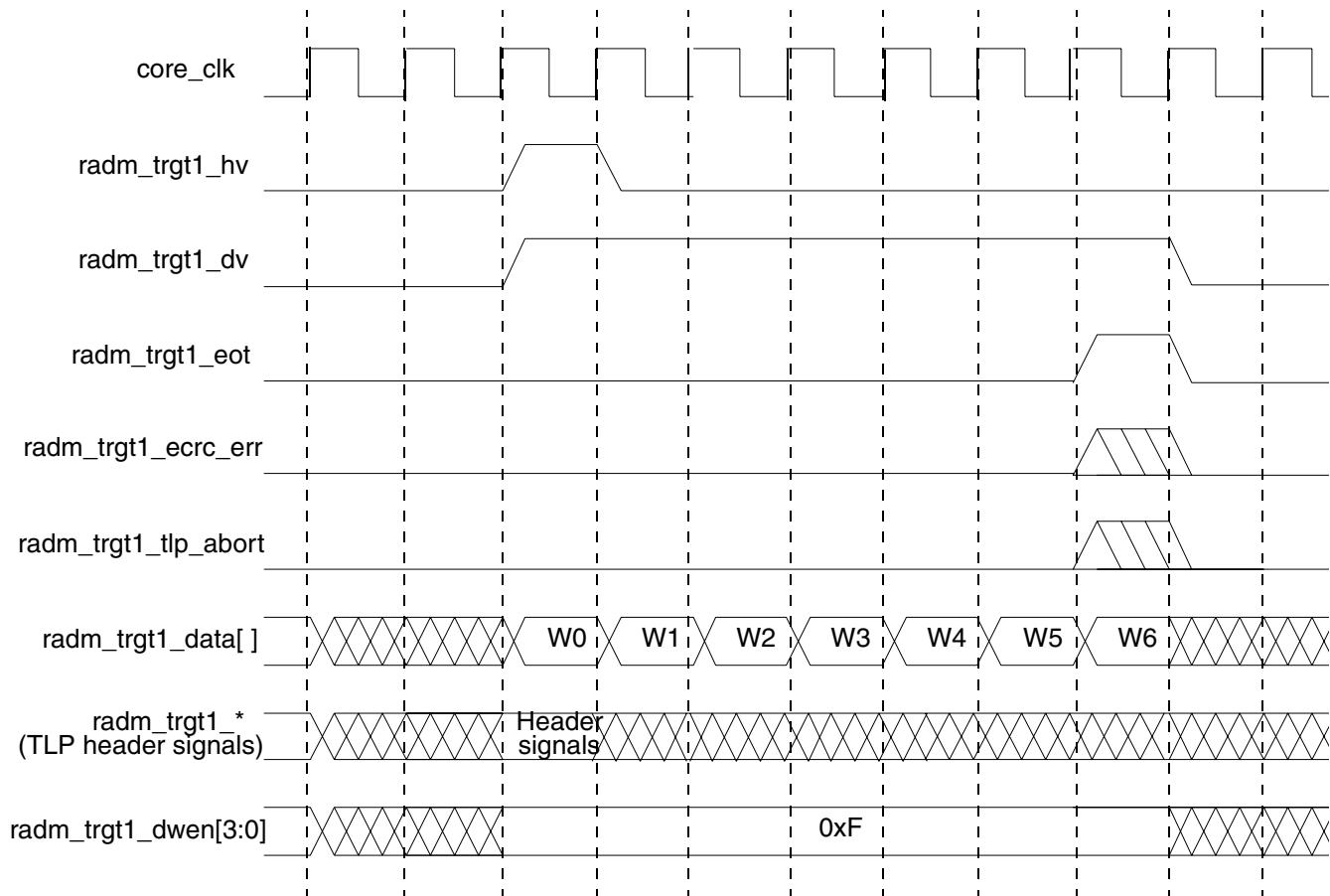
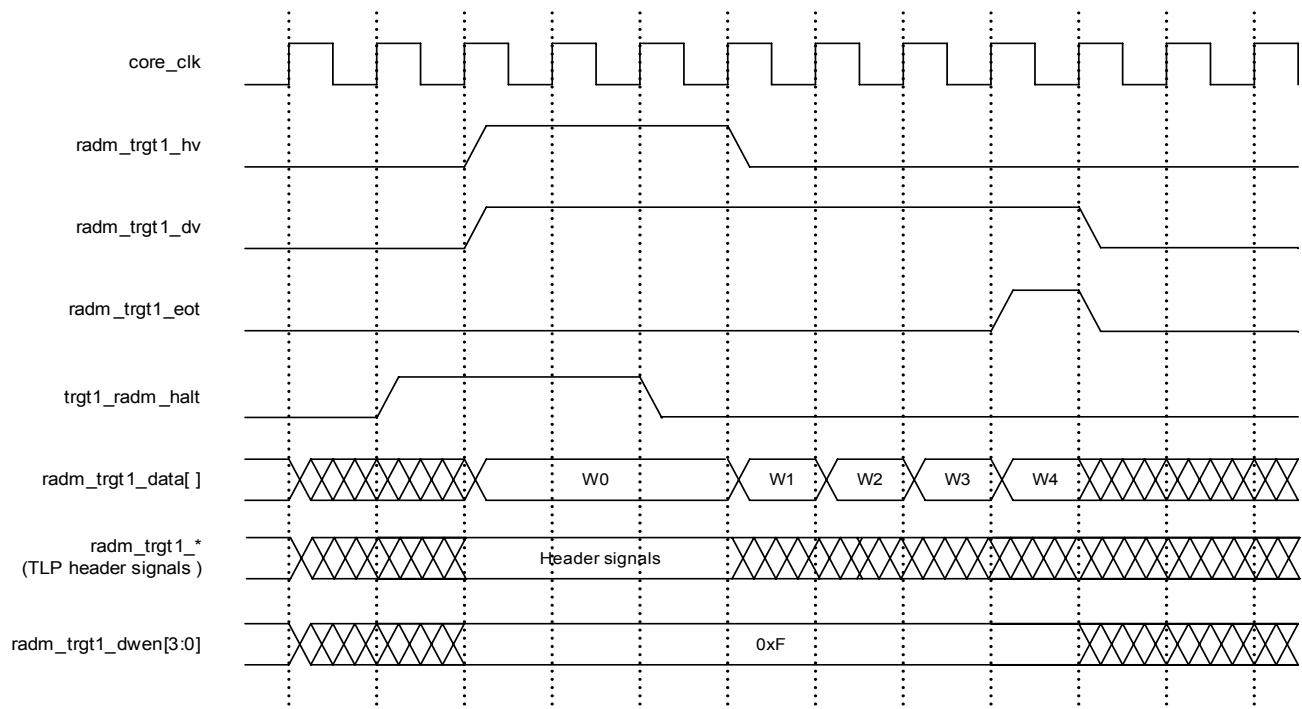


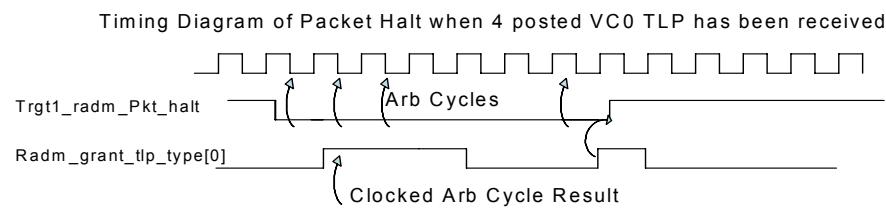
Figure 3-19 shows an example RTRGT1 transaction in which the application inserts wait states by asserting `trgt1_radm_halt`. The core begins a transaction by asserting `radm_trgt1_hv` and `radm_trgt1_dv` along with valid data and header on `radm_trgt1_data` and the header outputs, then holds all RTRGT1 outputs at their current values until the application deasserts `trgt1_radm_halt`. After sampling `trgt1_radm_halt` deasserted, the core continues the 5-DWORD transaction to completion.

Figure 3-19 RTRGT1 Transaction: MWr Request TLP with Wait States

[Figure 3-20](#) illustrates the relationship between `radm_grant_tlp_type` and `trgt1_radm_pkt_halt`. (Note that signal `radm_grant_tlp_type` only exists for segmented buffer configurations.) Output signal `radm_grant_tlp_type` allows the application to have accurate control over the number of transactions of a particular type that the application wants to receive. Signal `trgt1_radm_pkt_halt` acts like a flow control On/Off. It alone will not control the exact number of transactions that the core will output, due to the pipe lines inside the receiver queue. The pipe line depth depends on the core's configuration. Therefore, if the application wants to use `trgt1_radm_pkt_halt` to control their FIFO or queue not as a water mark, then it must monitor `radm_grant_tlp_type` for the accurate number of transactions that the application can take.

For example, the application can design a counter that monitors `radm_grant_tlp_type`. When `radm_grant_tlp_type` is asserted, the application increments the counter. When `radm_trgt1_hv` of the correct type of transaction is asserted, the application decrements the counter. When this counter reaches the number that the application has the queue size designed for, the application asserts `trgt1_radm_pkt_halt`.

Signal `radm_grant_tlp_type` is a clocked output. Signal `trgt1_radm_pkt_halt` should be asserted by the application using combinatorial logic in order to stop another transaction from being selected.

**Figure 3-20 RTRGT1 Transaction: Relationship Between radm_grant_tlp_type and trgt1_radm_pkt_halt**

3.6 External Local Bus Interface (ELBI) (DM / EP / SW)

The ELBI is an interface to access the application register block for incoming Requests that are routed to the ELBI (by way of RTRGT0). The LBC is the master that drives the ELBI.

The ELBI is used only in the DM core in EP mode, in the EP core, and in the SW core when configured as an upstream port.

3.6.1 ELBI Protocol Rules

The ELBI protocol rules are:

- ❖ Assertion of lbc_ext_cs indicates an active request cycle.
- ❖ lbc_ext_wr indicates the byte enable of a write access. All bits zero indicates a read access.

ELBI accesses are limited to one DWORD. Incoming requests targeted for the ELBI with more than one DWORD are dropped (if a write,) or returned CPL with a Completer Abort (if a read).



Note It is expected that the application only decodes the valid address bits to the BAR limit specified.

- ❖ lbc_ext_dout is only valid when lbc_ext_cs is asserted.
- ❖ lbc_ext_cs and ext_lbc_ack form a synchronous handshake. The core keeps lbc_ext_cs asserted until the application asserts ext_lbc_ack. The wait time between lbc_ext_cs and ext_lbc_ack is unlimited. The application must return an ack, otherwise, the transaction will hang. It is suggested that the application designs an error access detection circuit. When mis-access is encountered, return an ack with null data.

The originators of an ELBI access are the remote link partner and the DBI access from the application. The remote link partner can issue configuration, IO and memory type transactions to access the ELBI. The memory and IO transactions from the remote link partner are directed to the ELBI interface by the core with one exception: The PL registers can be accessed by a memory read/write from the link partner if the core is configured to have its PL registers mapped to a memory BAR. Note that this is not supported prior to release 2.8. The configuration type of transaction of the link partner can be directed to the ELBI if it contains a register address above the core's configuration register address range. The DBI can also access the CDM registers and application's registers similar to the link partner's access method. Please refer to the DBI interface section for additional information.

3.6.2 ELBI Signals

[Table 3-6](#) defines the ELBI signals.

Table 3-6 ELBI Interface Signals

Signal	Input/O utput	Description
ext_lbc_ack[NF-1:0]	I	<p>Function: Indicates that the requested read or write operation to an external register block is complete.</p> <p>There is no requirement on the time interval between the request and the assertion of ext_lbc_ack. To avoid a lockup condition in the event that the register block never asserts ext_lbc_ack, the external logic may implement a timeout mechanism.</p> <p>The width of the ext_lbc_ack signal is equal to the number of functions in your core configuration (NF). That is, there is one ext_lbc_ack bit for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
ext_lbc_din[32*NF-1:0]	I	<p>Function: Data bus from the external register block. There are 32 bits of ext_lbc_din for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: ext_lbc_ack is asserted</p>
lbc_ext_cs[NF-1:0]	O	<p>Function: The core asserts ext_lbc_cs when a received TLP for a Read or Write Request has an address in the range of the application device, as determined by the BAR configuration.</p> <p>The width of the ext_lbc_cs signal is equal to the number of functions in your core configuration (NF). That is, there is one ext_lbc_cs bit for each configured function.</p> <p>The core deasserts ext_lbc_cs only after the external register block acknowledges completion of the access by asserting the corresponding bit of ext_lbc_ack.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-6 ELBI Interface Signals (Continued)

Signal	Input/O utput	Description
lbc_ext_addr [`CX_LBC_EXT_AW-1:0]	O	<p>Function: Address bus to the external register block. The width of the address bus is the value you select for the `CX_LBC_EXT_AW parameter.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_dout[31:0]	O	<p>Function: Write data bus to the external register block, driven to all functions in a multi-function configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_wr[3:0]	O	<p>Function: Indicates if the external register access is a read or a write. For writes, lbc_ext_wr also indicates the byte enables:</p> <ul style="list-style-type: none"> • 0000b: Read • 0001b: Write byte 0 • 0010b: Write byte 1 • 0100b: Write byte 2 • 1000b: Write byte 3 • 1111b: Write all bytes <p>Combinations of byte enables (for example, 0011b) are also valid.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_rom_access	O	<p>Function: Indicates that the current ELBI access is for expansion ROM.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>

Table 3-6 ELBI Interface Signals (Continued)

Signal	Input/O utput	Description
lbc_ext_io_access	O	<p>Function: Indicates that the current ELBI access is an I/O access.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_bar_num[2:0]	O	<p>Function: The BAR number of the current ELBI access:</p> <ul style="list-style-type: none"> • 000b: BAR 0 • 001b: BAR 1 • 010b: BAR 2 • 011b: BAR 3 • 100b: BAR 4 • 101b: BAR 5 <p>110b is not used.</p> <p>111b is used to indicate a configuration access. This is designed for an application with application-specific configuration registers such as vendor capability.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_vfunc_active	O	<p>Function: Indicates that a VF is being accessed via the LBC.</p> <p>state 0: No VF is active and lbc_ext_vfunc_num is invalid. A PF is valid and identified by lbc_ext_func_num.</p> <p>state 1: A VF is active and is identified by lbc_ext_vfunc_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode /EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: lbc_ext_cs is asserted</p>
lbc_ext_vfunc_num[NVFUNC_N UM_WD-2:0]	O	<p>Function: Indicates that a VF is being accessed via the LBC.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode /EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: lbc_ext_cs and lbc_ext_vfunc_active are asserted</p>

3.6.3 Example ELBI Transactions

The following diagrams illustrate example transactions on the ELBI:

- ❖ [Figure 3-21](#) depicts an example of the LBC performing a write access to an external application register through the ELBI.
- ❖ [Figure 3-22](#) depicts an example of the LBC performing a read access to an external application register through the ELBI.

Figure 3-21 ELBI Transaction: Write Access to External Registers

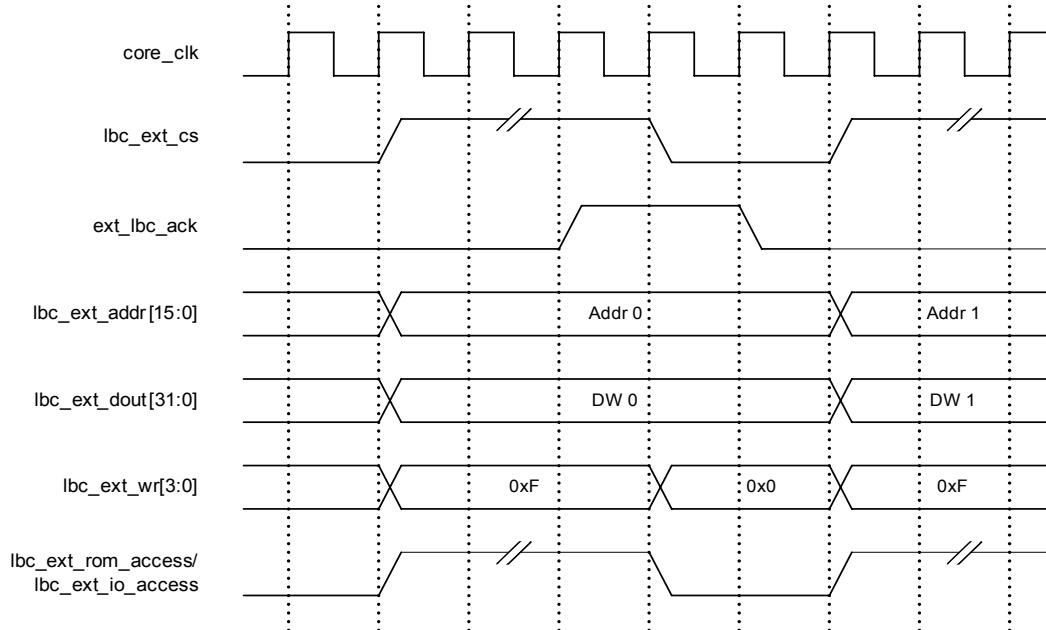
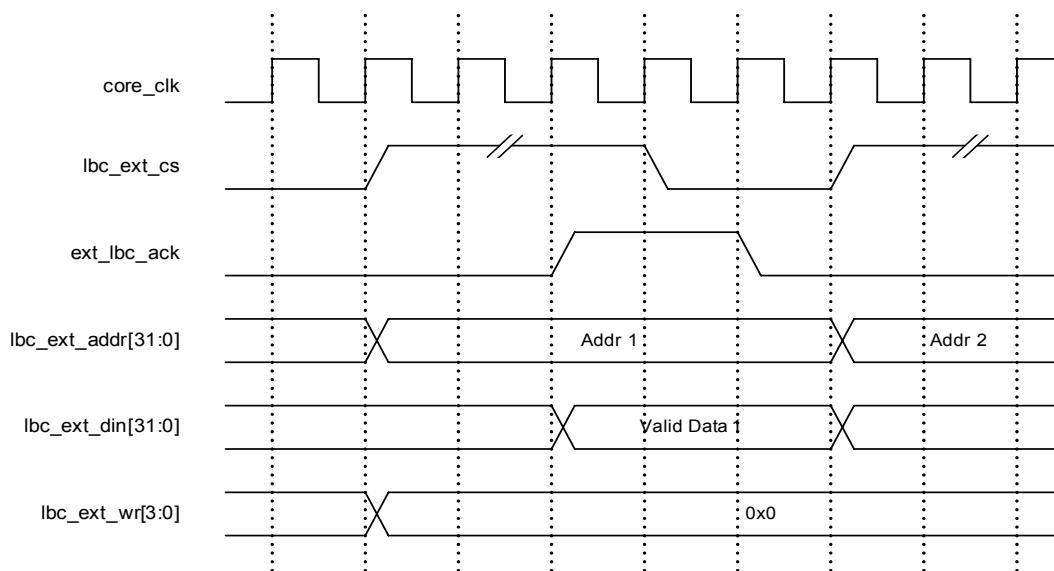


Figure 3-22 ELBI Transaction: Read Access to External Registers





3.7 Data Bus Interface (DBI)

The DBI interface is designed to enable the application to access the core's registers and external application-specific registers via the ELBI interface.

The application can access the configuration space through the DBI. Bits [11:0] of the DBI address bus select the target register. Bits [18:16] of the DBI address bus select the target physical function when not configured to support SR-IOV. When SR-IOV is supported, the physical function is indicated on the dbi_func_num and the virtual function is indicated on dbi_vfunc_num. Host software accesses the configuration registers through PCI Express Configuration Requests.

The core has two types of registers, one is the configuration space registers and the other is the memory BAR mapped registers. The application can have two types of registers as well, one for configuration space application-specific registers such as vendor capability registers, and another for memory BAR mapped registers.

dbi_addr[0] determines the access to either the core's registers or the application's registers. dbi_bar_num determines the type of access to either the core or the application's registers. A dbi_bar_num of 3'b111 indicates non-memory mapped registers.

It is required that either the core or the application respond to each DBI access. Otherwise, the DBI interface will hang.

One can use an address larger than 12 bits on dbi_addr if you access the ELBI interface instead of the Configuration Registers. You can use up to 32 bits if the core has been configured for a 32-bit ELBI address width with the parameter CX_LBC_EXT_AW.

3.7.1 DBI Protocol Rules

The DBI protocol rules are similar to the ELBI protocol rules:

- ❖ Assertion of dbi_cs indicates an active CPU cycle.
- ❖ dbi_wr indicates the byte enables of a write access. All bits zero indicates a read access.
- ❖ dbi_cs and lbc_dbi_ack form a synchronous handshake. dbi_cs must remain asserted until the core asserts lbc_dbi_ack.
- ❖ If dbi_addr[0] = 0, the access is to an internal configuration register in the CDM.
- ❖ If dbi_addr[0] = 1, the access is to an external register on the ELBI.

The DBI includes a second chip select signal (dbi_cs2). The CPU or EEPROM uses dbi_cs2 to write to a BAR Mask register, which changes the size of the corresponding BAR.

3.7.2 DBI Signals

[Table 3-7](#) defines the DBI signals.

Table 3-7 DBI Interface Signals

Signal	Input/O utput	Description
dbi_cs	I	<p>Function: Chip select input to access the configuration registers.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
dbi_addr[31:0]	I	<p>Function: Address of the configuration register for the current DBI access:</p> <ul style="list-style-type: none"> • [31:19]: Not used • [18:16]: Function number • [15:12]: Not used • [11:2]: Register address, must be DWORD-aligned. • [1]: Not used • [0]: Target of DBI access: 0 to access internal register; 1 to access external register on the ELBI <p>One can use an address larger than 12 bits on dbi_addr if you access the ELBI interface instead of the Configuration Registers. You can use up to 32 bits if the core has been configured for a 32-bit ELBI address width with the parameter CX_LBC_EXT_AW.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: lbc_ext_cs is asserted</p>
dbi_din[31:0]	I	<p>Function: Write data bus to the selected configuration register.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: dbi_cs is asserted</p>

Table 3-7 DBI Interface Signals (Continued)

Signal	Input/O utput	Description
dbi_wr[3:0]	I	<p>Function: Indicates if the configuration register access is a read or a write. For writes, dbi_wr also indicates the byte enables:</p> <ul style="list-style-type: none"> • 0000b: Read • 0001b: Write byte 0 • 0010b: Write byte 1 • 0100b: Write byte 2 • 1000b: Write byte 3 • 1111b: Write all bytes <p>Combinations of byte enables (for example, 0101b) are also valid.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: dbi_cs is asserted</p>
dbi_cs2	I	<p>Function: Additional chip select that enables writing to BAR mask registers. To write to a BAR mask register, the application must assert dbi_cs2 in addition to dbi_cs.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: dbi_cs is asserted</p>
dbi_func_num[2:0]	I	<p>Function: The Function number of the current DBI access.</p> <ul style="list-style-type: none"> • 000b: Function 0 • 001b: Function 1 • 010b: Function 2 • 011b: Function 3 • 100b: Function 4 • 101b: Function 5 • 110b: Function 6 • 111b: Function 7 <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: 'DBI_MULTI_FUNC_BAR_EN and 'CX_SRIOV_ENABLE</p> <p>Valid when: dbi_cs is asserted</p>

Table 3-7 DBI Interface Signals (Continued)

Signal	Input/O utput	Description
dbi_vfunc_active	I	<p>Function: Indicates that a VF is being accessed via the DBI.</p> <p>state 0: No VF is active and dbi_vfunc_num is invalid. A PF is valid and identified by dbi_func_num.</p> <p>state 1: A VF is active and is identified by dbi_vfunc_num.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC or SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: dbi_cs is asserted</p>
dbi_vfunc_num [NVFUNC_NUM_WD-2:0]	I	<p>Function: Indicates which VF is being accessed via the DBI.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC or SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: dbi_cs and dbi_vfunc_active are asserted</p>
dbi_rom_access	I	<p>Function: Indicates that the current DBI access is for expansion ROM.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: 'DBI_MULTI_FUNC_BAR_EN</p> <p>Valid when: dbi_cs is asserted</p>
dbi_io_access	I	<p>Function: Indicates that the current DBI access is an I/O access.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: 'DBI_MULTI_FUNC_BAR_EN</p> <p>Valid when: dbi_cs is asserted</p>

Table 3-7 DBI Interface Signals (Continued)

Signal	Input/O utput	Description
dbi_bar_num[2:0]	I	<p>Function: The BAR number of the current DBI access:</p> <ul style="list-style-type: none"> • 000b: BAR 0 • 001b: BAR 1 • 010b: BAR 2 • 011b: BAR 3 • 100b: BAR 4 • 101b: BAR 5 <p>110b is not used. 111b indicates an access to ELBI Configuration space.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP (not RC or SW)</p> <p>Populated by: 'DBI_MULTI_FUNC_BAR_EN</p> <p>Valid when: dbi_cs is asserted</p>
lbc_dbi_ack	O	<p>Function: Indicates that the requested read or write operation to the selected configuration register is complete.</p> <p>To access a non-existent register, the signal will be asserted one clock cycle, not two cycles.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
lbc_dbi_dout[31:0]	O	<p>Function: Read data bus from the selected configuration register.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: lbc_dbi_ack is asserted</p>

3.7.3 Example DBI Transactions

The following diagrams illustrate example transactions on the DBI:

- ❖ [Figure 3-23](#) depicts an example of a CPU or EEPROM performing write accesses to internal configuration space registers through the DBI.
- ❖ [Figure 3-24](#) depicts an example of a CPU or EEPROM performing read accesses to configuration space registers through the DBI.

Figure 3-23 DBI Transaction: Register Write

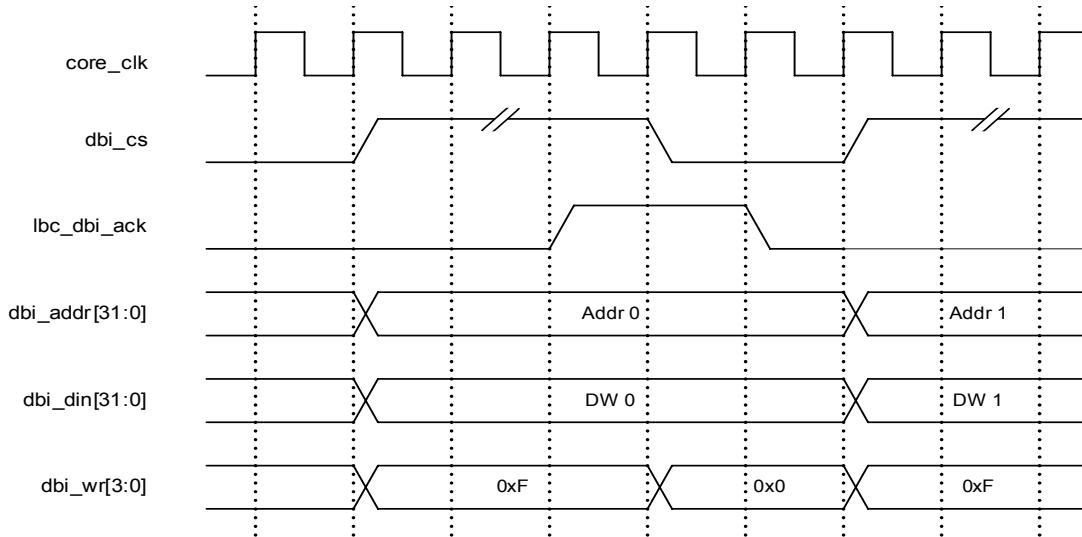
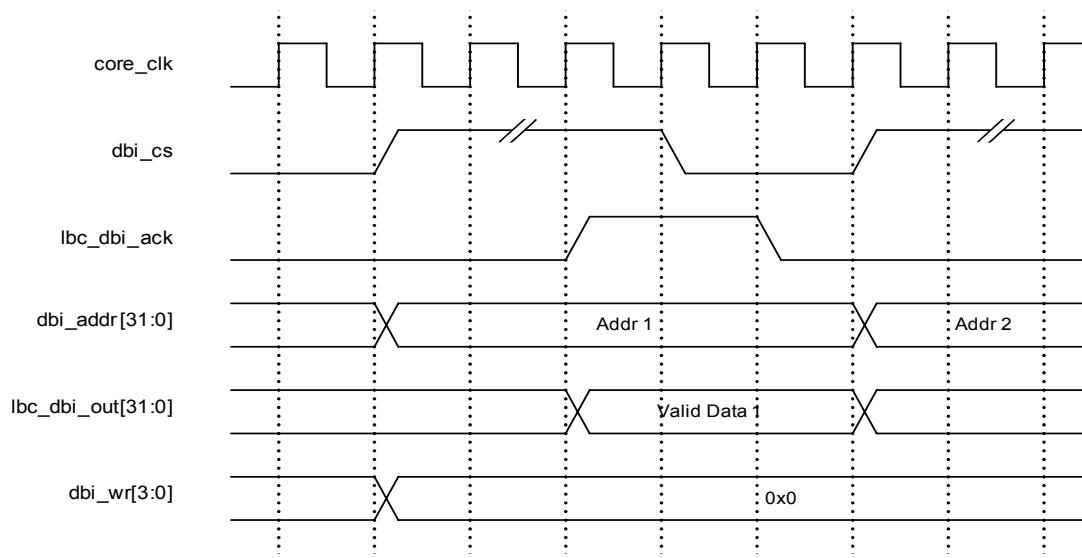


Figure 3-24 DBI Transaction: Register Read





3.8 Message Signaled Interrupt (MSI) Interface (DM / EP / SW)

The MSI interface enables the application to request the core to send an MSI. The MSI interface is used only in the DM core in EP mode, in the EP core, and in the SW core when configured as an upstream port.

The MSI interface protocol is a simple synchronous request/acknowledge handshake. When `ven_msi_req` is asserted, it must remain asserted until the core asserts `ven_msi_grant`. The MSI is requested by application logic through the MSI interface. The core then generates the corresponding Memory Write.

The following MSI interface signals are also used for MSI-X, depending on whether MSI or MSI-X is enabled: `ven_msi_req`, `ven_msi_fun_num`, `ven_msi_tc`, and `ven_msi_grant`.

PVM (Per Vector Masking) is supported with MSI.

3.8.1 MSI Interface Signals

[Table 3-8](#) defines the MSI interface signals.

Table 3-8 MSI Interface Signals

Signal	Input/Output	Description
<code>ven_msi_req</code>	I	<p>Function: Request from the application to send an MSI when MSI is enabled.</p> <p>When MSI-X is enabled instead of MSI, assertion of <code>ven_msi_req</code> causes the core to generate an MSI-X Message.</p> <p>Once asserted, <code>ven_msi_req</code> must remain asserted until the core asserts <code>ven_msi_grant</code>.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: <code>core_clk</code></p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
<code>ven_msi_fun_num[NFUNC_WD-1:0]</code>	I	<p>Function: The function number of the MSI request.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: <code>core_clk</code></p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: <code>ven_msi_req</code> is asserted</p>



Table 3-8 MSI Interface Signals (Continued)

Signal	Input/O utput	Description
ven_msi_vfun_active	I	<p>Function: Indicates that a VF is being accessed via the MSI interface.</p> <p>state 0: No VF is active and ven_msi_vfun_num is invalid. A PF is valid and identified by ven_msi_fun_num.</p> <p>state 1: A VF is active and is identified by ven_msi_vfun_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: Not validated by another signal</p>
ven_msi_vfun_num[NVFUNC_NUM_M_WD-2:0]	I	<p>Function: Indicates which VF is being accessed via the MSI interface.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: ven_msi_vfun_active is asserted</p>
cfg_msi_mask[(32*NF)-1:0]	O	<p>Function: Contents of the Per Vector Mask register in the MSI Capability structure. For each bit that is set, the function is prohibited from sending the associated message.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: MSI_PVM_EN</p> <p>Valid when: Not validated by another signal</p>
cfg_msi_pending[(32*NF)-1:0]	O	<p>Function: Contents of the Vector Interrupt Pending register in the MSI Capability structure. For each bit that is set, the function has a pending associated message.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: MSI_PVM_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-8 MSI Interface Signals (Continued)

Signal	Input/O utput	Description
ven_msi_tc[2:0]	I	<p>Function: Traffic Class of the MSI request, valid when ven_msi_req is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: ven_msi_req is asserted</p>
ven_msi_vector[4:0]	I	<p>Function: Used to modulate the lower five bits of the MSI Data register when multiple Message mode is enabled.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: ven_msi_req is asserted. Valid only when multiple Message mode is enabled for the device through the MSI Control register.</p>
ven_msi_grant	O	<p>Function: One-cycle pulse that indicates that the core has accepted the request to send an MSI. After asserting ven_msi_grant for one cycle, the core does not wait for ven_msi_req to be deasserted then reasserted to generate another MSI. If ven_msi_req remains asserted after the core asserts ven_msi_grant for one cycle, the core will generate another MSI.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_msi_en[NF-1:0]	O	<p>Function: Indicates that MSI is enabled (INTx Message will not be sent), one bit per configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-8 MSI Interface Signals (Continued)

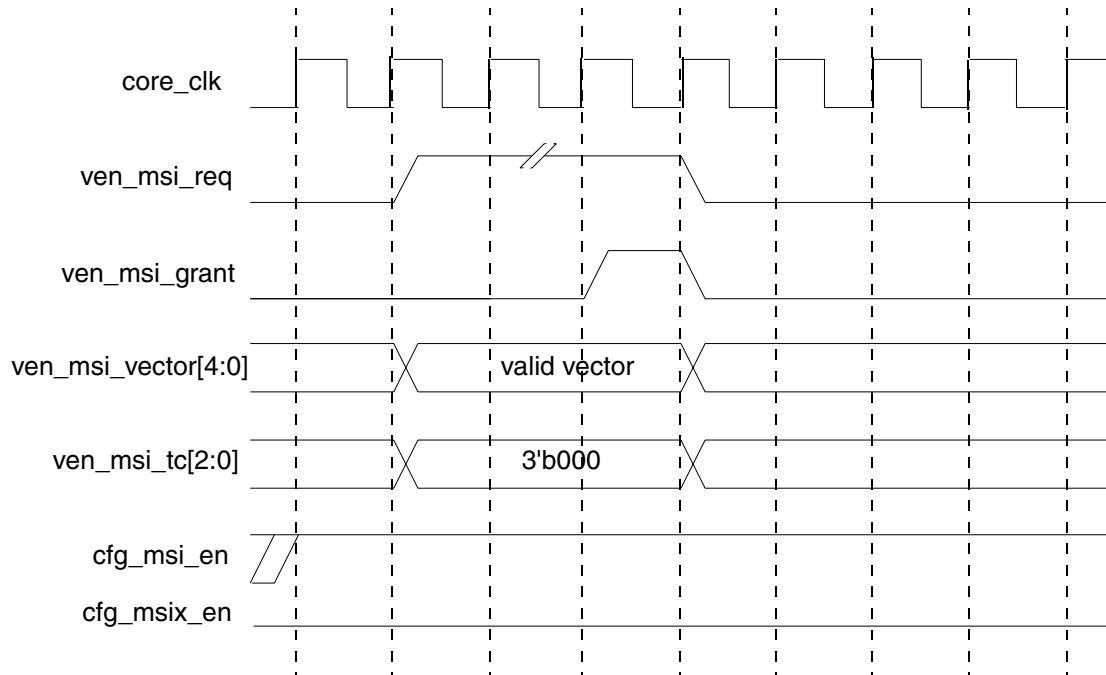
Signal	Input/O utput	Description
cfg_msi_addr[(64*NF-1):0]	O	<p>Function: Contents of the MSI Lower 32 Bits and Upper 32 Bits Address registers in the MSI Capability structure.</p> <p>There are 64 bits of cfg_msi_addr for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSI_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_msi_data[(16*NF-1):0]	O	<p>Function: Contents of the MSI Data register in the MSI Capability structure.</p> <p>There are 16 bits of cfg_msi_data for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSI_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_msi_64[NF-1:0]	O	<p>Function: The 64-bit Address Capable bit of the MSI Control register in the MSI Capability structure, one bit for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSI_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_multi_msi_en[(3*NF-1):0]	O	<p>Function: The Multiple Message Capable field of the MSI Control register in the MSI Capability structure.</p> <p>There are 3 bits of cfg_multi_msi_en for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSI_IO</p> <p>Valid when: Not validated by another signal</p>

3.8.2 Example MSI Transaction

[Figure 3-25](#) shows an example of the application client requesting the core to send an MSI Message upstream when `cfg_msi_en` is asserted. `ven_msi_req` stays asserted until the core asserts `ven_msi_grant`.

The internal MSI arbiter performs arbitration for MSI requests from different functions. `ven_msi_grant` is one-cycle pulse acknowledging `ven_msi_req`. `ven_msi_req` is not required to be deasserted before reasserting again. If `ven_msi_req` remains asserted, the core will generate another MSI.

Figure 3-25 MSI Message Request



3.9 MSI-X Interface (optional) (DM / EP / SW)

The MSI-X interface enables the application to request the core to send an MSI-X Message. The MSI-X interface is used only in the DM core in EP mode, in the EP core, and in the SW core when configured as an upstream port.

When MSI-X is enabled, the following MSI interface signals are used for MSI-X instead of MSI: `ven_msi_req`, `ven_msi_fun_num`, `ven_msi_tc`, and `ven_msi_grant`.

To request the core to send an MSI-X Message, the application uses the signals listed above in the same manner as when sending an MSI. In addition, the application must supply the MSI-X address and data on `msix_addr` and `msix_data`, respectively. Host system software stores the MSI-X address and data values in the MSI-X tables located in application memory space during system initialization.

3.9.1 MSI-X Interface Signals (optional)

Table 3-9 shows the MSI-X interface signals.

Table 3-9 MSI-X Interface Signals

Signal	Input/O utput	Description
msix_addr[63:0]	I	Function: The address value for the MSI-X. Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/SW (not RC) Populated by: `MSIX_CAP_ENABLE Valid when: ven_msi_req is asserted
msix_data[31:0]	I	Function: The data value for the MSI-X. Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/SW (not RC) Populated by: `MSIX_CAP_ENABLE Valid when: ven_msi_req is asserted
cfg_msix_en[NF-1:0]	O	Function: The MSI-X Enable bit of the MSI-X Control register in the MSI-X Capability structure. There is 1 bit of cfg_msix_en for each configured function. Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/SW (not RC) Populated by: `MSIX_CAP_ENABLE Valid when: Not validated by another signal
cfg_msix_func_mask[NF-1:0]	O	Function: The Function Mask bit of the MSI-X Control register in the MSI-X Capability structure. There is 1 bit of cfg_msix_func_mask for each configured function. Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/EP/SW (not RC) Populated by: `MSIX_CAP_ENABLE Valid when: Not validated by another signal

Table 3-9 MSI-X Interface Signals (Continued)

Signal	Input/O utput	Description
cfg_vf_msix_en[NVFUNC-1:0]	O	<p>Function: MSIX_EN bits from the Message Control Register in the MSI-X Capability register of each VF. The application needs to ensure that the corresponding bit is set before requesting to send a message for the particular VF.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: `MSIX_EN and 'CX_SRIOV_ENABLE</p> <p>Valid when: Not validated by another signal</p>
cfg_vf_msix_func_mask[NVFUNC-1:0]	O	<p>Function: Function mask bits from the Message Control Register in the MSI-X Capability register of each VF.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: `MSIX_EN and 'CX_SRIOV_ENABLE</p> <p>Valid when: Not validated by another signal</p>
cfg_vf_bme[NVFUNC-1:0]	O	<p>Function: Bus master enable bit from the Control Register in the PCI header of each VF.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE, downstream ports only</p> <p>Valid when: Not validated by another signal</p>
cfg_msix_table_size[(11*NF)-1:0]	O	<p>Function: The MSI-X Table Size field of the MSI-X Control register in the MSI-X Capability structure.</p> <p>There are 11 bits of cfg_msix_table_size for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSIX_CAP_ENABLE and `MSIX_IO</p> <p>Valid when: Not validated by another signal</p>

Table 3-9 MSI-X Interface Signals (Continued)

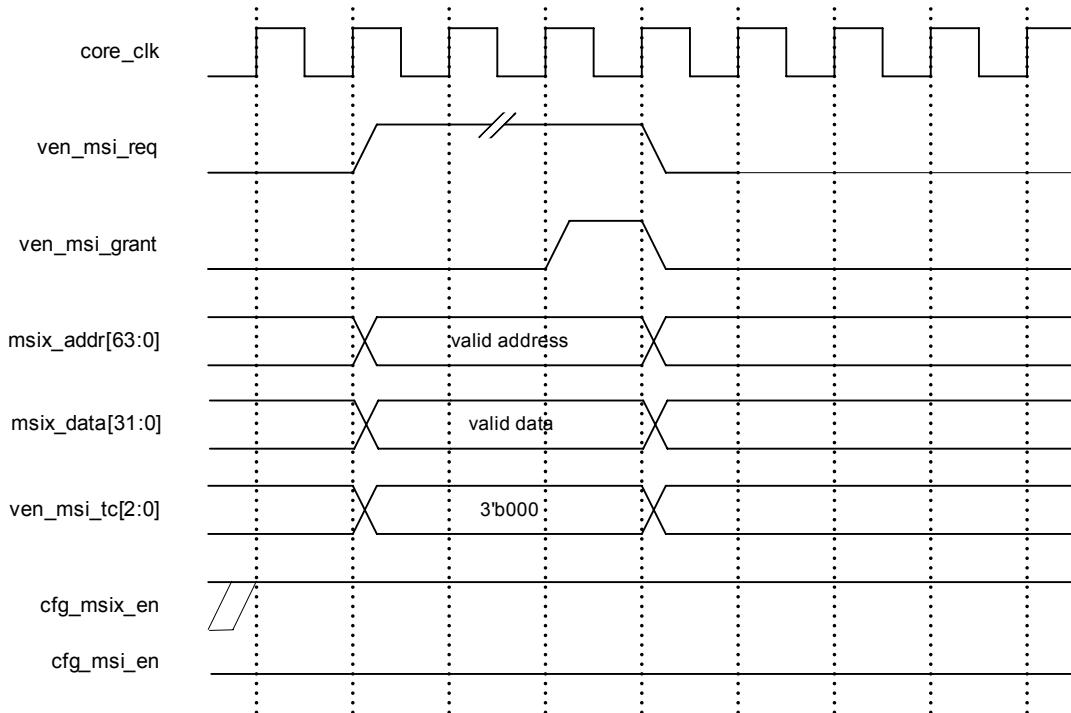
Signal	Input/O utput	Description
cfg_msix_table_bir[(3*NF)-1:0]	O	<p>Function: Table BAR Indicator Register (BIR) field of the MSI-X Table Offset and BIR register in the MSI-X Capability structure.</p> <p>There are 3 bits of cfg_msix_table_bir for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSIX_CAP_ENABLE and `MSIX_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_msix_table_offset[(29*NF)-1:0]	O	<p>Function: Table Offset field of the MSI-X Table Offset and BIR register in the MSI-X Capability structure.</p> <p>There are 29 bits of cfg_msix_table_offset for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSIX_CAP_ENABLE and `MSIX_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_msix_pba_bir[(3*NF)-1:0]	O	<p>Function: PBA BIR field of the MSI-X PBA Offset and BIR register in the MSI-X Capability structure.</p> <p>There are 3 bits of cfg_msix_pba_bir for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSIX_CAP_ENABLE and `MSIX_IO</p> <p>Valid when: Not validated by another signal</p>
cfg_msix_pba_offset[(29*NF)-1:0]	O	<p>Function: PBA Offset field of the MSI-X PBA Offset and BIR register in the MSI-X Capability structure.</p> <p>There are 29 bits of cfg_msix_pba_offset for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: `MSIX_CAP_ENABLE and `MSIX_IO</p> <p>Valid when: Not validated by another signal</p>

3.9.2 Example MSI-X Transaction

Figure 3-26 shows an example of the application requesting the core to send an MSI-X Message upstream when cfg_msix_en is asserted. ven_msi_req stays asserted until the core asserts ven_msi_grant.

As with MSI, the internal arbiter performs arbitration for MSI-X requests from different functions. ven_msi_grant is a one-cycle pulse, after which the core does not wait for ven_msi_req to be deasserted. If ven_msi_req remains asserted, the core will generate another MSI-X.

Figure 3-26 MSI-X Message Request





3.10 Vital Product Data (VPD) Support (optional)

The core supports VPD Read and Write Requests through the DBI and the cfg_vpd_int signal. This is an optional interface populated by defining VPD_IO_ENABLE.

3.10.1 Vital Product Data (VPD) Read

The sequence of events for a VPD Read cycle is:

1. The core sends a request (single-cycle pulse of the cfg_vpd_int signal) to the application to read or write vital product data.
2. The application reads the VPD Control and Capabilities register.
3. The VPD Flag bit (bit 31) is set to 1'b0, to indicate a Read request. The application fetches 4 bytes of data from the VPD Address location and transfers this to the VPD Data register.
4. The application sets the VPD Flag bit to 1'b1, to indicate the request is complete.

3.10.2 Vital Product Data (VPD) Write

The sequence of events for a VPD Write cycle is:

1. The core sends a request (single-cycle pulse of the cfg_vpd_int signal) to the application to read or write vital product data.
2. The application reads the VPD Control and Capabilities register.
3. The VPD Flag bit (bit 31) is set to 1'b1, to indicate a write request. The application reads the VPD Data register and transfers this to the location specified by the VPD Address register.
4. The application sets the VPD Flag bit to 1'b0, to indicate the request is complete.

3.11 Power Budgeting (optional)

When enabled, this feature allows the core to report the power it consumes in the device(s) and power management states in various operating conditions.

[Table 3-10](#) shows the power budgeting signals.

Table 3-10 Power Budgeting Signals

Signal	Input/O utput	Description
cfg_pwr_budget_data_reg[31:0]	I	Function: Whatever is in the Data Select Register. Active State: High Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `PWR_BUDGET_IO_ENABLE Valid when: Not validated by another signal
cfg_pwr_budget_func_num[2:0]	I	Function: Function # of data register above (because this capability is per function) Active State: High Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `PWR_BUDGET_IO_ENABLE Valid when: Not validated by another signal
cfg_pwr_budget_data_sel_reg[7:0]	O	Function: New Data Register value. Data needs to be held. Active State: High Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `PWR_BUDGET_IO_ENABLE Valid when: Not validated by another signal
cfg_pwr_budget_sel[NF-1:0]	O	Function: One cycle pulse signal indicates new data_sel_reg. Something changed the value of Data Select Register, so the application should take a look at the new value and provide the updated Data Register value corresponding to it. Active State: High Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `PWR_BUDGET_IO_ENABLE Valid when: Not validated by another signal



3.12 Vendor Message Interface (VMI)

The VMI is an interface for sending vendor-defined Messages. `ven_msg_req` must be tied to ground if the VMI is not used. This feature is enabled using the `VENDOR_MESSAGE_SUPPORT` parameter.

The VMI protocol is also a simple synchronous request/acknowledge handshake. When `ven_msg_req` is asserted, it must remain asserted until the core asserts `ven_msg_grant`. The time between `ven_msg_req` and `ven_msg_grant` is undefined. The application must deassert `ven_msg_req` on the next clock cycle after sampling `ven_msg_grant` high.

The VMI can only be used to send a header-only (no payload) vendor-defined Message (up to 4 DWORDS). To send a vendor-defined Message that includes a payload in addition to the payload contained in DWORD 4 of the Message header, the application must use one of the transmit client interfaces.

If vendor message support is not needed, a narrower (than 64 bits) `radm_trgt1_addr` address bus can be used (via the [automatic] setting of the `FLT_Q_ADDR_WIDTH` parameter). This saves gates and makes some RAMs significantly more narrow.

When `VENDOR_MESSAGE_SUPPORT` is disabled, it is expected that driver software should not issue such messages to the EP core. If software driver issues such transfer (due to legacy PCIe drivers, or drivers that are designed to handle multiple revisions of ASICs) then the message will be processed following the same filtering rules and in the same manner as if vendor messages are enabled, except that bytes 8-11 of the header will not make it to the `trgt1` interface. In other words, the application will still see an 'hv' (header valid) signal for the message and the application side needs to handle this anomaly.



The control for the actual dropping of the Vendor Message is in Filter Mask Register 2.

3.12.1 VMI Signals

[Table 3-11](#) defines the VMI signals.

Table 3-11 VMI Signals

Signal	Input/O utput	Description
<code>ven_msg_req</code>	I	<p>Function: Request from the application to send a vendor-defined Message. Once asserted, <code>ven_msg_req</code> must remain asserted until the core asserts <code>ven_msg_grant</code>.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: <code>core_clk</code></p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-11 VMI Signals (Continued)

Signal	Input/O utput	Description
ven_msg_fmt[1:0]	I	<p>Function: The Format field for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_type[4:0]	I	<p>Function: The Type field for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_tc[2:0]	I	<p>Function: The Traffic Class field for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_td	I	<p>Function: The TLP Digest (TD) bit for the vendor-defined Message TLP, valid when ven_msg_req is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_ep	I	<p>Function: The Poisoned TLP (EP) bit for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>

Table 3-11 VMI Signals (Continued)

Signal	Input/O utput	Description
ven_msg_attr[1:0]	I	<p>Function: The Attributes field for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_len[9:0]	I	<p>Function: The Length field for the vendor-defined Message TLP (indicates length of data payload in DWORDs).</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_fun_num[NFUNC_WD- 1:0]	I	<p>Function: Function Number for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_vfun_active	I	<p>Function: Indicates that a VF is accessing the VMI.</p> <p>state 0: No VF is active and ven_msg_vfun_num is invalid. A PF is valid and identified by ven_msg_fun_num.</p> <p>state 1: A VF is active and is identified by ven_msg_vfun_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE</p> <p>Valid when: ven_msg_req is asserted</p>

Table 3-11 VMI Signals (Continued)

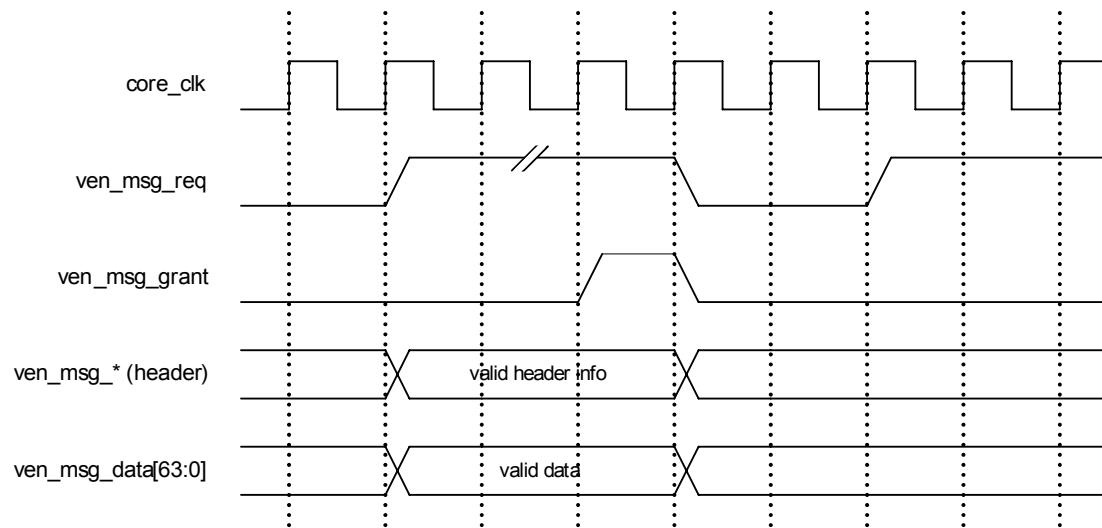
Signal	Input/O utput	Description
ven_msg_vfun_num[NVFUNC_N UM_WD-2:0]	I	<p>Function: Number of the virtual function accessing the VMI interface.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: 'CX_SRIOV_ENABLE'</p> <p>Valid when: ven_msg_req and ven_msg_vfun_active are asserted</p>
ven_msg_tag[7:0]	I	<p>Function: Tag for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_code[7:0]	I	<p>Function: The Message Code for the vendor-defined Message TLP.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_data[63:0]	I	<p>Function: DWORDs 3 and 4 (bytes 8–15, with byte 15 appearing on ven_msg_data[7:0]) of the vendor-defined Message header.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: ven_msg_req is asserted</p>
ven_msg_grant	O	<p>Function: One-cycle pulse that indicates that the core has accepted the request to send the vendor-defined Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>



3.12.2 Example VMI Transaction

Figure 3-27 shows an example where the application client requests the core to send a vendor-defined Message upstream. `ven_msg_req` stays asserted until the core asserts `ven_msg_grant`.

Figure 3-27 Vendor Message Request



3.13 System Information Interface (SII)

The SII exchanges various system-related information between the core and the application.

Most of the SII signals are provided for flexibility. The application is not required to use all of the SII signals. The application logic is expected to drive and monitor the signals that it needs to function correctly. SII inputs that the application does not require should be driven to 0.

The SII signals are grouped as follows:

- ❖ SII Signals: Overall Core Control
- ❖ SII Signals: Interrupt
- ❖ SII Signals: Power Management
- ❖ SII Signals: Electromechanical
- ❖ SII Signals: Messages
- ❖ SII Signals: Configuration Information
- ❖ SII Signals: Transmit Control
- ❖ SII Signals: Receive Control/Completion Timeout
- ❖ SII Signals: Reset Generation
- ❖ SII Signals: Debug
- ❖ SII Signals: Diagnostic and Debug Control (optional)

3.13.1 SII Signals: Overall Core Control

[Table 3-12](#) defines the SII signals related to overall control of the core.

Table 3-12 SII Signals: Overall Core Control

Signal	Input/O utput	Description
device_type[3:0]	I	<p>Function: The device type:</p> <ul style="list-style-type: none"> • 0000b: PCI Express Endpoint device • 0001b: Legacy PCI Express Endpoint device • 0100b: Root Port of PCI Express Root Complex <p>The value driven on device_type[3:0] at reset determines the operating mode of the core at run time.</p> <p>device_type 0000b or 0001b causes the DM core to operate in EP mode. device_type 0100b causes the DNM core to operate in RC mode.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/SW (not EP or RC cores)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-12 SII Signals: Overall Core Control (Continued)

Signal	Input/O utput	Description
app_ltssm_enable	I	<p>Function: Driven low by the application after reset to hold the LTSSM in the Detect state until the application is ready. When the application has finished initializing the core configuration registers, it asserts app_ltssm_enable to allow the LTSSM to continue Link establishment.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
rx_lane_flip_en	I	<p>Function: Performs Lane reversal for receive Lanes, for use when automatic Lane reversal does not occur because Lane 0 is not detected.</p> <p>For example, in a situation where an x4 core is connected to an x8 device that has its Lanes reversed, the core does not detect Lane 0 of the x8 device (only Lanes 7 down to 4) and therefore does not perform Lane reversal during Link initialization. In this case, the application can assert rx_lane_flip_en to override the default behavior and force the core to perform Lane reversal for receive Lanes.</p> <p>In most cases, rx_lane_flip_en should be wired to a static value at the chip level.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_LANE_FLIP_CTRL_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-12 SII Signals: Overall Core Control (Continued)

Signal	Input/O utput	Description
tx_lane_flip_en	I	<p>Function: Initiates Lane reversal for transmit Lanes, for use when automatic Lane reversal does not occur because Lane 0 is not detected.</p> <p>For example, in a situation where an x4 core is connected to an x8 device that has its Lanes reversed, the core does not detect Lane 0 of the x8 device (only Lanes 7 down to 4) and therefore, does not perform Lane reversal during Link initialization. In this case, the application can assert tx_lane_flip_en to override the default behavior and force the core to perform Lane reversal for transmit Lanes.</p> <p>In most cases, tx_lane_flip_en should be wired to a static value at the chip level.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_LANE_FLIP_CTRL_EN</p> <p>Valid when: Not validated by another signal</p>



3.13.2 SII Signals: Interrupt

[Table 3-13](#) defines the interrupt-related signals.

Table 3-13 SII Signals: Interrupt

Signal	Input/O utput	Description
dp_inta	I	<p>Function: INTA From Downstream Port. Indicates to an upstream port the state of the application logic's "Virtual Interrupt" wire. If legacy interrupts are enabled, a rising edge on this signal causes the upstream port to send an Assert_INTA message; a falling edge causes the port to send a Deassert_INTA message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as an upstream port</p>
dp_intb	I	<p>Function: INTB From Downstream Port. Indicates to an upstream port the state of the application logic's "Virtual Interrupt" wire. If legacy interrupts are enabled, a rising edge on this signal causes the upstream port to send an Assert_INTB message; a falling edge causes the port to send a Deassert_INTB message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as an upstream port</p>
dp_intc	I	<p>Function: INTC From Downstream Port. Indicates to an upstream port the state of the application logic's "Virtual Interrupt" wire. If legacy interrupts are enabled, a rising edge on this signal causes the upstream port to send an Assert_INTC message; a falling edge causes the port to send a Deassert_INTC message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as an upstream port</p>



Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
dp_intd	I	<p>Function: INTD From Downstream Port. Indicates to an upstream port the state of the application logic's "Virtual Interrupt" wire. If legacy interrupts are enabled, a rising edge on this signal causes the upstream port to send an Assert_INTD message; a falling edge causes the port to send a Deassert_INTD message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as an upstream port</p>
xal_xmt_cpl_ca[NF-1:0]	I	<p>Function: Transmitted Completion with CA Status. A one-clock-cycle pulse asserted when the core transmits a completion with CA status for a request it received.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
xal_rcvd_cpl_ca[NF-1:0]	I	<p>Function: Received Completion with CA Status. A one-clock-cycle pulse asserted when the core receives a completion with CA status in response to a request from the core.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
xal_rcvd_cpl_ur[NF-1:0]	I	<p>Function: Received Completion with UR Status. A one-clock-cycle pulse asserted when the core receives a completion with UR status in response to a request from the core.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
sys_int[NF-1:0]	I	<p>Function: When sys_int goes from low to high, the core generates an Assert_INTx Message. When sys_int goes from high to low, the core generates a Deassert_INTx Message.</p> <p>There is a separate sys_int input bit for each function in your core configuration. The Interrupt Pin register for the corresponding function determines which INTx Message the core generates (INTA, INTB, INTC, or INTD).</p> <p>Legacy and native PCIe devices capable of generating an interrupt must support both Assert_INTx/Deassert_INTx and MSI or MSI-X. sys_int is intended to generate a message that emulates the legacy PCI Interrupts.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_inta_asserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received an Assert_INTA Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_inta_deasserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received a Deassert_INTA Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
radm_intb_asserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received an Assert_INTB Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_intb_deasserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received a Deassert_INTB Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_intc_asserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received an Assert_INTC Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_intc_deasserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received a Deassert_INTC Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
radm_intd_asserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received an Assert_INTD Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_intd_deasserted	O	<p>Function: One-clock-cycle pulse that indicates that the core received a Deassert_INTD Message from the downstream device.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_aer_rc_err_int[NF-1:0]	O	<p>Function: The core asserts cfg_aer_rc_err_int when a reported error condition causes a bit to be set in the Root Error Status register and the associated error message reporting enable bit is set in the Root Error Command register.</p> <p>There is one bit of cfg_aer_rc_err_int for each configured function.</p> <p>Note: cfg_aer_rc_err_int is set when an internally generated error message is generated/received by the RC. Since the RC itself generates it, this needs to be propagated up to the System software which would then need to read the error registers to see which error occurred.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
cfg_aer_rc_err_msi[NF-1:0]	O	<p>Function: The core asserts cfg_aer_rc_err_msi for one clock cycle when all of the following conditions are true:</p> <ul style="list-style-type: none"> • MSI or MSI-X is enabled. • A reported error condition causes a bit to be set in the Root Error Status register. • The associated error message reporting enable bit is set in the Root Error Command register. <p>The core does not check if the associated MSI vector (asserted on cfg_aer_int_msg_num) is unmasked. It is up to the application to check whether the vector is masked or unmasked.</p> <p>There is one bit of cfg_aer_rc_err_msi for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_aer_int_msg_num[(NF*5)-1:0]	O	<p>Function: From bits [31:27] of the Root Error Status register, used when MSI or MSI-X is enabled. Assertion of cfg_aer_rc_err_msi along with a value on cfg_aer_int_msg_num is equivalent to the core receiving an MSI with the cfg_aer_int_msg_num value as the MSI vector.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
cfg_pme_int[NF-1:0]	O	<p>Function: The core asserts cfg_pme_int when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The INTx Assertion Disable bit in the Command register is 0. • The PME Interrupt Enable bit in the Root Control register is set to 1. • The PME Status bit in the Root Status register is set to 1. <p>Note: Signal cfg_pme_msi is a pulse signal (only asserted for one clock cycle). Signal cfg_pme_int is a level signal, essentially an AND of the PME interrupt enable and receipt of the pm_pme message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_pme_msi[NF-1:0]	O	<p>Function: The core asserts cfg_pme_msi (as a one-cycle pulse) when all of the following conditions are true:</p> <ul style="list-style-type: none"> • MSI or MSI-X is enabled. • The PME Interrupt Enable bit in the Root Control register is set to 1. • The PME Status bit in the Root Status register is set to 1. <p>The core does not check if the associated MSI vector (asserted on cfg_pcie_cap_int_msg_num) is unmasked. It is up to the application to check whether the vector is masked or unmasked.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
cfg_PCIE_cap_int_msg_num[(NF*5)-1:0]	O	<p>Function: From bits [13:9] of the PCI Express Capabilities register, used when MSI or MSI-X is enabled. Assertion of hp_msi or cfg_pme_msi along with a value on cfg_PCIE_cap_int_msg_num is equivalent to the core receiving an MSI with the cfg_PCIE_cap_int_msg_num value as the MSI vector.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
hp_pme[NF-1:0]	O	<p>Function: The core asserts hp_pme when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The PME Enable bit in the Power Management Control and Status register is set to 1. • Any bit in the Slot Status register transitions from 0 to 1 and the associated event notification is enabled in the Slot Control register. <p>The core does not check if the PM state is D1, D2, or D3hot. It is up to the application to check the value on pm_dstate to make sure the device is in D1, D2, or D3hot. There is one bit of hp_pme for each configured function.</p> <p>Note: hp_pme is pulsed only when any hot plug status bit changes from 0 to 1 (as is hp_msi). hp_int stays asserted as long as the status bit is set. In addition, hp_pme is asserted only if PME is enabled, but it does not matter if hot-plug interrupts are enabled.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

**Table 3-13 SII Signals: Interrupt (Continued)**

Signal	Input/O utput	Description
hp_int[NF-1:0]	O	<p>Function: The core asserts hp_int when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The INTx Assertion Disable bit in the Command register is 0. • Hot-Plug interrupts are enabled in the Slot Control register. • Any bit in the Slot Status register is equal to 1, and the associated event notification is enabled in the Slot Control register. <p>There is one bit of hp_int for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
hp_msi[NF-1:0]	O	<p>Function: The core asserts hp_msi (as a one-cycle pulse) when the logical AND of the following conditions transitions from false to true:</p> <ul style="list-style-type: none"> • MSI or MSI-X is enabled. • Hot-Plug interrupts are enabled in the Slot Control register. • Any bit in the Slot Status register transitions from 0 to 1 and the associated event notification is enabled in the Slot Control register. <p>There is one bit of hp_int for each configured function.</p> <p>Note: hp_msi is only pulsed when any of the hot plug status bits change from 0 to 1 (as is hp_pme).</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_vpd_int	O	<p>Function: This pin is set, as a one cycle pulse, to notify application to read the VPD registers.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `VPD_IO_ENABLE</p> <p>Valid when: Not validated by another signal.</p>



Table 3-13 SII Signals: Interrupt (Continued)

Signal	Input/O utput	Description
cfg_link_auto_bw_int	O	<p>Function: This pin is set as a notification when the Link Autonomous Bandwidth Status register (Link Status register bit 15) is updated and the Link Autonomous Bandwidth Interrupt Enable (Link Control register bit 11) is set. This bit is not applicable to, and is reserved, for Endpoint devices, PCI Express to PCI/PCI-X bridges, and upstream ports of Switches</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>
cfg_bw_mgt_int	O	<p>Function: This pin is set as a notification when the Link Bandwidth Management Status register (Link Status register bit 14) is updated and the Link Bandwidth Management Interrupt Enable (Link Control register bit 10) is set. This bit is not applicable to, and is reserved, for Endpoint devices, PCI Express to PCI/PCI-X bridges, and upstream ports of Switches.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>

3.13.3 SII Signals: Power Management

[Table 3-14](#) defines the SII signals that relate to power management.

Table 3-14 SII Signals: Power Management

Signal	Input/O utput	Description
outband_pwrup_cmd[NF-1:0]	I	<p>Function: Wake Up. Used by application logic to wake up the PMC state machine from the D3 state. Upon wake-up, the core sends a PM_PME Message. Needs to be asserted for one clock cycle.</p> <p>There is a separate outband_pwrup_cmd input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: aux_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
all_dwsp_in_l1	I	<p>Function: All Downstream Ports in L1. Indicates to an upstream Switch Port that all downstream Switch Ports are in the L1 state.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
all_dwsp_in_rl0s	I	<p>Function: Receive Side of All Downstream Ports in L0s. Indicates to an upstream Switch Port that the receive side of all downstream Switch Ports are in the L0s state.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>

Table 3-14 SII Signals: Power Management (Continued)

Signal	Input/O utput	Description
upsp_in_rl0s	I	<p>Function: Receive Side of Upstream Port in L0s. Indicates to a downstream Switch Port that the receive side of the upstream Switch Port is in the L1 state.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a upstream port</p>
one_dwsp_exit_l1	I	<p>Function: Downstream Port Exiting L1. Indicates to an upstream Switch Port that one or more downstream Switch Ports is exiting the L1 state.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
one_dwsp_exit_l23	I	<p>Function: Downstream Port Exiting L23. Indicates to an upstream Switch Port that one or more downstream Switch Ports is exiting the L2 or L3 state.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by:</p> <p>Valid when: Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
apps_pm_xmt_pme[NF-1:0]	I	<p>Function: Wake Up. Used by application logic to wake up the PMC state machine from the D3 state. Upon wake-up, the core sends a PM_PME Message. Needs to be asserted for one clock cycle.</p> <p>There is a separate outband_pwrup_cmd input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: aux_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-14 SII Signals: Power Management (Continued)

Signal	Input/O utput	Description
sys_aux_pwr_det	I	<p>Function: Auxiliary Power Detected. Indicates that auxiliary power (Vaux) is present.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: (Static)</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
app_req_entr_l1	I	<p>Function: Application Request to Enter L1. Request from the application to enter ASPM state L1. The app_req_entr_l1 signal is for use by applications that need to control L1 entry instead of using the L1 entry timer as defined in the <i>PCI Express 2.0 specification</i>, only effective if L1 is enabled. The core ignores the L1 entry request on app_req_entr_l1 when it is busy processing a transaction.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
app_req_exit_l1	I	<p>Function: Application Request to Exit L1. Request from the application to exit ASPM state L1, only effective if L1 is enabled.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-14 SII Signals: Power Management (Continued)

Signal	Input/O utput	Description
app_ready_entr_l23	I	<p>Function: Application Ready to Enter L23. Indication from the application that it is ready to enter the L23 state. The app_ready_entr_l23 signal is provided for applications that must control L23 entry (in case certain tasks must be performed before going into L23). The core will delay sending PM_Enter_L23 (in response to PM_Turn_Off) until this signal becomes active. Hardwire to 1 for applications that do not require this feature.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

3.13.4 SII Signals: Electromechanical

[Table 3-15](#) defines the SII signals that relate to electromechanical information.

Table 3-15 SII Signals: Electromechanical

Signal	Input/O utput	Description
sys_atten_button_pressed[NF-1:0]	I	<p>Function: Attention Button Pressed. Indicates that the system attention button was pressed, sets the Attention Button Pressed bit in the Slot Status Register.</p> <p>There is a separate sys_atten_button_pressed input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sys_pre_det_state[NF-1:0]	I	<p>Function: Presence Detect State. Indicates whether or not a card is present in the slot:</p> <ul style="list-style-type: none"> • 0: Slot is empty • 1: Card is present in the slot <p>There is a separate sys_pre_det_state input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sys_mrl_sensor_state[NF-1:0]	I	<p>Function: MRL Sensor State. Indicates the state of the manually-operated retention latch (MRL) sensor:</p> <ul style="list-style-type: none"> • 0: MRL is closed • 1: MRL is open <p>There is a separate sys_mrl_sensor_state input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-15 SII Signals: Electromechanical (Continued)

Signal	Input/O utput	Description
sys_pwr_fault_det[NF-1:0]	I	<p>Function: Power Fault Detected. Indicates the power controller detected a power fault at this slot. There is a separate sys_pwr_fault_det input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sys_mrl_sensor_chged[NF-1:0]	I	<p>Function: MRL Sensor Changed. Indicates that the state of MRL sensor has changed. There is a separate sys_mrl_sensor_chged input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sys_pre_det_chged[NF-1:0]	I	<p>Function: Presence Detect Changed. Indicates that the state of card present detector has changed. There is a separate sys_pre_det_chged input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sys_cmd_cpled_int[NF-1:0]	I	<p>Function: Command Completed Interrupt. Indicates that the Hot-Plug controller completed a command. There is a separate sys_cmd_cpled_int input bit for each function in your core configuration.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-15 SII Signals: Electromechanical (Continued)

Signal	Input/O utput	Description
sys_eml_interlock_engaged[NF-1:0]	I	<p>Function: System Electromechanical Interlock Engaged. Indicates whether the system electromechanical interlock is engaged and controls the state of the Electromechanical Interlock Status bit in the Slot Status register.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
nhp_pm_pme[NF-1:0]	I	<p>Function: Native Hot-Plug Event. Enables the application to notify the core of a native Hot-Plug event (applicable only in D1, D2, or D3hot). The core wakes up the device upon detecting a rising edge on nhp_pm_pme.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: only when device is in D1, D2 or D3hot</p>
cfg_eml_control[NF-1:0]	O	<p>Function: Electromechanical Interlock Control. The state of the Electromechanical Interlock Control bit in the Slot Control register.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
wake	O	<p>Function: Wake Up. Wake up from power management unit. The core generates wake to request the system to restore power and clock when a beacon has been detected. wake is a active high signal and its rising edge should be detected to drive the WAKE# on the connector. Assertion of wake could be a clock or multiple clock cycles.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-15 SII Signals: Electromechanical (Continued)

Signal	Input/O utput	Description
clk_req_n	O	<p>Function: Clock Enable. Allows the application clock generation module to turn off core_clk based on the current power management state:</p> <ul style="list-style-type: none"> • 1: The current power state allows core_clk to be shut down. • 0: core_clk is required to be active for the current power state. <p>clk_req_n does not indicate the clock requirement for the PHY. Clock control logic must also determine whether the PHY requires the clock to be active before shutting down the PHY clock. The core_clk should not be shut off before the PHY acknowledges the state change with phystatus.</p> <p>Active State: Low</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

3.13.5 SII Signals: Messages

Table 3-16 defines the SII signals that relate to sending and receiving Messages.

Table 3-16 SII Signals: Messages

Signal	Input/O utput	Description
all_dwsp_rcvd_toack_msg	I	<p>Function: All Downstream Ports Received PME Turnoff Acknowledge. Indicates to an upstream Switch Port that all downstream Switch Ports have received a PME_TO_Ack Message. When configured as an upstream Switch Port, the core generates a PME_TO_Ack Message to the upstream device when all_dwsp_rcvd_toack_msg is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
app_unlock_msg	I	<p>Function: Request from the application to generate an Unlock Message. To request an Unlock Message, the application asserts app_unlock_msg for one clock cycle. The core does not return an acknowledgement signal to the application.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
apps_pm_xmt_turnoff	I	<p>Function: Request from the application to generate a PM_Turn_Off Message. To request a PM_Turn_Off Message, the application asserts apps_pm_xmt_turnoff for one clock cycle. The core does not return an acknowledgement signal to the application.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-16 SII Signals: Messages (Continued)

Signal	Input/O utput	Description
cfg_sys_err_rc[NF-1:0]	O	<p>Function: System error detected. A one-clock-cycle pulse that indicates if any device in the hierarchy reports any of the following errors and the associated enable bit is set in the Root Control register: ERR_COR, ERR_FATAL, ERR_NONFATAL. Also asserted when an internal error is detected.</p> <p>There is 1 bit of cfg_sys_err_rc assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC (not EP or SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
pm_req_dwsp_turnoff	O	<p>Function: Request Downstream Port PME Turn Off. Asserted by upstream Switch Port when it has received a PME_Turn_Off Message from the upstream device. The application can either:</p> <ul style="list-style-type: none"> Respond to pm_req_dwsp_turnoff assertion by generating PME_Turn_Off Messages to all downstream Ports <p>or</p> <ul style="list-style-type: none"> Ignore pm_req_dwsp_turnoff and, instead of generating a PME_Turn_Off Message in response to pm_req_dwsp_turnoff, pass the received PME_Turn_Off Message through to the downstream Ports. <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a upstream port</p>
radm_msg_req_id[15:0]	O	<p>Function: The Requester ID of vendor-defined Message, valid when radmin_vendor_msg is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-16 SII Signals: Messages (Continued)

Signal	Input/O utput	Description
radm_correctable_err	O	<p>Function: One-clock-cycle pulse that indicates that the core received an ERR_COR Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_nonfatal_err	O	<p>Function: One-clock-cycle pulse that indicates that the core received an ERR_NONFATAL Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_fatal_err	O	<p>Function: One-clock-cycle pulse that indicates that the core received an ERR_FATAL Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_pm_pme	O	<p>Function: One-clock-cycle pulse that indicates that the core received a PM_PME Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_pm_to_ack	O	<p>Function: One-clock-cycle pulse that indicates that the core received a PME_TO_Ack Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-16 SII Signals: Messages (Continued)

Signal	Input/O utput	Description
radm_msg_unlock	O	<p>Function: One-cycle pulse that indicates that the core received an Unlock Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_vendor_msg	O	<p>Function: One-cycle pulse that indicates the core received a vendor-defined Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
radm_msg_payload[31:0]	O	<p>Function: The third DWORD of vendor-defined Message header, valid when radm_vendor_msg is asserted. To receive a Message with payload, the application must accept the Message from RTRGT1 as described in “Receive Filtering” on page 60.</p> <p>Note: This is the payload of a non-vendor or vendor-defined message or the third DW header of a vendor-defined message, not the payload.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: radm_vendor_msg is asserted</p>
radm_pm_turnoff	O	<p>Function: One-clock-cycle pulse that indicates that the core received a PME Turnoff Message.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

3.13.6 SII Signals: Configuration Information

Table 3-17 lists the signals that provide system configuration information to the application. Several of the signals are available for the application to monitor so that the application sends only valid TLPs for the current configuration. For example, the application must make sure that it only sends TLPs that do not exceed the maximum payload size because the core does not split large requests into legally-sized TLPs for transmission.

Table 3-17 SII Signals: Configuration Information

Signal	Input/O utput	Description
cfg_bar0_start[(64*NF)-1:0]	O	<p>Function: The starting address of BAR 0 (either a memory or I/O BAR). There are 64 bits of cfg_bar0_start assigned to each configured function. The actual BAR start address present on any 64-bit segment may be either a 64-bit or 32-bit BAR start address.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_bar1_start[(64*NF)-1:0]	O	<p>Function: The starting address of BAR 1 (either a memory or I/O BAR). There are 64 bits of cfg_bar1_start assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_bar0_limit[(64*NF)-1:0]	O	<p>Function: The end address of BAR 0 (either a memory or I/O BAR). There are 64 bits of cfg_bar0_limit assigned to each configured function. The actual BAR end address present on any 64-bit segment may be either a 64-bit or 32-bit BAR end address.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_bar1_limit[(64*NF)-1:0]	O	<p>Function: The end address of BAR 1 (either a memory or I/O BAR). There are 64 bits of cfg_bar1_limit assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_exp_rom_start[(32*NF)-1:0]	O	<p>Function: The starting address of expansion ROM. There are 32 bits of cfg_exp_rom_start assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_exp_rom_limit[(32*NF)-1:0]	O	<p>Function: The end address of expansion ROM. There are 32 bits of cfg_exp_rom_limit assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_bus_master_en[NF-1:0]	O	<p>Function: The state of the Bus Master Enable bit in the PCI-compatible Command register. There is 1 bit of cfg_bus_master_en assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_mem_space_en[NF-1:0]	O	<p>Function: The state of the Memory Space Enable bit in the PCI-compatible Command register. There is 1 bit of cfg_mem_space_en assigned to each configured function. Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_max_rd_req_size[(3*NF)-1:0]	O	<p>Function: The value of the Max_Read_Request_Size field in the Device Control register. There are 3 bits of cfg_max_rd_req_size assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_max_payload_size[(3*NF)-1:0]	O	<p>Function: The value of the Max_Payload_Size field in the Device Control register. There are 3 bits of cfg_max_payload_size assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_rcb[NF-1:0]	O	<p>Function: The value of the RCB bit in the Link Control register. There is 1 bit of cfg_rcb assigned to each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_pm_no_soft_rst[NF-1:0]	O	<p>Function: Indicates no reset to non-sticky registers of CDM when LTSSM link goes down.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_bridge_crs_en[NF-1:0]	O	<p>Bridge Configuration Retry Enable</p> <p>Function: Indicates the status of the Bridge Configuration Retry Enable bit in the Device Control register.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Applicable only for PCI Express to PCI(-X) bridge devices</p>
cfg_ext_tag_en[NF-1:0]	O	<p>Function: When enabled, core supports up to 8-bit tag values.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `EXT_TAG_SUPPORTED</p> <p>Valid when: Not validated by another signal</p>
cfg_pbus_num[7:0] or cfg_pbus_num[(8*NF)-1:0]	O	<p>Function: The primary bus number assigned to the function. The number of bits depends on the value of `MULTI_DEVICE_AND_BUS_PER_FUNC_EN:</p> <ul style="list-style-type: none"> If `MULTI_DEVICE_AND_BUS_PER_FUNC_EN = 0, there are 8 bits of cfg_pbus_num ([7:0]), regardless of the number of functions. If `MULTI_DEVICE_AND_BUS_PER_FUNC_EN = 1, there are 8 bits of cfg_pbus_num for each configured function. <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_2ndbus_num[(8*NF)-1:0]	O	<p>Function: Configured Secondary Bus Number. The secondary bus number assigned to the device. When the host software detects the upstream switch port, it will assign a secondary bus number. This bus number will then be used in configuration requests to detect the downstream ports. The switch core must see this so that it can route configuration requests.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
cfg_subbus_num	O	<p>Function: Configured Subordinate Bus Number. When the switch core receives a configuration request from the upstream port, it must see the secondary and subordinate bus number fields from the downstream ports to route the request.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Switch is configured as a downstream port</p>
cfg_pbus_dev_num[4:0] or cfg_pbus_dev_num[(5*NF)-1:0]	O	<p>Function: The device number assigned to the function. The number of bits depends on the value of `MULTI_DEVICE_AND_BUS_PER_FUNC_EN:</p> <ul style="list-style-type: none"> If `MULTI_DEVICE_AND_BUS_PER_FUNC_EN = 0, there are 5 bits of cfg_pbus_dev_num ([4:0]), regardless of the number of functions. If `MULTI_DEVICE_AND_BUS_PER_FUNC_EN = 1, there are 5 bits of cfg_pbus_dev_num for each configured function. <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/ Output	Description
cfg_pwr_ind[(2*NF)-1:0]	O	<p>Function: Controls the system power indicator (from bits [9:8] of the Slot Control register), per function:</p> <ul style="list-style-type: none"> • 00b: Reserved • 01b: On • 10b: Blink • 11b: Off <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/RC/SW (not EP) Populated by: Always Valid when: Not validated by another signal</p>
cfg_atten_ind[(2*NF)-1:0]	O	<p>Function: Controls the system attention indicator (from bits [7:6] of the Slot Control register), per function:</p> <ul style="list-style-type: none"> • 00b: Reserved • 01b: On • 10b: Blink • 11b: Off <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/RC/SW (not EP) Populated by: Always Valid when: Not validated by another signal</p>
cfg_pwr_ctrler_ctrl[NF-1:0]	O	<p>Function: Controls the system power controller (from bit 10 of the Slot Control register), per function:</p> <ul style="list-style-type: none"> • 0: Power On • 1: Power Off <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: DM/RC/SW (not EP) Populated by: Always Valid when: Not validated by another signal</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_mem_base[15:0]	O	<p>Function: Configured Memory Base register. The contents of the Memory Base register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_mem_limit[15:0]	O	<p>Function: Configured Memory Limit register. The contents of the Memory Limit register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_pref_mem_base[15:0]	O	<p>Function: Configured Prefetchable Memory Base register. The contents of the Prefetchable Memory Base register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_pref_mem_limit[15:0]	O	<p>Function: Configured Prefetchable Memory Limit register. The contents of the Prefetchable Memory Limit register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_io_base[7:0]	O	<p>Function: Configured I/O Base register. The contents of the I/O Base register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_io_limit[7:0]	O	<p>Function: Configured I/O Limit register. The contents of the I/O Limit register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_io_base_upper16[15:0]	O	<p>Function: Configured I/O Base Upper 16 bits. The contents of the I/O Base Upper 16 Bits register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_io_limit_upper16[15:0]	O	<p>Function: Configured I/O Limit Upper 16 bits. The contents of the I/O Limit Upper 16 Bits register from the PCIe type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_io_space_en	O	<p>Function: Configured I/O Space Enable. The contents of the I/O Space Enable bit in the PCIe Type 1 configuration space.</p> <p>Memory and I/O ranges are programmed in each downstream Switch port to route request packets.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_vc_enable[NVC-1:0]	O	<p>Function: Configured VC Enable. The state of the VC enable for each supported VC. NVC represents the number of supported VCs. VC0 is always supported and enabled.</p> <p>A multi-VC Switch would need the TC-to-VC mapping pins to route packets to the internal buffers.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>

Table 3-17 SII Signals: Configuration Information (Continued)

Signal	Input/O utput	Description
cfg_vc_struc_vc_id_map[(NVC*3)-1:0]	O	<p>Function: Configured Structural VC to VC Identifier Map. This bus provides the mapping from the structural (physical) VC numbering scheme, that is, the order the VCs are declared in the configuration space, to the logical VC identifiers that are assigned by system software. Index this bus with the structural ID and the logical ID is returned - bits [2:0] returns the ID for structural VC 0 and always equal 0. Bits [5:3] return the ID for structural VC 1, etc.</p> <p>A multi-VC Switch would need the TC-to-VC mapping pins to route packets to the internal buffers.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_vc_id_vc_struc_map[23:0]	O	<p>Function: Configured VC Identifier to Structural VC Map. This bus provides the mapping from the logical VC Identifiers that are defined by the system and the structural (physical) VC numbering scheme. Index this bus with the logical ID and the structural ID is returned - bits [2:0] returns the structural position for the VC assigned to VC 0 and always equal 0. Bits [5:3] return the structural ID (position) for the VC declared as VC 1, etc. If a given ID is not assigned, the bus returns 0.</p> <p>A multi-VC Switch would need the TC-to-VC mapping pins to route packets to the internal buffers.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_send_cor_err[NF-1:0]	O	<p>Function: Send Correctable Error. The core detected a correctable error at its receive path.</p> <p>This error signal is designed to allow the Switch to report its detect receive error from the downstream port to the upstream port.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>

**Table 3-17 SII Signals: Configuration Information (Continued)**

Signal	Input/O utput	Description
cfg_send_nf_err[NF-1:0]	O	<p>Function: Send Non-Fatal Error. The core detected a non-fatal error at its receive path.</p> <p>This error signal is designed to allow the Switch to report its detect receive error from the downstream port to the upstream port.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>
cfg_send_f_err[NF-1:0]	O	<p>Function: Send Fatal Error. The core detected a fatal error at its receive path.</p> <p>This error signal is designed to allow the Switch to report its detect receive error from the downstream port to the upstream port.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: SW only</p> <p>Populated by: Always</p> <p>Valid when: Always</p>



3.13.7 SII Signals: Transmit Control

[Table 3-18](#) defines the SII signals that relate to control of TLP transmission. The transmit controls are needed for non-default configurations. For example, pm_xtlh_block_tlp is only needed when transmit client interface blocking is disabled. (`CX_CLIENTN_BLOCK_NEW_TLP = 0`). The `*_cdts` outputs are only needed if you want the application to check available credits before submitting TLPs for transmission. The error reporting signals are only needed if the application is checking and reporting errors.

Table 3-18 SII Signals: Transmit Control

Signal	Input/O utput	Description
app_hdr_valid	I	<p>Function: One-clock-cycle pulse that indicates that the data on the app_hdr_log, app_err_bus, and app_err_func_num inputs is valid.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_ERR_EN</p> <p>Valid when: Not validated by another signal</p>
app_hdr_log[127:0]	I	<p>Function: The header of the TLP that contained the error indicated on app_err_bus, valid when app_hdr_valid is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_ERR_EN</p> <p>Valid when: app_hdr_valid is asserted</p>

Table 3-18 SII Signals: Transmit Control (Continued)

Signal	Input/O utput	Description
app_err_bus[7:0]	I	<p>Function: The type of error that the application detected, valid when app_hdr_valid is asserted. The core combines the values on the app_err_bus bits with the internally-detected error signals to set the corresponding bit in the Uncorrectable Error Status register:</p> <ul style="list-style-type: none"> • app_err_bus[0]: Malformed TLP • app_err_bus[1]: Receiver overflow • app_err_bus[2]: Unexpected Completion • app_err_bus[3]: Completer Abort • app_err_bus[4]: Completion timeout • app_err_bus[5]: Unsupported Request • app_err_bus[6]: ECRC check failed • app_err_bus[7]: Poisoned TLP received <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_ERR_EN</p> <p>Valid when: app_hdr_valid is asserted</p>
app_err_advisory	I	<p>Function: Indicates that the application error is an advisory error. The application should assert app_err_advisory under either of the following conditions:</p> <ul style="list-style-type: none"> • The core is configured to mask Completion timeout errors, the application is reporting a Completion timeout error on app_err_bus, and the application intends to resend the Request. In such cases the error is an advisory error, as described in <i>PCI Express 2.0 specification</i>. If the application does not intend to resend the Request, then the application must keep app_err_advisory deasserted when reporting a Completion timeout error. • The core is configured to forward poisoned TLPs to the application and the application is going to treat the poisoned TLP as a normal TLP, as described in <i>PCI Express 2.0 specification</i>. Upon receipt of a poisoned TLP, the application must report the error on app_err_bus and either assert app_err_advisory (to indicate an advisory error) or deassert app_err_advisory (to indicate that the application is dropping the TLP). <p>Refer to <i>PCI Express 2.0 specification</i> to determine when an application error is an advisory error.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_ERR_EN</p> <p>Valid when: app_hdr_valid is asserted</p>

Table 3-18 SII Signals: Transmit Control (Continued)

Signal	Input/O utput	Description
app_err_func_num[NFUNC_WD-1:0]	I	<p>Function: The number of the function that is reporting the error indicated on app_err_bus, valid when app_hdr_valid is asserted.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_ERR_EN</p> <p>Valid when: app_hdr_valid is asserted</p>
app_err_vfunc_num[NVFUNC_NUM_WD-2:0]	I	<p>Function: The number of the virtual function that is reporting the error indicated on app_err_bus, valid when app_hdr_valid is asserted.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: `APP_RETURN_ERR_EN and 'CX_SRIOV_ENABLE</p> <p>Valid when: app_hdr_valid is asserted</p>
pm_xtlh_block_tlp	O	<p>Function: Indicates that the application must stop generating new outgoing Request TLPs due to the current power management state. The application can continue to generate Completion TLPs.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Always</p>

Table 3-18 SII Signals: Transmit Control (Continued)

Signal	Input/O utput	Description
xadm_ph_cdts[(8*NVC)-1:0]	O	<p>Function: The amount of Posted header buffer space currently available at the receiver at the other end of the Link (in units of Posted header credits). The core maintains the current number of credits on xadm_ph_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 8 bits of xadm_ph_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Always</p>
xadm_pd_cdts[(12*NVC)-1:0]	O	<p>Function: The amount of Posted data buffer space currently available at the receiver at the other end of the Link (in units of Posted data credits). The core maintains the current number of credits on xadm_pd_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 12 bits of xadm_pd_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Always</p>
xadm_nph_cdts[(8*NVC)-1:0]	O	<p>Function: The amount of Non-Posted header buffer space currently available at the receiver at the other end of the Link (in units of Non-Posted header credits). The core maintains the current number of credits on xadm_nph_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 8 bits of xadm_nph_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Always</p>

Table 3-18 SII Signals: Transmit Control (Continued)

Signal	Input/O utput	Description
xadm_npd_cdts[(12*NVC)-1:0]	O	<p>Function: The amount of Non-Posted data buffer space currently available at the receiver at the other end of the Link (in units of Non-Posted data credits). The core maintains the current number of credits on xadm_npd_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 12 bits of xadm_npd_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Only needed if you want the application to check available credits before submitting TLPs for transmission</p>
xadm_cplh_cdts[(8*NVC)-1:0]	O	<p>Function: The amount of Completion header buffer space currently available at the receiver at the other end of the Link (in units of Completion header credits). The core maintains the current number of credits on xadm_cplh_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 8 bits of xadm_cplh_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: core_clk</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Only needed if you want the application to check available credits before submitting TLPs for transmission</p>

Table 3-18 SII Signals: Transmit Control (Continued)

Signal	Input/O utput	Description
xadm_cpld_cdts[(12*NVC)-1:0]	O	<p>Function: The amount of Completion data buffer space currently available at the receiver at the other end of the Link (in units of Completion data credits). The core maintains the current number of credits on xadm_cpld_cdts as the receiver continues to send UpdateFC DLLPs and the application uses credits by sending new TLPs to be transmitted.</p> <p>There are 12 bits of xadm_cpld_cdts assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `XADM_CRD_EN</p> <p>Valid when: Only needed if you want the application to check available credits before submitting TLPs for transmission</p>
app_parity_errs[2:0]	O	<p>Function: Indicates that the core detected a parity error, one bit for each of the following parity errors:</p> <ul style="list-style-type: none"> • app_parity_errs[0]: XTLH detected a parity error on the transmit datapath • app_parity_errs[1]: XADM detected a parity error on the transmit datapath • app_parity_errs[2]: RADM detected a parity error on the receive datapath <p>The app_parity_errs signals are one-cycle pulses that indicate the associated parity error occurred; they do not indicate which packet contained the parity error.</p> <p>A suggested usage of the app_parity_errs signals is to register each bit and to provide a control to turn off system notification of parity errors. By doing so, the application can choose to only respond to the first detection of a parity error.</p> <p>The core performs transmit and receive parity checking only if you select 8, 16, or 32 for the `CX_CLIENT_PAR_MODE configuration option.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_PAR_ERR_OUT_EN</p> <p>Valid when: app_hdr_valid is asserted</p>

3.13.8 SII Signals: Receive Control/Completion Timeout

[Table 3-19](#) defines the SII signals that relate to control of TLP reception and completion timeout.

Table 3-19 SII Signals: Receive Control/Completion Timeout

Signal	Input/O utput	Description
app_ph_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Posted header credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_ph_ca input each time the application consumes a Posted header credit so that the core can update the number of Posted header credits available.</p> <p>There is 1 bit of app_ph_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>
app_pd_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Posted data credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_pd_ca input each time the application consumes a Posted data credit so that the core can update the number of Posted data credits available.</p> <p>The application must assert app_pd_ca once for each packet, even if the packet contains less than 16 bytes (1 credit) of data.</p> <p>There is 1 bit of app_pd_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-19 SII Signals: Receive Control/Completion Timeout (Continued)

Signal	Input/O utput	Description
app_nph_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Non-Posted header credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_nph_ca input each time the application consumes a Non-Posted header credit so that the core can update the number of Non-Posted header credits available.</p> <p>There is 1 bit of app_nph_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>
app_npd_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Non-Posted data credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_npd_ca input each time the application consumes a Non-Posted data credit so that the core can update the number of Non-Posted data credits available.</p> <p>The application must assert app_npd_ca once for each packet, even if the packet contains less than 16 bytes (1 credit) of data.</p> <p>There is 1 bit of app_npd_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>

Table 3-19 SII Signals: Receive Control/Completion Timeout (Continued)

Signal	Input/O utput	Description
app_cplh_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Completion header credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_cplh_ca input each time the application consumes a Completion header credit so that the core can update the number of Completion header credits available.</p> <p>There is 1 bit of app_cplh_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>
app_cpld_ca[NVC-1:0]	I	<p>Function: One-cycle pulse indicating that Completion data credits have been allocated. In implementations where the application is responsible for queuing the receive data (such as in bypass mode), the application must pulse the app_cpld_ca input each time the application consumes a Completion data credit so that the core can update the number of Completion data credits available.</p> <p>The application must assert app_cpld_ca once for each packet, even if the packet contains less than 16 bytes (1 credit) of data.</p> <p>There is 1 bit of app_cpld_ca assigned to each configured Virtual Channel.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `APP_RETURN_CRD_EN</p> <p>Valid when: Not validated by another signal</p>
radm_cpl_timeout	O	<p>Function: Indicates that the Completion TLP for a Request has not been received within the expected time window.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-19 SII Signals: Receive Control/Completion Timeout (Continued)

Signal	Input/O utput	Description
radm_timeout_fun_num[NFUNC_WD-1:0]	O	<p>Function: The Function Number of the timed out Completion</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: radm_cpl_timeout is asserted</p>
radm_timeout_cpl_tc[2:0]	O	<p>Function: The Traffic Class of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: radm_cpl_timeout is asserted</p>
radm_timeout_vfun_active	O	<p>Function: Indicates that a virtual function (VF) had a CPL timeout.</p> <p>state 0: No VF is active and radm_timeout_vfun_num is invalid. A PF is valid and identified by radm_timeout_fun_num.</p> <p>state 1: A VF is active and is identified by radm_timeout_vfun_num.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: `CX_SRIOV_ENABLE</p> <p>Valid when: radm_cpl_timeout is asserted</p>
radm_timeout_vfun_num[NVFUNC_NUM_WD-2:0]	O	<p>Function: Indicates which virtual function (VF) had a CPL timeout.</p> <p>Active State: NA</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP mode/EP (not RC/SW)</p> <p>Populated by: `CX_SRIOV_ENABLE</p> <p>Valid when: when radm_cpl_timeout and radm_timeout_vfun_active are asserted</p>

Table 3-19 SII Signals: Receive Control/Completion Timeout (Continued)

Signal	Input/O utput	Description
radm_timeout_cpl_attr[1:0]	O	<p>Function: The Attributes field of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: radm_cpl_timeout is asserted</p>
radm_timeout_cpl_len[11:0]	O	<p>Function: Length (in bytes) of the timed out completion. For a split completion, it indicates the number of bytes remaining to be delivered when the completion timed out.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: radm_cpl_timeout is asserted. If `RADM_CPL_LUT_STORE_BYT_CNT is not populated this signal will always be 0.</p>
radm_timeout_cpl_tag[7:0]	O	<p>Function: The Tag field of the timed out Completion.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: Always</p> <p>Valid when: radm_cpl_timeout is asserted</p>
radm_cpl_lut_valid	O	<p>Function: Indicates which entries in the completion lookup table have valid entries stored.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/EP (not SW)</p> <p>Populated by: `RADM_TAG_PORT_EN</p> <p>Valid when: Always</p>

3.13.9 SII Signals: Reset Generation

[Table 3-20](#) defines the SII signals that relate to reset generation.

Table 3-20 SII Signals: Reset Generation

Signal	Input/O utput	Description
xmlh_link_up	O	<p>Function: PHY Link up/down indicator:</p> <ul style="list-style-type: none"> • 1: Link is up • 0: Link is down <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
link_req_rst_not	O	<p>Function: Reset request due to Link down status. A high-to-low transition on link_req_rst_not indicates that the core is requesting external logic to reset the core because the PHY link is down.</p> <p>Active State: Low (high-to-low transition is the reset request)</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

3.13.10 SII Signals: Debug

[Table 3-21](#) defines the SII signals that provide useful information for system debugging.

Table 3-21 SII Signals: Debug

Signal	Input/O utput	Description
rdlh_link_up	O	<p>Function: Data Link Layer up/down indicator: This status from the Flow Control Initialization State Machine indicates that flow control has been initiated and the Data Link Layer is ready to transmit and receive packets. NOTE: For multi-VC designs, this signal indicates status for VC0 only.</p> <ul style="list-style-type: none"> • 1: Link is up • 0: Link is down <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: All Populated by: Always Valid when: Not validated by another signal</p>
pm_curnt_state[2:0]	O	<p>Function: Indicates the current power state. The pm_curnt_state output is intended for debugging purposes, not for system operation.</p> <p>Active State: High Registered: Optionally Synchronous to: aux_clk Device Type: All Populated by: Always Valid when: Not validated by another signal</p>
xmlh_ltssm_state[4:0]	O	<p>Function: Current state of the LTSSM. The xmlh_ltssm_state output is intended for debugging purposes, not for system operation.</p> <p>Active State: High Registered: Optionally Synchronous to: core_clk Device Type: All Populated by: Always Valid when: Not validated by another signal</p>

Table 3-21 SII Signals: Debug (Continued)

Signal	Input/O utput	Description
cxpl_debug_info[63:0]	O	<p>Function: State of selected internal signals, for debugging purposes only:</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <ul style="list-style-type: none"> [63]: xmlh_scrambler_disable [62]: xmlh_link_disable [61]: xmlh_link_in_training [60]: xmlh_rcvr_revers_pol_en [59]: xmlh_training_rst_n [58:55]: 0000b [54]: mac_phy_txdetectrx_loopback [53]: mac_phy_txelecidle[0] [52]: mac_phy_txcompliance[0] [51]: app_init_rst [50:48]: 000b [47:40]: rmlh_ts_link_num [39:37]: 000b [36]: xmlh_link_up [35]: rmlh_inskip_rcv [34]: rmlh_ts1_rcvd [33]: rmlh_ts2_rcvd [32]: rmlh_rcvd_lane_rev [31:28]: rmlh_ts_link_ctrl [27]: rmlh_ts_lane_num_is_k237 [26]: rmlh_ts_link_num_is_k237 [25]: rmlh_rcvd_idle[0] [24]: rmlh_rcvd_idle[1] [23:8]: mac_phy_txdata [7:6]: mac_phy_txdata [5]: xmtbyte_skip_sent [4:0]: xmlh_ltssm_state <p>Scrambling disabled for the link</p> <p>LTSSM in DISABLE state. Link inoperable</p> <p>LTSSM performing link training</p> <p>LTSSM testing for polarity reversal</p> <p>LTSSM-negotiated link reset</p> <p>Constant/reserved</p> <p>PIPE receiver detect/loopback request</p> <p>PIPE transmit electrical idle request</p> <p>PIPE transmit compliance request</p> <p>Application request to initiate training reset</p> <p>Constant/reserved</p> <p>Link number advertised/confirmed by link partner</p> <p>Constant/reserved</p> <p>LTSSM reports PHY link up</p> <p>Receiver reports skip reception</p> <p>TS1 training sequence received (pulse)</p> <p>TS2 training sequence received (pulse)</p> <p>Receiver detected lane reversal</p> <p>Link control bits advertised by link partner</p> <p>Currently receiving k237 (PAD) in place of lane number</p> <p>Currently receiving k237 (PAD) in place of link number</p> <p>Receiver is receiving logical idle</p> <p>2nd symbol is also idle (16bit PHY interface only)</p> <p>PIPE Transmit data</p> <p>PIPE transmit K indication</p> <p>A skip ordered set has been transmitted</p> <p>LTSSM current state. See source for encodings</p>

Table 3-21 SII Signals: Debug (Continued)

Signal	Input/O utput	Description
rtlh_rfc_upd[RX_NDLLP-1:0]	O	<p>Function: Indicates that the core received a Flow Control Update DLLP, used for applications that implement Flow Control outside the core.</p> <p>The width of rtlh_rfc_upd (RX_NDLLP) depends on which version of the core you are using. For the 32 and 64-bit versions of the core, RX_NDLLP = 1; rtlh_rfc_upd is a 1-bit signal that indicates that the core received a Flow Control Update DLLP and the data from the Flow Control Update DLLP is available on rtlh_rfc_data[31:0].</p> <p>For the 128-bit versions of the core, RX_NDLLP = 2. rtlh_rfc_upd is a 2-bit signal because it is possible to receive two Flow Control Update DLLPs in one clock cycle. In this case, assertion of one bit indicates one Flow Control Update DLLP was received and assertion of both bits indicates two Flow Control Update DLLPs were received. Each bit corresponds to one DWORD on rtlh_rfc_data:</p> <ul style="list-style-type: none"> • rtlh_rfc_upd[0] indicates valid data Flow Control Update data on rtlh_rfc_data[31:0] • rtlh_rfc_upd[1] indicates valid data Flow Control Update data on rtlh_rfc_data[63:32] <p>The core only asserts rtlh_rfc_upd[1] when two Flow Control Update DLLPs are received in the same clock cycle.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always (in RC mode)</p> <p>Valid when: Not validated by another signal</p>

Table 3-21 SII Signals: Debug (Continued)

Signal	Input/O utput	Description
rtlh_rfc_data[32*RX_NDLLP-1:0]	O	<p>Function: The data from a received Flow Control Update DLLP.</p> <p>The width of rtlh_rfc_upd (RX_NDLLP) depends on which version of the core you are using. For the 32 and 64-bit versions of the core, RX_NDLLP = 1; rtlh_rfc_data is a 32-bit signal that is valid when rtlh_rfc_upd is asserted.</p> <p>For the 128-bit versions of the core, RX_NDLLP = 2. rtlh_rfc_data is a 64-bit signal:</p> <ul style="list-style-type: none"> • rtlh_rfc_data[31:0] is valid when rtlh_rfc_upd[0] is asserted • rtlh_rfc_data[63:32] is valid when rtlh_rfc_upd[1] is asserted <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: Always (in RC mode)</p> <p>Valid when: Not validated by another signal</p>
pm_dstate[(3*NF)-1:0]	O	<p>Function: The current power management D-state of the function:</p> <ul style="list-style-type: none"> • 000b: D0 • 001b: D1 • 010b: D2 • 011b: D3 • 100b: Uninitialized • Other values: Not applicable <p>There are 3 bits of pm_dstate for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: rtlh_rfc_upd is asserted</p>
pm_status[NF-1:0]	O	<p>Function: PME Status bit from the PMCSR.</p> <p>There is 1 bit of pm_status for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-21 SII Signals: Debug (Continued)

Signal	Input/O utput	Description
pm_pme_en[NF-1:0]	O	<p>Function: PME Enable bit in the PMCSR.</p> <p>There is 1 bit of pm_pme_en for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
aux_pm_en[NF-1:0]	O	<p>Function: Auxiliary Power Enable bit in the Device Control register.</p> <p>There is 1 bit of aux_pm_en for each configured function.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>



3.13.11 SII Signals: Diagnostic and Debug Control (optional)

A diagnostic and debug control feature has been added to the core. This feature must be enabled by defining DIAGNOSTIC_ENABLE in coreConsultant.

There are two group of signals introduced. The control bus group (diag_ctrl_bus) is used by the application to insert control over the core's transmission. The diagnostic bus group (diag_status_bus) contains all of the important signals (status) from each core module. These two group of signals are designed to assist the application's debug and control over the core when it is implemented in the application logic.

The control bus, at this time, has 2 bits implemented. Bit 0 is used to insert an LCRC error, bit 1 to insert an ECRC error. LCRC and ECRC errors of a particular packet can be inserted by the application. The rising edge of these signals will enable the core to assert an LCRC or ECRC to the packet that it currently being transferred. When LCRC or ECRC error packets are transmitted by the core, the core asserts signal lcrc_err_asserted or ecrc_err_asserted to report that the requested action has been completed.

The diagnostic bus group is quite large. The application must implement the mux control logic to select the necessary sections (individual diagnostic buses) within the diagnostic bus group.

This handshake between control and status allows the application to control a specific packet being injected with an CRC or ECRC error. The LCRC and ECRC errors are generated by simply inverting the last bit of the LCRC or ECRC value.

The PCIe core (release 2.8 and above) now provides meaningful signal names, instead of requiring indexing into the debug and diagnostic vectors. There are two modules introduced: diag_status_top.v and diag_status.v as part of the core.

The diag_status_top.v file contains the module definition, including input and output ports. This file can be modified by the customer to bring out signals that are interesting to their particular design. The diag_status.v file is maintained by Synopsys and performs the conversion from indexing into the debug and diagnostic vectors to meaningful signal names. These files are not automatically instantiated from the core. If you are interested in using these files, please contact Synopsys Customer Support for further details.

[Table 3-22](#) defines all of the diagnostic and debug control signals. The individual diagnostic buses and their corresponding signals are identified in the table in the order from most significant bit to least significant bit (with respect to the entire diagnostic bus group).

Table 3-22 SII Signals: Diagnostic and Debug Control

Signal	Signal Width (in Bits)	Comments
Control Bus Group (diag_ctrl_bus)		
Diagnostic Control Bus		
diag_fast_link_mode	1	[2]: Force fast link mode
diag_ctrl_bus	2	[0]: Invert LSB of LCRC [1]: Invert LSB of ECRC
Diagnostic Bus Group (diag_status_bus)		
CDM Diagnostic Bus		

Table 3-22 SII Signals: Diagnostic and Debug Control (Continued)

Signal	Signal Width (in Bits)	Comments
lbc_cdm_addr	32	Local bus Address
lbc_cdm_data	32	Local bus Data
lbc_cdm_cs	NF	Local bus chip select
lbc_cdm_wr	4	Local bus Write enable (per byte)
cdm_lbc_ack	NF	Local bus Acknowledge
radm_rcvd_wreq_poisoned	NF	Received a write with poisoned payload
radm_rcvd_cpl_poisoned	NF	Received a completion with poisoned payload
radm_mlf_tlp_err	NF	Received a malformed TLP
radm_ecrc_err	NF	Received a TLP with ECRC error
radm_rcvd_ur_req	NF	Received a request which device does not support
radm_rcvd_ca_req	NF	Completer aborted request
radm_rcvd_cpl_ca	NF	Received a completion with CA status
radm_rcvd_cpl_ur	NF	Received a completion with UR status
radm_unexp_cpl_err	NF	Received an unexpected completion
radm_qoverflow	NVC	Receive queue overflow. Normally happens only when flow control advertisements are ignored
radm_cpl_timeout	1	An request failed to complete in the allotted time
int_xadm_fc_prot_err	1	Flow control update protocol violation (opt. checks)
rmlh_rcvd_err	1	Received PHY error this cycle
rtlh_fc_prot_err	1	Flow control protocol violation (watchdog timer)
rdlh_prot_err	1	DLLP protocol error (out of sequence DLLP)
rdlh_bad_tlp_err	1	Received TLP with DataLink Layer error
rdlh_bad_dllp_err	1	Received DLLP with DataLink Layer error
xdlh_replay_num_rlover_err	1	Max retries exceeded
xdlh_replay_timeout_err	1	Replay timer expired
pm_dev_num	5*NF	Device number (per function)
pm_bus_num	8*NF	Bus number (per function)
XADM Diagnostic Bus		
arb_reqs	NCL+2	Transmit requests from each client
fc_cds_pass	(NCL+2)*NVC	Credit check passed for each request
grant_ack	NCL+2	This client has completed his request (1-hot)
active_grant	NCL+2	Currently granted client (1-hot)
RADM Diagnostic Bus		

Table 3-22 SII Signals: Diagnostic and Debug Control (Continued)

Signal	Signal Width (in Bits)	Comments
form_filt_hv	1	Information to the receive packet filter block is valid
form_filt_hdr	128	Header data (size of stored header is configurable)
form_filt_dv	1	Packet is in payload stage
rtlh_radm_data	32*NW	Packet data from receive transaction layer
rtlh_radm_dwen	NW	DWORD enables
rtlh_radm_eot	1	End of TLP received this cycle
rtlh_radm_dllp_err	1	This TLP has a DataLink Layer Error (for example, LCRC)
rtlh_radm_malform_tlp_err	1	This TLP is malformed
int_rtlh_radm_ecrc_err	1	This TLP had an ECRC error
flt_q_header_destination	2	Destination interface <ul style="list-style-type: none">• 0 = Drop• 1 = TRGT0• 2 = TRGT1• 3 = CPL
flt_q_header_cpl_status	3	Completion Status of the current received completion <ul style="list-style-type: none">• 0 = Successful• 1 = Unsupported Request (UR)• 2 = Configuration Retry (CRS)• 4 = Completer Abort (CA)
flt_q_cpl_abort	1	Current completion is being aborted (trashed)
flt_q_cpl_last	1	Final completion for the transaction
cpl_mlf_err	1	Malformed completion
cpl_ur_err	1	Received completion with UR status
cpl_ca_err	1	Received completion with CA status
unexpected_cpl_err	1	Received a completion that was unexpected
PM Diagnostic Bus		
xdlh_nodllp_pending	1	There are no pending or in-progress packets going to the PHY.
xdlh_xmt_pme_ack	1	DataLink layer just transmitted a PME msg (PM_PME, PME_TURN_OFF, PME_ACTICE_STATE_NAK, or PME_TO_ACK).
xdlh_not_expecting_ack	1	All transmitted TLPs have been acknowledged by the link partner.
xadm_had_enough_credit	NVC	The core has enough transmit credits of each type (per VC) to meet the PM entry criteria.
xadm_notlp_pending	1	No TLP transmit requests currently pending (from internal or external clients).
xadm_no_fc_credit	NVC	No credits of any type are available (per VC).

Table 3-22 SII Signals: Diagnostic and Debug Control (Continued)

Signal	Signal Width (in Bits)	Comments
CXPL Diagnostic Bus		
MAC Layer Diagnostic Signals		
rmlh_inskip_rcv	1	Skip character is received this cycle
rmlh_ts_rcv_err	1	During training, lane 0 TS sequence problem
rmlh_ts1_rcvd	1	At least one active lane received TS1 this cycle
rmlh_ts2_rcvd	1	At least one active lane received TS2 this cycle
rmlh_ts_lane_num_is_k237	1	Received lane number (lane 0) is PAD (k237)
rmlh_deskew_alignment_err	1	Deskew logic overflow - unable to align lanes
rmlh_ts_link_num_is_k237	1	Received link number (lane 0) is PAD (k237)
rmlh_rcvd_lane_rev	1	Receive logic detected logical lane reversal
rmlh_rcvd_idle[0]	1	Logical Idle seen for 8+ symbols on all active lanes
rmlh_rcvd_idle[1]	1	Logical Idle seen for 1+ symbols on all active lanes
rmlh_rcvd_eidle_set	1	Received EIDLE ordered set, any active lane
rmlh_lanes_rcving	NL	Lanes active in link training
rmlh_rdlh_nak	NW	MAC layer detected runt STP (per DWORD)
rmlh_rdlh_dllp_start	NW	DLLP starting (per DWORD)
rmlh_rdlh_tlp_start	NW	TLP starting (per DWORD)
rmlh_rdlh_pkt_end	NW	DLLP/TLP ending (per DWORD)
rmlh_rdlh_pkt_edb	NW	Packet terminated with EDB (per DWORD)
rmlh_rdlh_pkt_dv	1	Receive data interface from PHY to DLL valid this cycle (rmlh_rdlh_*)
rmlh_rdlh_pkterr	NW	Physical Error detected (per DWORD)
Data Link Layer Diagnostic Signals		
rdlh_rtlh_tlp_sot	NW	Start of TLP (per DWORD)
rdlh_rtlh_tlp_eot	NW	End of TLP (per DWORD)
rdlh_rtlh_tlp_dv	1	Receive data interface from DLL to Transaction Layer valid this cycle (rdlh_rtlh_*)
xdlh_xmlh_start_link_retrain	1	Max replays attempted, request to retrain link
rtlh_req_link_retrain	1	Receive watchdog timer expired, retrain link
cfg_link_retrain	1	Software programmed link retrain request
rdlh_xdlh_rcvd_nack	1	DataLink Layer received NAK DLLP
rdlh_xdlh_rcvd_ack	1	DataLink Layer received ACK DLLP
rdlh_xdlh_rcvd_acknack_seqnum	12	Sequence number corresponding to NAK/ACK
rdlh_xdlh_req2send_ack	1	DataLink Layer request to send ACK
rdlh_xdlh_req2send_ack_due2dup	1	Request to send ACK due to duplicate TLP
rdlh_xdlh_req2send_nack	1	DataLink Layer request to send NAK
rdlh_xdlh_req_acknack_seqnum	12	Sequence Number for ACK/NAK DLLP

Table 3-22 SII Signals: Diagnostic and Debug Control (Continued)

Signal	Signal Width (in Bits)	Comments
xdlh_xmlh_eot	NW	Transmit end of TLP/DLLP (per DWORD)
xdlh_xmlh_stp	NW	Transmit Start of TLP (per DWORD)
xdlh_xmlh_sdp	NW	Transmit Start of DLLP (per DWORD)
xdlh_xmlh_data[31:8]	32*NW	Transmit packet payload (completely framed)
xmlh_xdlh_halt	1	PHY Layer not accepting data this cycle
lcrc_err_asserted	1	Link CRC corrupted for this packet
ecrc_err_asserted	1	End-to-end CRC corrupted for this packet
xtlh_xdlh_sot	1	Transmit Start of TLP this cycle
xtlh_xdlh_eot	1	Transmit End of TLP this cycle
xtlh_xdlh_badeot	1	Nullify this transmit TLP (invert CRC, append EDB)
xdlh_xtlh_halt	1	DataLink Layer is not accepting data this cycle
xtlh_xdlh_data	NW*32	Transmit TLP data
fc_credit_consumed_vc	3	Virtual Channel of received TLP
fc_credit_consumed_fctype	2	Flow control type consumed (P=0, NP=1, CPL=2)
fc_credit_consumed	1	Reception of a TLP resulted in credits consumed
fc_credit_consumed_amt	9	Flow control type consumed (P=0, NP=1, CPL=2) (NOTE: hdr credits consumed is always 1)

3.14 PHY Interface (PIPE)

The interface from the core to the PHY follows the PIPE standard, including the recommendations for multi-Lane PIPE. The intent is to connect the core directly to a PIPE-compliant PHY. In the multi-Lane case, the per-Lane signals are concatenated to form a bus, with the low-order bit(s) corresponding to Lane 0.

In addition to the PIPE-compliant interface signals, the core provides two 32-bit sideband buses (`cfg_phy_control` and `phy_cfg_status`) that connect to core configuration registers. You can optionally use the sideband buses for additional PHY control and/or status monitoring.

If `CX_FREQ_STEP_DOWN_EN` is defined, then a module called `freq_step` is placed in between the pipe interface and the core's pipe interface. This module steps up/down the signals to/from the pipe interface. For example, the core can run at 62.5Mhz (4 symbols per clock) and the pipe can run at 250Mhz (1 symbol per clock).

If the Synopsys PHY is chosen as the `PHY_TYPE`, then the PHY will be instantiated inside the `DWC_PCIE` core. In this case, there will be no external PIPE interface. See “[Synopsys PHY Interface](#)” for detailed Synopsys PHY interface signals.

[Table 3-23](#) defines the core PHY interface signals.

Table 3-23 PHY Interface Signals

Signal	Input/O utput	Description
<code>phy_mac_rxdata[(NL*NB*8)-1:0]</code>	I	<p>Function: Parallel received data, 8, 16, or 32 bits per Lane:</p> <ul style="list-style-type: none"> Bits 7:0 correspond to the first symbol of Lane 0. Bits 15:8 correspond to the second symbol of Lane 0 (16-bit PHY interface) or the single symbol of Lane 1 (8-bit PHY interface). Bits 23:16 correspond to the third symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 2 (8-bit PHY interface) or the first symbol of Lane 1 (16-bit PHY interface). Bits 31:24 correspond to the fourth symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 3 (8-bit PHY interface) or the second symbol of Lane 1 (16-bit PHY interface). <p>The remaining bits continue similarly.</p> <p>Size (bits): Number of Lanes (NL) x Bytes per Lane (NB) x 8</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: <code>pipe_clk</code> if <code>CX_FREQ_STEP_DOWN_EN</code> is defined, else <code>core_clk</code></p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: <code>phy_mac_rxvalid</code> is asserted, per lane</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
phy_mac_rxdatak[(NL*NB)-1:0]	I	<p>Function: Control (K character) indicator bits for received data:</p> <ul style="list-style-type: none"> • Bit 0 corresponds to the first symbol of Lane 0. • Bit 1 corresponds to the second symbol of Lane 0 (16-bit PHY interface) or the single symbol of Lane 1 (8-bit PHY interface). • Bit 2 corresponds to the third symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 2 (8-bit PHY interface) or the first symbol of Lane 1 (16-bit PHY interface). • Bits 3 corresponds to the fourth symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 3 (8-bit PHY interface) or the second symbol of Lane 1 (16-bit PHY interface). <p>The remaining bits continue similarly.</p> <p>Size (bits): Number of Lanes (NL) x Bytes per Lane (NB)</p> <p>Active State: High indicates control symbol, low indicates data symbol.</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: phy_mac_rxvalid is asserted, per lane</p>
phy_mac_rxvalid[NL-1:0]	I	<p>Function: Indicates symbol lock and valid data for each Lane. Bit 0 corresponds to Lane 0.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
phy_mac_phystatus[NL-1:0]	I	<p>Function: Communicates completion of PHY functions including power management transitions and receiver detection.</p> <p>Note: The core_clk should not be shut off before the PHY acknowledges the state change with phystatus.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
phy_mac_rxelecidle[NL-1:0]	I	<p>Function: Indicates receiver detection of an Electrical Idle for each Lane. This may be signaled asynchronously by the PHY macro. Bit 0 corresponds to Lane 0; bit 1 corresponds to Lane 1, and so on, up to the maximum number of Lanes.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
phy_mac_rxstatus[(NL*3)-1:0]	I	<p>Function: Receive status and error codes for each Lane. Bits 2:0 correspond to Lane 0; bits 5:3 correspond to Lane 1, and so on, up to the maximum number of Lanes.</p> <p>Codes:</p> <ul style="list-style-type: none"> • 000: Received data OK • 001: 1 SKP added • 010: 1 SKP removed • 011: Receiver detected • 100: 8b/10b decode error • 101: Elastic buffer overflow • 110: Elastic buffer underflow • 111: Receive disparity error <p>Size (bits): Number of Lanes (NL) x 3</p> <p>Active State: See “Codes” above</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
phy_cfg_status[31:0]	I	<p>Function: Input bus that can optionally be used to read PHY status. The phy_cfg_status bus maps to the PHY Status register. For specific bit usage, see “PHY Status Register” on page 575.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to:</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
mac_phy_txdata[(NL*NB*8)-1:0]	O	<p>Function: Parallel data for transmission:</p> <ul style="list-style-type: none"> Bits 7:0 correspond to the first symbol of Lane 0. Bits 15:8 correspond to the second symbol of Lane 0 (16-bit PHY interface) or the single symbol of Lane 1 (8-bit PHY interface). Bits 23:16 correspond to the third symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 2 (8-bit PHY interface) or the first symbol of Lane 1 (16-bit PHY interface). Bits 31:24 correspond to the fourth symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 3 (8-bit PHY interface) or the second symbol of Lane 1 (16-bit PHY interface). <p>The remaining bits continue similarly.</p> <p>Size (bits): Number of Lanes (NL) x Bytes per Lane (NB) x 8</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: mac_phy_txelecidle is deasserted, per lane</p>
mac_phy_txdatak[(NL*NB)-1:0]	O	<p>Function: Control (K character) indicator bits for transmitted data:</p> <ul style="list-style-type: none"> Bit 0 corresponds to the first symbol of Lane 0. Bit 1 corresponds to the second symbol of Lane 0 (16-bit PHY interface) or the single symbol of Lane 1 (8-bit PHY interface). Bit 2 corresponds to the third symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 2 (8-bit PHY interface) or the first symbol of Lane 1 (16-bit PHY interface). Bit 3 corresponds to the fourth symbol of Lane 0 (32-bit PHY interface) or the single symbol of Lane 3 (8-bit PHY interface) or the second symbol of Lane 1 (16-bit PHY interface). <p>The remaining bits continue similarly.</p> <p>Size (bits): Number of Lanes (NL) x Bytes per Lane (NB)</p> <p>Active State: High indicates control symbol, low indicates data symbol.</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: mac_phy_txelecidle is deasserted, per lane</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
mac_phy_txdetectrx_loopback[NL-1:0]	O	<p>Function: Combined loopback and transmit detect control, per PIPE specification.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
mac_phy_txelecidle[NL-1:0]	O	<p>Function: Forces transmit output to Electrical Idle for each Lane on which it is asserted. Bit 0 corresponds to Lane 0; bit 1 corresponds to Lane 1, and so on, up to the maximum number of Lanes.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
mac_phy_txcompliance[NL-1:0]	O	<p>Function: Sets the running disparity to negative. Used when transmitting compliance pattern. Bit 0 corresponds to Lane 0; bit 1 corresponds to Lane 1, and so on, up to the maximum number of Lanes.</p> <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
mac_phy_rxpolarity[NL-1:0]	O	<p>Function: Directs the PHY to perform a polarity inversion on the received data on the specified Lanes. Bit 0 corresponds to Lane 0; bit 1 corresponds to Lane 1, and so on, up to the maximum number of Lanes.</p> <ul style="list-style-type: none"> • 1: Polarity inversion • 0: No polarity inversion <p>Size (bits): NL (1 bit per Lane)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined, else core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
mac_phy_powerdown[1:0]	O	<p>Function: Power control bits to the PHY.</p> <p>mac_phy_powerdown is a 2-bit signal that is shared by all Lanes.</p> <p>Power states (with corresponding Link states) are:</p> <ul style="list-style-type: none"> • 00: P0 (L0: normal) • 01: P0s (L0s: low recovery time, power saving) • 10: P1 (L1: longer recovery time, additional power saving.) • 11: P2 (L2: lowest power state) <p>Active State: See “Power States” above</p> <p>Registered: Optionally</p> <p>Synchronous to: aux_clock</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
mac_phy_rate	O	<p>Function: Controls the link signaling rate:</p> <ul style="list-style-type: none"> • 0: Use 2.5 GT/s signaling rate • 1: Use 5.0GT/s signaling rate <p>PIPE implementations that only support 2.5GT/s signaling rate do not implement this signal.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined; core_clk if CX_FREQ_STEP_DOWN_EN is not defined</p> <p>Device Type: All</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
mac_phy_txdeemph	O	<p>Function: Selects transmitter de-emphasis:</p> <ul style="list-style-type: none"> • 0: -6 dB de-emphasis at 5GT/s • 1: -3.5 dB de-emphasis at 5GT/s <p>PIPE implementations that only support 2.5 GT/s signaling rate do not implement this signal.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined; core_clk if CX_FREQ_STEP_DOWN_EN is not defined</p> <p>Device Type: All</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>
mac_phy_txmargin[2:0]	O	<p>Function: Selects transmitter voltage levels:</p> <ul style="list-style-type: none"> • 000: Normal operating range • 001: 800-1200 mV for Full swing* or 400-700mV for Half swing • 010: Required and vendor defined • 011: Required and vendor defined • 100: Required and 200-400 mV for Full swing* or 100-200 mV for Half swing* if the last value or vendor defined • 101: Optional and 200-400 mV for Full swing* or 100-200 mV for Half swing* if the last value or vendor defined or Reserved if no other values supported • 110: Optional and 200-400 mV for Full swing* or 100-200 mV for Half swing* if the last value or vendor defined or Reserved if no other values supported • 111: optional and 200-400 mV for Full swing* or 100-200 mV for Half swing* if the last value or Reserved if no other values supported <p>*PIPE2 implementations that only support 2.5GT/s signaling rate do not implement this signal.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
mac_phy_txswing	O	<p>Function: Controls transmitter voltage swing level:</p> <ul style="list-style-type: none"> • 0: Full swing • 1: Low swing <p>Implementation of this signal is optional</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: pipe_clk if CX_FREQ_STEP_DOWN_EN is defined; core_clk if CX_FREQ_STEP_DOWN_EN is not defined</p> <p>Device Type: All</p> <p>Populated by: `CX_GEN2_MODE != 2</p> <p>Valid when: Not validated by another signal</p>
cfg_phy_control[31:0]	O	<p>Function: Output bus that can optionally be used for additional PHY control purposes. The cfg_phy_control bus maps to the PHY Control register. For specific bit usage, see “PHY Control Register” on page 576.</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
cfg_hw_auto_sp_dis	O	<p>Function: Autonomous speed disable</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM in EP/EP mode (not RC/SW)</p> <p>Populated by: 'CX_GEN2_SPEED, downstream ports only</p> <p>Valid when:</p>
rmlh_rcvd_ebuf_overflow_err[NL-1:0]	O	<p>Function: A qualified output signal from the PIPE interface indicating that the elastic buffer has overflowed.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ENABLE_FEATURE_SET_T</p> <p>Valid when: Not validated by another signal</p>

Table 3-23 PHY Interface Signals (Continued)

Signal	Input/O utput	Description
rcvd_err_count[31:0]	O	<p>Function: Received 8b/10b + Disparity error count output</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ENABLE FEATURE SET_T</p> <p>Valid when: Not validated by another signal</p>

3.15 Synopsys PHY Interface

If the Synopsys PHY is chosen as the PHY_TYPE, then the PHY will be instantiated inside the DWC_PCIE core. In this case, there will be no external PIPE interface.

Many of the signals described in this section are direct inputs/outputs of the hard-macro portion of the PHY, or are related to JTAG or DFT. For further clarification on these signals, please refer to pipe_pcs.pdf and pcie1phy_{tsmc130g}_wf_dbook.pdf in the doc directory in the PHY installation area.

[Table 3-24](#) defines the Synopsys PHY interface signals that are connected directly to the PHY; that is, they are direct inputs/outputs of the hard-macro portion of the PHY.

[Table 3-25](#) defines the Synopsys PHY interface signals that are specific to the Synopsys PHY.

Table 3-24 Synopsys PHY Interface Signals: DWC_PCIE Synopsys PHY I/O's

Signal	Input/O utput	Description
snps_phy_phy_acjt_lvl[4:0]	I	Function: Output level for AC-JTAG (1149.6) mode
snps_phy_bs_ck	I	Function: Boundary scan (BS) clock
snps_phy_bs_ctl[5:0]	I	Function: BS control signals
snps_phy_bs_in	I	Function: Input to PHY boundary scan shift register.
snps_phy_bs_out	O	Function: Output of PHY boundary scan shift register.
snps_phy_cko_alive	O	Function: Available clock output which can be controlled by cko_alive_con
snps_phy_cko_alive_con	I	Function: Frequency and source control for cko_alive.
snps_phy_common_clocks	I	Function: When active, indicates that a zero PPM data rate offset exists between the far-end transmitter and the near-end receiver. Active state: High
snps_phy_cr_ack	O	Function: Interface for internal PHY control register access.
snps_phy_cr_cap_addr	I	Function: Interface for internal PHY control register access.
snps_phy_cr_cap_data	I	Function: Interface for internal PHY control register access.
snps_phy_cr_data_in[15:0]	I	Function: Interface for internal PHY control register access.
snps_phy_cr_data_out[15:0]	O	Function: Interface for internal PHY control register access.
snps_phy_cr_read	I	Function: Interface for internal PHY control register access.
snps_phy_cr_write	I	Function: Interface for internal PHY control register access.
snps_phy_dtb_out[1:0]	O	Function: High-speed digital test bus output.
snps_phy_fast_tech	I	Function: Tied high or low based on process technology.
snps_phy_gd	I	Function: Ground connection to PHY.

Table 3-24 Synopsys PHY Interface Signals: DWC_pcie Synopsys PHY I/O's (Continued)

Signal	Input/O utput	Description
snps_phy_gdq[NL-1:0]	I	Function: Ground connection per lane to PHY.
snps_phy_phy_los_lvl[4:0]	I	Function: Loss-of-signal detection sensitivity.
snps_phy_mpll_ncy[4:0]	I	Function: Multiplier control for MPLL based on reference clock frequency.
snps_phy_mpll_ncy5[1:0]	I	Function: Multiplier control for MPLL based on reference clock frequency.
snps_phy_mpll_prescale	I	Function: Multiplier control for MPLL based on reference clock frequency.
snps_phy_mpll_ss_en	I	Function: Enable spread-spectrum clock generation (from non-spread-spectrum source).
snps_phy_pcs_test_clk	I	Function: Test mode clock which can be routed to all flops in the PCS layer via assertion of pcs_test_mode.
snps_phy_pcs_test_mode	I	Function: When active, puts PCS layer into testability mode. Active state: High
snps_phy_pddq_h	I	Function: Power-down for iddq testing.
snps_phy_power_good	O	Function: Indicates that power supplies have settled after a power-on. Can be used to control reset sequence.
snps_phy_resref_f	IO	Function: Calibration resistor connection.
snps_phy_resref_s	I	Function: Calibration resistor connection.
snps_phy_rtune_do_tune	I	Function: Initiates a re-calibration of internal termination resistors w.r.t. the reference resistor. Tie low. Active state: Low
snps_phy_tck	I	Function: Test access port for PHY module.
snps_phy_tms	I	Function: Test access port for PHY module.
snps_phy_tdi	I	Function: Test access port for PHY module.
snps_phy_tdo	O	Function: Test access port for PHY module.
snps_phy_tdo_en	O	Function: Test access port for PHY module.
snps_phy_phy_tx_calc[NL-1:0]	I	Function: Indicate changes have been made to boost or attenuation values.
snps_phy_phy_tx_clk_align[NL-1:0]	I	Function: Normally used when transmitter input clock latency is change dynamically. Tie low.
snps_phy_phy_tx_lvl[4:0]	I	Function: Transmitter output level.

Table 3-24 Synopsys PHY Interface Signals: DWC_pcie Synopsys PHY I/O's (Continued)

Signal	Input/O utput	Description
snps_phy_use_refclk_alt	I	Function: Indicates that MPPLL should use alt_refclk_p/m signals instead of refclk_p/m.
snps_phy_vp	I	Function: Supply connection to PHY.
snps_phy_vp_is_1p2	I	Function: Indicates whether vp is tied to 1.2V supply.
snps_phy_vph	I	Function: Supply connection to PHY.
snps_phy_vph_is_3p3	I	Function: Indicates whether vph is tied to 3.3V supply.
snps_phy_vpq[NL-1:0]	I	Function: Supply connection per lane to PHY.
snps_phy_rx_eq_val[NL*3-1:0]	I	Function: Receiver Equalization Control.
snps_phy_tx_atten[NL*3-1:0]	I	Function: Transmit Attenuation Control.
snps_phy_tx_boost[NL*4-1:0]	I	Function: Transmit Boost Control.
snps_phy_tx_edgerate[NL*2-1:0]	I	Function: Transmit Edge Rate Control.
snps_phy_vreg[NL-1:0]	O	Function: Regulated Voltage Supply for Transmitter.
refclk_alt_m	I	Function: Alternate reference clock inputs when refclk_m is not used.
refclk_alt_p	I	Function: Alternate reference clock inputs when refclk_p is not used.
refclk_m	I	Function: Reference clock input from pad.
refclk_p	I	Function: Reference clock input from pad.
rx_p[NL-1:0]	I	Function: Differential serial data input from pads.
rx_m[NL-1:0]	I	Function: Differential serial data input from pads.
tx_p[NL-1:0]	O	Function: Differential serial data input from pads.
tx_m[NL-1:0]	O	Function: Differential serial data input from pads.

Table 3-25 Synopsys PHY Interface Signals: DWC_pcie Synopsys PHY Specific I/O's

Signal	Input/O utput	Description
snps_phy_mac_phystatus[NL-1:0]	O	<p>Function: Used to communicate completion of several PHY functions.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: snps_phy_pipe_clk when pipe_clk is running. When in low-power states this signal is asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>
snps_phy_mac_rxelecidle[NL-1:0]	O	<p>Function: Indicates receiver detection of an electrical idle. This is an asynchronous signal.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>
snps_phy_pipe_clk	O	<p>Function: Pipe Clock generated from the PHY.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: This is a clock source</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>
phy_clkreq_n	O	<p>Function: PHY is ready for clock removal.</p> <p>Active State: Low</p> <p>Registered: No</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>
mac_phy_clkreq_n	I	<p>Function: Indicates that application logic and MAC are ready for clock removal.</p> <p>Active State: Low</p> <p>Registered: No</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>

Table 3-25 Synopsys PHY Interface Signals: DWC_pcie Synopsys PHY Specific I/O's (Continued)

Signal	Input/O utput	Description
mac_phy_rst_n	I	<p>Function: PHY reset</p> <p>Active State: Low</p> <p>Registered: No</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>
phy_powerdown[1:0]	I	<p>Function: Power management signal to control PHY power states. Generally this would come from application logic. An output of the core, mac_phy_powerdown, is used in application logic along with other states and clocks to determine when to assert phy_powerdown.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: Asynchronous</p> <p>Device Type: All</p> <p>Populated by: PHY_TYPE=4</p> <p>Valid when: Not validate by another signal</p>



3.16 Clock and Reset

Table 3-26 defines the clock and reset signals. For more information about clock and reset implementation, see the *DesignWare Cores PCI Express User Manual*, Chapter 1, “Product Overview” Section “Speed and Clock Requirements.”

Table 3-26 Clock and Reset Signals

Signal	Input/O utput	Description
core_clk	I	<p>Function: The primary clock input to the core. It is assumed that all input signals other than resets are synchronous to this clock. Depending on the core configuration, core_clk is either 62.5 Mhz, 125 MHz, 250MHz, or 500 MHz.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
pipe_clk	I	<p>Function: Clock relative to the signals on the pipe interface. The signals on the pipe interface are synchronous to this clock. Depending on the core configuration, pipe_clk is either 125Mhz or 250Mhz. This clock is used in the freq_step module to step up/down the signals to/from the pipe interface. For example, the core can run at 62.5Mhz (4 symbols per clock) and the pipe can run at 250Mhz (1 symbol per clock).</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: `CX_FREQ_STEP_DOWN_EN</p> <p>Valid when: Not validated by another signal</p>
core_rst_n	I	<p>Function: Resets the core, except for the PMC module. Upon initial power-up, core_rst_n is asserted in order to reset the non-auxiliary power domain logic. reset logic should assert core_rst_n whenever link_req_rst_not is transitioned from high to low.</p> <p>It is recommended that you reset the application logic together with the core.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: Deasserted with core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>



Table 3-26 Clock and Reset Signals (Continued)

Signal	Input/O utput	Description
aux_clk	I	<p>Function: Auxiliary clock to the PMC domain. The partitioning of the core enables some functions to operate on aux_clk in certain power management states.</p> <p>The PMC is partitioned to run on aux_clk in L2 states, if supported by the application.</p> <p>In normal operation, the aux_clk and core_clk inputs are assumed to be equivalent and the physical design implementation must assume they are the same in terms of clock tree matching and skew. For power management enabled solutions, it is the responsibility of the application designer to manage any clock switching required to gate core_clk and switch aux_clk from the normal operating frequency to the slower clock rate used in the low power states.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
pwr_rst_n	I	<p>Function: Resets the PMC module. The pwr_rst_n signal is used as the “cold reset” applied to the core on power-up. pwr_rst_n must be asserted following the application of Aux power or following the application of main power if Aux power is not available to the device.</p> <p>pwr_rst_n resets all registers in the aux_clk domain, including sticky bits.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: Deasserted to aux_clk.</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
sticky_rst_n	I	<p>Function: Resets all sticky bit registers in the configuration register space.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-26 Clock and Reset Signals (Continued)

Signal	Input/O utput	Description
non_sticky_rst_n	I	<p>Function: Resets all non-sticky bit registers in the configuration register space.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
app_init_rst	I	<p>Function: Request from the application to send a Hot Reset to the downstream device. The Hot Reset request is sent when a single cycle pulse is applied to this pin. (For details on Hot Reset, see the User Guide, Chapter 3, Implementation Guidelines.)</p> <p>Active State: High</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: RC, downstream port of a Switch</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
app_req_retry_en	I	<p>Function: Provides a capability to defer incoming Configuration Requests until initialization is complete. When app_req_retry_en is asserted, the core completes incoming Configuration Requests with a Configuration Request Retry Status. Other incoming Requests complete with Unsupported Request status.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: EP</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-26 Clock and Reset Signals (Continued)

Signal	Input/O utput	Description
training_rst_n	O	<p>Function: Hot Reset from upstream component.</p> <p>As defined in the <i>PCI Express 2.0 specification</i>, an upstream component can Hot Reset a downstream component by sending two consecutive TS1 ordered sets with the hot_reset bit asserted. When the core LTSSM receives two consecutive TS1 ordered sets with the hot_reset bit asserted, it asserts training_rst_n for one clock cycle. To accomplish the Hot Reset, the training_rst_n output must be routed to core_RST_n through the reset tree.</p> <p>Note: Endpoint devices cannot Hot Reset upstream components. However, Endpoint devices can be Hot Reset by upstream components.</p> <p>Note: It is recommended that you use link_req_RST_not to reset the core rather than training_rst_n. The training_rst_n signal actually drives link_req_RST_not. (The training_rst_n signal is a legacy signal and has been maintained or customers who have taped out with earlier versions of the core.)</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/EP/SW (not RC)</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>



3.17 External RAM Interface (optional)

If the macro `CX_RAM_AT_TOP_IF` is defined, the interfaces for the retry buffer and receive queue buffers become top-level interfaces of the core for connection of external memory modules.

If `CX_RAM_AT_TOP_IF` is not defined, the retry buffer and receive queue buffers are instantiated inside the top-level hierarchy of the core. External RAM may be preferred for customers doing their own RAM protection (for example, ECC), or doing RAM layout separately.



The parity error signals for the RAMs (*_parerr*) are present regardless of the value of `CX_RAM_AT_TOP_IF`. That is, the *_parerr* signals listed in the external RAM interface signal tables exist as top-level signals regardless of whether the RAMs are internal or external to the core. The direction of the *_parerr* signals depends on whether the RAMs are internal or external:

- If `CX_RAM_AT_TOP_IF` is defined (external RAMs, not FGPA applications), the *_parerr signals are top-level inputs with signal names ending with the string _parrerr_in.
- If `CX_RAM_AT_TOP_IF` is not defined (internal RAMs, FGPA applications only), the *_parerr signals are top-level outputs with signal names ending with the string _parrerr_out.

3.17.1 Retry Buffer and SOT Buffer Interface

[Table 3-27](#) describes the retry buffer and SOT buffer interface signals.

Table 3-27 Retry Buffer and SOT Buffer Interface Signals

Signal	Input/O utput	Description
Retry buffer interface (single-port RAM):		
xdlh_retryram_addr[RBUF_PW-1:0]	O	<p>Function: Address to the retry buffer RAM. The width of the address bus (RBUF_PW) is automatically set as a function of the retry buffer depth, which is either automatically calculated or set explicitly by the user depending whether or not you enable autosizing.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: xdlh_retryram_we or xdlh_retryram_en is asserted</p>



Table 3-27 Retry Buffer and SOT Buffer Interface Signals (Continued)

Signal	Input/O utput	Description
xdlh_retryram_data[`RBUF_WIDTH-1:0]	O	<p>Function: Write data to the retry buffer RAM. The width of the write data bus depends on the application datapath width:</p> <ul style="list-style-type: none"> • For a 32-bit core, `RBUF_WIDTH = 34 • For a 64-bit core, `RBUF_WIDTH = 68 • For a 128-bit core, `RBUF_WIDTH = 134 <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: xdlh_retryram_we is asserted</p>
xdlh_retryram_we	O	<p>Function: Write enable to the retry buffer RAM.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
xdlh_retryram_en	O	<p>Function: Read enable to the retry buffer RAM.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
xdlh_retryram_par_chk_val	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-27 Retry Buffer and SOT Buffer Interface Signals (Continued)

Signal	Input/O utput	Description
retryram_xdlh_data[RBUF_WIDTH-1:0]	I	<p>Function: Read data from the retry buffer RAM. The width is the same as the xdlh_retryram_data width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
retryram_xdlh_parerr_in or retryram_xdlh_parerr_out	I or O	<p>Function: Indicates that the retry buffer parity-checking logic detected a data parity error.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> • If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> - retryram_xdlh_parerr_in is an input signal from external parity error detection logic. - retryram_xdlh_parerr_out does not exist. • If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> - retryram_xdlh_parerr_out is an output signal from internal parity error detection logic. - retryram_xdlh_parerr_in does not exist. <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated as an output</p>
Retry SOT buffer interface (single-port RAM):		
xdlh_retrysotram_addr[SOTBUF_PW-1:0]	O	<p>Function: Address to the SOT buffer RAM. The width of the address bus (SOTBUF_PW) is automatically set as a function of the SOT buffer depth, which is automatically calculated.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: xdlh_retrysotram_we or xdlh_retrysotram_en is asserted</p>

Table 3-27 Retry Buffer and SOT Buffer Interface Signals (Continued)

Signal	Input/O utput	Description
xdlh_retrysotram_data[RBUF_PW-1:0]	O	<p>Function: Write data to the SOT buffer RAM. The width of the write data bus is automatically set to the width of the retry buffer address bus. (Each location in the SOT buffer is used to store a retry buffer address.)</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: xdlh_retrysotram_we is asserted</p>
xdlh_retrysotram_we	O	<p>Function: Write enable to the SOT buffer RAM.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
xdlh_retrysotram_en	O	<p>Function: Read enable to the SOT buffer RAM.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
xdlh_retrysotram_par_chk_val	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>
retrysotram_xdlh_data[RBUF_PW-1:0]	I	<p>Function: Read data from the SOT buffer RAM. The width is the same as the xdlh_retrysotram_data width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Always</p> <p>Valid when: Not validated by another signal</p>

Table 3-27 Retry Buffer and SOT Buffer Interface Signals (Continued)

Signal	Input/O utput	Description
retrystrom_xdlh_parerr_in or retrystrom_xdlh_parerr_out	I or O	<p>Function: Indicates that the SOT buffer parity-checking logic detected a data parity error.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> • If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> - retrystrom_xdlh_parerr_in is an input signal from external parity-checking logic. - retrystrom_xdlh_parerr_out does not exist. • If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> - retrystrom_xdlh_parerrout is an output signal from internal parity-checking logic. - retrystrom_xdlh_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: All Populated by: Always Valid when: Not validated as an output</p>

3.17.2 Receive Queue Interface

The signals for the receive queues depend on the receive queue configuration:

- ❖ In the multiple-buffer configuration, you connect a separate memory module pair (header and data) for each non-bypassed TLP type for each VC. [Table 3-28](#) describes the interface signals. For queues that are bypassed, you do not need to connect the associated memories.
- ❖ In the single-buffer-per-VC configuration, you connect a single memory module pair (header and data) to the Posted buffer interface for each configured VC, using the Posted buffer interface signals listed in [Table 3-28](#). The memories connected to the Posted buffer interface are used for all incoming, non-bypassed Posted, Non-Posted, and Completion TLPs for the associated VC.
- ❖ In the segmented-buffer configuration, there are two external memory modules to connect using the interface signals listed in [Table 3-29](#):
 - ◆ Posted queue header memory
 - ◆ Posted queue data memory

The two memory modules are used for all non-bypassed TLP types and all VCs. Initial segmentation within each memory is controlled through hardware configuration options. At runtime, if dynamic re-segmentation is enabled, the application can change the buffer segmentation by writing to the Port Logic registers.

SW only: Only the segmented-buffer configuration is supported in the SW core.

The address and data widths of the memory interfaces are automatically derived based on other design parameters, as described in the tables. You can view the memory depths and widths in the coreConsultant Specify Configuration dialog. The memory sizes are also available in the simulation log after running a simulation. To view memory size information in the simulation log:

1. Configure the core.
2. Run a simulation.
3. Examine the simulation log. The simulation log will contain a message detailing the memory sizes.

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP)

Signal	Input/O utput	Description
Posted Request header queue interface (2-port RAM):		
p_hdrq_dataout [(RADM_PQ_HWD*NVC)-1:0]	I	<p>Function: Read data from the Posted Request header queue buffer.</p> <p>The width of the read data bus is automatically set according to (number of bits to store the header using only the number of address bits required for the largest configured BAR) x (number of VCs). The minimum width is 88*NVC; the maximum is 142*NVC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: p_hdrq_wea is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_hdrq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_parerr_in[NVC-1:0] or p_hdrq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Posted header queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> • If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> - p_hdrq_parerr_in is an input signal from external parity-checking logic. - p_hdrq_parerr_out does not exist. • If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> - p_hdrq_parerr_out is an output signal from internal parity-checking logic. - p_hdrq_parerr_in does not exist. <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated as an output</p>
p_hdrq_addra [(RADM_PQ_HPW*NVC)-1:0]	O	<p>Function: Address to port A of the Posted Request header queue buffer. The width of the address bus is automatically set as a function of the Posted header queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: p_hdrq_ena or p_hdrq_wea is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_hdrq_addrb [(RADM_PQ_HPW*NVC)-1:0]	O	<p>Function: Address to Port B of the Posted Request header queue buffer. The width is the same as the p_hdrq_addra width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: p_hdrq_enb or p_hdrq_web is asserted</p>
p_hdrq_depth [(RADM_PQ_HPW*NVC)-1:0]	I	<p>Function: The depth of the Posted Request header queue. The width of p_hdrq_depth is the same as the p_hdrq_addra width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: p_hdrq_wea is asserted</p>
p_hdrq_datain [(RADM_PQ_HWD*NVC)-1:0]	O	<p>Function: Write data to the Posted Request header queue buffer. The width is the same as the p_hdrq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_hdrq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Posted Request data queue interface (2-port RAM):		
p_dataq_dataout[(DATAQ_WD*NVC)-1:0]	I	<p>Function: Read data from the Posted Request data queue buffer. The width of the read data bus is automatically set according to the datapath width and number of VCs, as follows:</p> <ul style="list-style-type: none"> For a 32-bit core, data width = 38 * NVC For a 64-bit core, data width = 73 * NVC For a 128-bit core, data width = 137 * NVC <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_dataq_parerr_in[NVC-1:0] or p_dataq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Posted data queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> p_dataq_parerr_in is an input signal from external parity-checking logic. p_dataq_parerr_out does not exist. If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> p_dataq_parerr_out is an output signal from internal parity-checking logic. p_dataq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple/single-buffer configurations Valid when: Not validated as an output</p>
p_dataq_addr [(RADM_PQ_DPW*NVC)-1:0]	O	<p>Function: Address to port A of the Posted Request data queue buffer. The width of the address bus is automatically set as a function of the Posted data queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple/single-buffer configurations Valid when: p_dataq_ena or p_dataq_wea is asserted</p>
p_dataq_addrb [(RADM_PQ_DPW*NVC)-1:0]	O	<p>Function: Address to port B of the Posted Request data queue buffer. The width is the same as the p_dataq_addr width.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple/single-buffer configurations Valid when: p_dataq_enb or p_dataq_web is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_dataq_depth [(RADM_PQ_DPW*NVC)-1:0]	I	<p>Function: The depth of the Posted Request data queue. The width of p_dataq_depth is the same as the p_dataq_addr width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_datain[(DATAQ_WD*NVC)-1:0]	O	<p>Function: Write data to the Posted Request data queue buffer. The width is the same as the p_dataq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: p_dataq_wea is asserted</p>
p_dataq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
p_dataq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple/single-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Non-Posted Request header queue interface (2-port RAM) (Not used in the single-buffer-per-VC configuration):		
np_hdrq_dataout [(RADM_NPQ_HWD*NVC)-1:0]	I	<p>Function: Read data from the Non-Posted Request header queue buffer.</p> <p>The width of the read data bus is automatically set according to (number of bits to store the header using only the number of address bits required for the largest configured BAR) x (number of VCs). The minimum width is 88*NVC; the maximum is 142*NVC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_hdrq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
np_hdrq_parerr_in[NVC-1:0] or np_hdrq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Non-Posted header queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> np_hdrq_parerr_in is an input signal from external parity-checking logic. np_hdrq_parerr_out does not exist. If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> np_hdrq_parerr_out is an output signal from internal parity-checking logic. np_hdrq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: Not validated as an output</p>
np_hdrq_addra [(RADM_NPQ_HPW*NVC)-1:0]	O	<p>Function: Address to port A of the Non-Posted Request header queue buffer. The width of the address bus is automatically set as a function of the Non-Posted header queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: np_hdrq_ena or np_hdrq_wea is asserted</p>
np_hdrq_addrb [(RADM_NPQ_HPW*NVC)-1:0]	O	<p>Function: Address to port B of the Non-Posted Request header queue buffer. The width is the same as the np_hdrq_addra width.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: np_hdrq_enb or np_hdrq_web is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
np_hdrq_depth [(RADM_NPQ_HPW*NVC)-1:0]	I	<p>Function: The depth of the Non-Posted Request header queue. The width of np_hdrq_depth is the same as the np_hdrq_addr width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_hdrq_datain [(RADM_NPQ_HWD*NVC)-1:0]	O	<p>Function: Write data to the Non-Posted Request header queue buffer. The width is the same as the np_hdrq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: np_hdrq_wea is asserted</p>
np_hdrq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Non-Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_hdrq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Non-Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
np_hdrq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Non-Posted Request header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Non-Posted Request data queue interface (2-port2-port RAM) (Not used in the single-buffer-per-VC configuration):		
np_dataq_dataout [(DATAQ_WD*NVC)-1:0]	I	<p>Function: Read data from the Non-Posted Request data queue buffer. The width of the read data bus is automatically set according to the datapath width and number of VCs, as follows:</p> <ul style="list-style-type: none"> For a 32-bit core, data width = 38 * NVC For a 64-bit core, data width = 73 * NVC For a 128-bit core, data width = 137 * NVC <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_dataq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
np_dataq_parerr_in[NVC-1:0] or np_dataq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Non-Posted data queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> • If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> - np_dataq_parerr_in is an input signal from external parity-checking logic. - np_dataq_parerr_out does not exist. • If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> - np_dataq_parerr_out is an output signal from internal parity-checking logic. - np_dataq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: Not validated as an output</p>
np_dataq_addr [(RADM_NPQ_DPW*NVC)-1:0]	O	<p>Function: Address to port A of the Non-Posted Request data queue buffer. The width of the address bus is automatically set as a function of the Non-Posted data queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: np_dataq_ena or np_dataq_wea is asserted</p>

**Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)**

Signal	Input/O utput	Description
np_dataq_addrb [(RADM_NPQ_DPW*NVC)-1:0]	O	<p>Function: Address to port B of the Non-Posted Request data queue buffer. The width is the same as the np_dataq_addrb width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: np_dataq_enb or np_dataq_web is asserted</p>
np_dataq_depth [(RADM_NPQ_DPW*NVC)-1:0]	I	<p>Function: The depth of the Non-Posted Request data queue. The width of np_dataq_depth is the same as the np_dataq_addrb width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_dataq_datain[(DATAQ_WD*NVC)-1:0]	O	<p>Function: Write data to the Non-Posted Request data queue buffer. The width is the same as the np_dataq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: np_dataq_web is asserted</p>
np_dataq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Non-Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>



Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
np_dataq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Non-Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
np_dataq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Non-Posted Request data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Completion header queue interface (2-port RAM) (Not used in the single-buffer-per-VC configuration):		
cpl_hdrq_dataout [(RADM_CPLQ_HWD*NVC)-1:0]	I	<p>Function: Read data from the Completion header queue buffer. The width of the read data bus is automatically set according to (number of bits to store the header using only the number of address bits required for the largest configured BAR) x (number of VCs). The minimum width is 88*NVC; the maximum is 142*NVC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_hdrq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_hdrq_parerr_in[NVC-1:0] or cpl_hdrq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Completion header queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> cpl_hdrq_parerr_in is an input signal from external parity-checking logic. cpl_hdrq_parerr_out does not exist. If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> cpl_hdrq_parerr_out is an output signal from internal parity-checking logic. cpl_hdrq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: Not validated as an output</p>
cpl_hdrq_addra [(RADM_CPLQ_HPW*NVC)-1:0]	O	<p>Function: Address to port A of the Completion header queue buffer. The width of the address bus is automatically set as a function of the Completion header queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: cpl_hdrq_wea or cpl_hdrq_ena is asserted</p>
cpl_hdrq_addrb [(RADM_CPLQ_HPW*NVC)-1:0]	O	<p>Function: Address to port B of the Completion header queue buffer. The width is the same as the cpl_hdrq_addra width.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: cpl_hdrq_web or cpl_hdrq_enb is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_hdrq_depth [(RADM_CPLQ_HPW*NVC)-1:0]	I	<p>Function: The depth of the Completion header queue. The width of cpl_hdrq_depth is the same as the cpl_hdrq_addr width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_hdrq_datain [(RADM_CPLQ_HWD*NVC)-1:0]	O	<p>Function: Write data to the Completion header queue buffer. The width is the same as the cpl_hdrq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: cpl_hdrq_enb is asserted</p>
cpl_hdrq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Completion header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_hdrq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Completion header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_hdrq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Completion header queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Completion data queue interface (2-port RAM) (Not used in the single-buffer-per-VC configuration):		
cpl_dataq_dataout [(DATAQ_WD*NVC)-1:0]	I	<p>Function: Read data from the Completion data queue buffer. The width of the read data bus is automatically set according to the datapath width and number of VCs, as follows:</p> <ul style="list-style-type: none"> For a 32-bit core, data width = 38 * NVC For a 64-bit core, data width = 73 * NVC For a 128-bit core, data width = 137 * NVC <p>Active State: High</p> <p>Registered: No</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_dataq_par_chk_val[NVC-1:0]	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_dataq_parerr_in[NVC-1:0] or cpl_dataq_parerr_out[NVC-1:0]	I or O	<p>Function: Indicates that the parity-checking logic for the Completion data queue buffer detected a data parity error, one bit for each configured VC.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> cpl_dataq_parerr_in is an input signal from external parity-checking logic. cpl_dataq_parerr_out does not exist. If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> cpl_dataq_parerr_out is an output signal from internal parity-checking logic. cpl_dataq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: Not validated as an output</p>
cpl_dataq_addr [(RADM_CPLQ_DPW*NVC)-1:0]	O	<p>Function: Address to port A of the Completion data queue buffer. The width of the address bus is automatically set as a function of the Completion data queue buffer depth, which is either automatically calculated or set explicitly by the user. The calculated (or user-specified) depth is multiplied by the number of VCs.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: DM / RC / EP (not SW) Populated by: Multiple-buffer configurations Valid when: cpl_dataq_wea or cpl_dataq_ena is asserted</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_dataq_addrb [(RADM_CPLQ_DPW*NVC)-1:0]	O	<p>Function: Address to port B of the Completion data queue buffer. The width is the same as the cpl_dataq_addra width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: cpl_dataq_web or cpl_dataq_enb is asserted</p>
cpl_dataq_depth [(RADM_CPLQ_DPW*NVC)-1:0]	I	<p>Function: The depth of the Completion data queue. The width of cpl_dataq_depth is the same as the cpl_dataq_addra width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_dataq_datain [(DATAQ_WD*NVC)-1:0]	O	<p>Function: Write data to the Completion data queue buffer. The width is the same as the cpl_dataq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: cpl_dataq_wea is asserted</p>
cpl_dataq_ena[NVC-1:0]	O	<p>Function: Port A read enable to the Completion data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-28 Receive Queue Buffer Signals: Multiple and Single-Buffer Configurations (DM / RC / EP) (Continued)

Signal	Input/O utput	Description
cpl_dataq_enb[NVC-1:0]	O	<p>Function: Port B read enable to the Completion data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
cpl_dataq_wea[NVC-1:0]	O	<p>Function: Port A write enable to the Completion data queue buffer, one bit for each configured VC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM / RC / EP (not SW)</p> <p>Populated by: Multiple-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-29 Receive Queue Buffer Signals: Segmented-Buffer Configuration

Signal	Input/O utput	Description
Header queue interface for all TLP Types and VCs (2-port RAM):		
p_hdrq_dataout [RADM_SBUF_HDRQ_WD-1:0]	I	<p>Function: Read data from the header queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_par_chk_val	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-29 Receive Queue Buffer Signals: Segmented-Buffer Configuration (Continued)

Signal	Input/O utput	Description
p_hdrq_parerr_in or p_hdrq_parerr_out	I or O	<p>Function: Indicates that the parity-checking logic for the header queue buffer detected a data parity error.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> p_hdrq_parerr_in is an input signal from external parity-checking logic. p_hdrq_parerr_out does not exist. If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> p_hdrq_parerr_out is an output signal from internal parity-checking logic. p_hdrq_parerr_in does not exist. <p>Active State: High Registered: No Synchronous to: core_clk Device Type: All Populated by: Segmented-buffer configurations Valid when: Not validated as output</p>
p_hdrq_addr_a [RADM_SBUF_HDRQ_PW-1:0]	O	<p>Function: Address to port A of the header queue buffer.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: All Populated by: Segmented-buffer configurations Valid when: p_hdrq_ena or p_hdrq_wea is asserted</p>
p_hdrq_addr_b [RADM_SBUF_HDRQ_PW-1:0]	O	<p>Function: Address to Port B of the header queue buffer. The width is the same as the p_hdrq_addr_a width.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: All Populated by: Segmented-buffer configurations Valid when: p_hdrq_enb or p_hdrq_web is asserted</p>

Table 3-29 Receive Queue Buffer Signals: Segmented-Buffer Configuration (Continued)

Signal	Input/O utput	Description
p_hdrq_datain [RADM_SBUF_HDRQ_WD-1:0]	O	<p>Function: Write data to the header queue buffer. The width is the same as the p_hdrq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: p_hdrq_wea is asserted</p>
p_hdrq_ena	O	<p>Function: Port A read enable to the header queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_enb	O	<p>Function: Port B read enable to the header queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_hdrq_wea	O	<p>Function: Port A write enable to the header queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
Data queue interface for all TLP Types and VCs (2-port RAM):		
p_dataq_dataout [RADM_SBUF_DATAQ_WD-1:0]	I	<p>Function: Read data from the data queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

Table 3-29 Receive Queue Buffer Signals: Segmented-Buffer Configuration (Continued)

Signal	Input/O utput	Description
p_dataq_par_chk_val	O	<p>Function: Qualifies parity checking in the external RAM parity-checking logic.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_parerr_in or p_dataq_parerr_out	I or O	<p>Function: Indicates that the parity-checking logic for the data queue buffer detected a data parity error.</p> <p>The direction of the parity error signal depends on whether the RAMs are internal or external:</p> <ul style="list-style-type: none"> • If `CX_RAM_AT_TOP_IF is defined (external RAMs): <ul style="list-style-type: none"> - p_dataq_parerr_in is an input signal from external parity-checking logic. - p_dataq_parerr_out does not exist. • If `CX_RAM_AT_TOP_IF is not defined (internal RAMs): <ul style="list-style-type: none"> - p_dataq_parerr_out is an output signal from internal parity-checking logic. - p_dataq_parerr_in does not exist. <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated as an output</p>
p_dataq_addr [RADM_SBUF_DATAQ_PW-1:0]	O	<p>Function: Address to port A of the data queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: p_dataq_ena or p_dataq_wea is asserted</p>

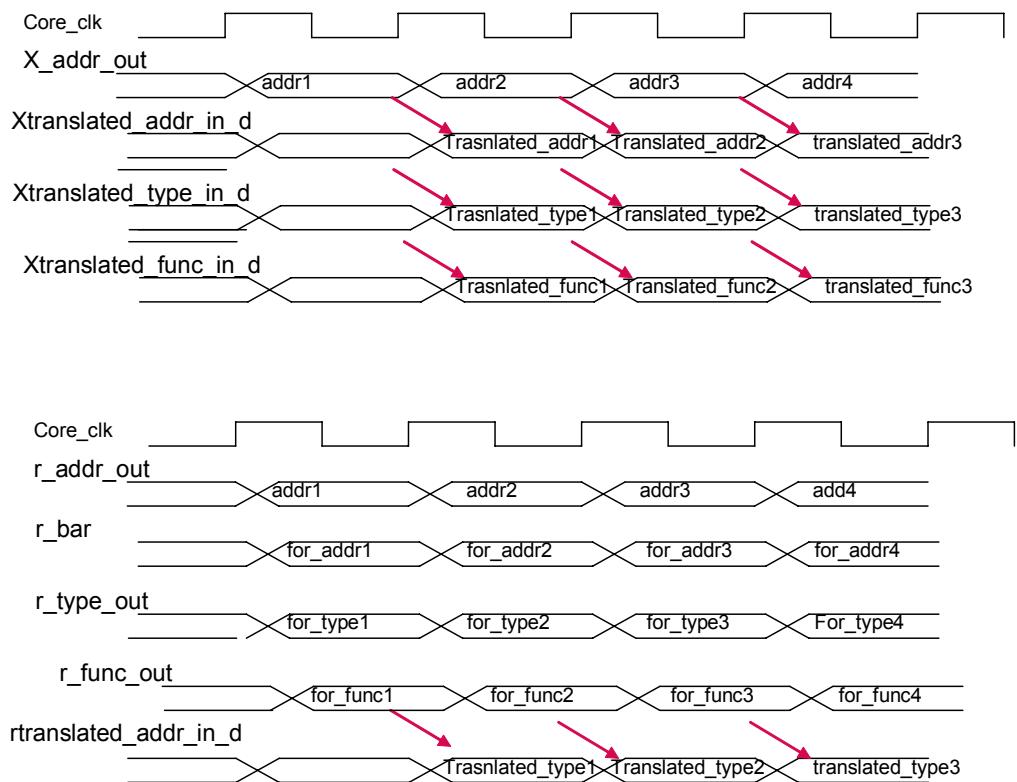
Table 3-29 Receive Queue Buffer Signals: Segmented-Buffer Configuration (Continued)

Signal	Input/O utput	Description
p_dataq_addrb [RADM_SBUF_DATAQ_PW-1:0]	O	<p>Function: Address to port B of the Posted Request data queue buffer. The width is the same as the p_dataq_addra width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: p_dataq_enb or p_dataq_web is asserted</p>
p_dataq_datain [RADM_SBUF_DATAQ_WD-1:0]	O	<p>Function: Write data to the data queue buffer. The width is the same as the p_dataq_dataout width.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_ena	O	<p>Function: Port A read enable to the data queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_enb	O	<p>Function: Port B read enable to the data queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>
p_dataq_wea	O	<p>Function: Port A write enable to the data queue buffer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: Segmented-buffer configurations</p> <p>Valid when: Not validated by another signal</p>

3.18 Address Translation (optional)

This address translation is a proprietary local address translation implementation and is not related to the PCI-SIG ATS specification support. [Figure 3-28](#) shows the timing diagram for address translation.

Figure 3-28 Interface Timing Relationship



The ADDR_TRANSLATION_SUPPORT_EN parameter enables the address translation feature. The signals identified in [Table 3-30](#) are added when address translation support is enabled by the application.



Note There is a tightly defined timing relationship between inputs and outputs that are designed for address translation. One cycle is allowed for the application to perform the address translation between inputs and outputs.

Table 3-30 Address Translation Signals

Signal	I/O	Description
xtranslated_addr_in_d[63:0]	I	<p>Function: The core uses this address to replace the address of current outbound request header. This replacement only applies to the transmit packet with a type of 0 (memory request). The application must drive this signal in the format of the PCIe header layout. For example, if the AXI Memory transfer is translated into a PCIe Configuration transfer, then the application should drive byte0 with the bus number, byte1 of the address is the device and function number, byte2 with the extended register address number, byte3 is register number etc.</p> <p>For MSG TLPs: <code>x_translated_addr_in_d[7:0]</code> is the message code <code>x_translated_addr_in_d[8]</code> is the bit that specifies whether the message has data or not.</p> <p>For Non-MSG TLPs: If GLOB_ADDR_ALIGN_EN is defined, <code>x_translated_addr_in_d[1:0]</code> must be forced to zero. This must be done to support MSG TLPs. If GLOB_ADDR_ALIGN_EN is not defined, then the client interface must not set the lower 2 bits of client0/1/2_tlp_addr.</p> <p>Active State: N/A Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `ADDR_TRANSLATION_SUPPORT_EN Valid when: xtranslate_enable is asserted</p>
xtranslated_type_in_d[4:0]	I	<p>Function: The core uses this request type to replace the current request type in the packet header. This replacement will only apply to the transmit packet with a type of 0 (memory request). This is designed to allow the application to map a particular memory region to the IO or configuration space.</p> <p>Any attempt to translate a memory-locked read TLP to any other TLP type will be ignored. The address and function translation in this case will still be honored.</p> <p>Active State: N/A Registered: Yes Synchronous to: core_clk Device Type: All Populated by: `ADDR_TRANSLATION_SUPPORT_EN Valid when: xtranslate_enable is asserted</p>

Table 3-30 Address Translation Signals (Continued)

Signal	I/O	Description
xtranslated_func_in_d[7:0]	I	<p>Function: The core uses this function number to replace the current function number in the packet header. The upper 5 bits are reserved for future use.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: xtranslate_enable is asserted</p>
x_addr_out[63:0]	O	<p>Function: This is the transmit address of an outbound request provided by the core. It can be translated by external translation logic into xtranslated_addr_in_d.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: xtranslate_enable is asserted</p>
xtranslate_enable	I	<p>Function: The application should drive this signal high when translations are desired.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: xtranslate_enable is asserted</p>
rtranslated_addr_in_d[63:0]	I	<p>Function: The core uses this address to replace the address of the current inbound request header. This replacement applies only to memory, IO and configuration requests. Received IO and configuration requests may be translated into the memory request.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: one cycle after r_addr_out is asserted</p>

Table 3-30 Address Translation Signals (Continued)

Signal	I/O	Description
r_addr_out[63:0]	O	<p>Function: This is the address of an inbound request provided by the core. It can be translated by external translation logic into rtranslated_addr_in_d.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>
r_type_out[4:0]	O	<p>Function: This is the received type field from the current inbound request's header.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>
r_func_out[7:0]	O	<p>Function: This is the received function number field from the current inbound request's header.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>
r_bar_out[2:0]	O	<p>Function: This is the bar number of the current received memory request.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>
r_rombar_out	O	<p>Function: Indicates a received TPL within the ROMBAR.</p> <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>

Table 3-30 Address Translation Signals (Continued)

Signal	I/O	Description
rtranslated_err_in_d[1:0]	I	<p>Function: Translation error indication</p> <p>Note: For future use - not supported in release 2.80a.</p> <ul style="list-style-type: none"> • 00: Normal • 01: UR • 10: CA • 11: Undefined (not allowed) <p>Active State: N/A</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: All</p> <p>Populated by: `ADDR_TRANSLATION_SUPPORT_EN</p> <p>Valid when: Always</p>

3.19 Peer-to-Peer (optional)

The CX_P2P_ENABLE parameter has been added to enable the Peer-to-Peer feature. The signals in [Table 3-31](#) are added when the core is configured for Peer-to-Peer support.

Table 3-31 Peer-to-Peer Signals

Signal	I/O	Description
client0_tlp_p2p_en	I	<p>Function: Used to distinguish the peer-to-peer traffic from locally generated traffic. When it is logic 1, a peer-to-peer packet is presented to the client0 interface. When it is logic 0, a locally generated packet is presented to the client0 interface.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client0_tlp_hv is asserted</p>
client1_tlp_p2p_en	I	<p>Function: Same operation as client0_tlp_p2p_en, but for the client1 interface.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client1_tlp_hv is asserted</p>
client2_tlp_p2p_en	I	<p>Function: Same operation as client0_tlp_p2p_en, but for the client2 interface.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client2_tlp_hv is asserted</p>
client0_app_dev_id[15:0]	I	<p>Function: Used to provide peer-to-peer traffic completer ID for completion TLPs or requester ID for request TLPs.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client0_tlp_hv is asserted</p>

Table 3-31 Peer-to-Peer Signals

Signal	I/O	Description
client1_app_dev_id[15:0]	I	<p>Function: Provides the same information as client0_app_dev_id but for client1.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client1_tlp_hv is asserted</p>
client2_app_dev_id[15:0]	I	<p>Function: Provides the same information as client0_app_dev_id but for client2.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client2_tlp_hv is asserted</p>
client0_hdr_rsvd[14:0]	I	<p>Function: Used to provide header reserved fields for the TLP being presented to the client0 interface.</p> <p>Below is the mapping of the rsvd bits to TLP header reserved fields:</p> <ul style="list-style-type: none"> client0_hdr_rsvdHeader [14]DW2[31] [13:12]DW2[25:24] [11:8]DW2[23:20] [7:6]DW0[19:18] [5]DW0[15] [4:1]DW0[11:8] [0]DW0[7] <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client0_tlp_hv is asserted</p>
client1_hdr_rsvd[14:0]	I	<p>Function: Provides the same information as client0_hdr_rsvd but for the client1 interface.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client1_tlp_hv is asserted</p>

Table 3-31 Peer-to-Peer Signals

Signal	I/O	Description
client2_hdr_rsvd[14:0]	I	<p>Function: Provides the same information as client0_hdr_rsvd but for the client1 interface.</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: client2_tlp_hv is asserted</p>
radm_trgt1_rsvd[14:0]	O	<p>Function: Used to output header reserved fields of TLPs presented on the target1 interface. The bit mapping is the same as client0_hdr_rsvd.</p> <p>Active State: N/A</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: radm_trgt1_hv is asserted</p>
radm_bypass_rsvd[14:0]	O	<p>Function: Used to output header reserved fields of TLPs presented on the bypass interface. The bit mapping is the same as client0_hdr_rsvd.</p> <p>Active State: N/A</p> <p>Registered: Optionally</p> <p>Synchronous to: core_clk</p> <p>Device Type: DM/RC/SW (not EP)</p> <p>Populated by: `CX_P2P_ENABLE</p> <p>Valid when: radm_bypass_hv is asserted</p>

3.20 Function Level Reset (FLR) (DM / EP)

Function Level Reset is a mechanism for allowing a specific function to be reset without affecting other functions or links. The CX_FLR_ENABLE parameter has been added to enable the Function level Reset feature.

The *PCI Express 2.0 Specification* requires the entire reset process to be completed within 100ms. The core requires $(2 * \text{CX_MAX_TAG} * \text{clock_period})$ to execute its internal reset. The application must take this core timing requirement into account so that the overall 100ms reset time is not violated.

[Table 3-32](#) defines the signals related to FLR support.

Table 3-32 Function Level Reset (FLR) Signals

Signal	I/O	Description
app_flr_pf_done[NF-1:0]	I	<p>Function: Indicates that FLR on a physical function has been completed, which means the application has completed initializing its data structures for the function and there are no more TLP(s) associated with this function in the application's transmit queue. This signal must be asserted until the core deasserts cfg_flr_pf_active for the function that has been reset.</p> <p>Active State: High Registered: No Synchronous to: core_clk Device Type: EP Populated by: CX_FLR_ENABLE Valid when: Always</p>
cfg_flr_pf_active[NF-1:0]	O	<p>Function: Set when the software initiates FLR on a physical function by writing to the "Initiate FLR" register bit of that function. This signal is held asserted until both the application and internal reset logic have been completed.</p> <p>Active State: High Registered: Yes Synchronous to: core_clk Device Type: EP Populated by: CX_FLR_ENABLE Valid when: Always</p>

Table 3-32 Function Level Reset (FLR) Signals

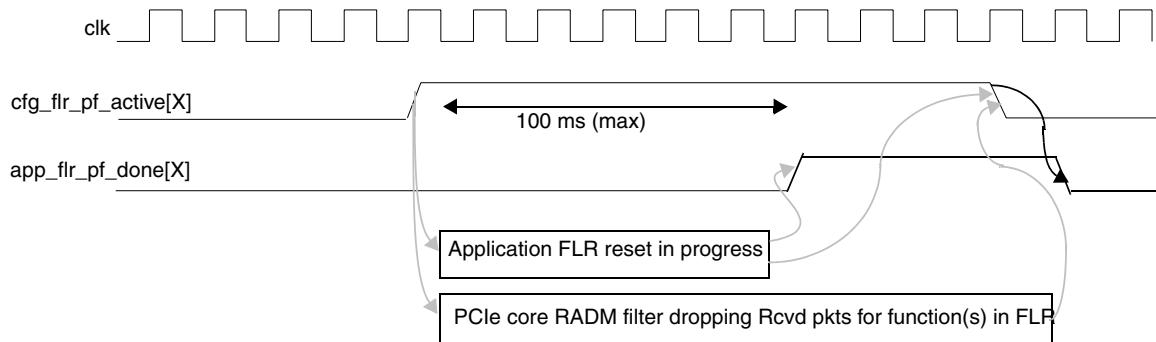
Signal	I/O	Description
app_flr_vf_done[NVFUNC-1:0]	I	<p>Function: Indicates that FLR on a virtual function has been completed, which means the application has completed initializing its data structures for the function and there are no more TLP(s) associated with this function in the application's transmit queue. This signal is held asserted until the core deasserts cfg_flr_vf_active for the function that has been reset.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: EP</p> <p>Populated by: CX_FLR_ENABLE and CX_SRIOV_ENABLE</p> <p>Valid when: Always</p>
cfg_flr_vf_active[NVFUNC-1:0]	O	<p>Function: Set when the software initiates FLR on a virtual function by writing to the "Initiate FLR" register bit of that function. This signal is held asserted until both the application and internal reset logic have been completed.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk</p> <p>Device Type: EP</p> <p>Populated by: CX_FLR_ENABLE and CX_SRIOV_ENABLE</p> <p>Valid when: Always</p>

3.20.1 FLR Timing

Figure 3-29 shows the FLR timing flow for Physical Functions FLR.

1. The host software writes to the Initiate FLR bit of the function to be reset (functionX).
2. The core asserts the `cfg_flr_pf_active[X]` signal.
3. The application cleans up functionX's application logic and verifies that no more TLPs related to functionX will get transmitted. During this time, the receive application logic should not halt the receive interface for TLPs directed to functionX.
4. Once the application logic is done, the application asserts `app_flr_pf_done[X]`. `app_flr_pf_done[X]` remains asserted until the core deasserts `cfg_flr_pf_active[X]`.
5. When the core sees `app_flr_pf_done[X]`, the core initiates its reset of functionX.
6. Once the reset is complete, the core deasserts `cfg_flr_pf_active[X]`.

Figure 3-29 FLR Timing Flow



For Virtual Functions FLR, the steps are the same except for the following:

- ❖ There is a separate signal called `cfg_flr_vf_active[NVF-1:0]`, where NVF is the total number of virtual functions (spans multiple PF).
- ❖ The application's done signal is `app_flr_vf_done[NVF-1:0]`
- ❖ The application should not assert `non_sticky_rst_n` when resetting VF or PF for FLR purposes.
- ❖ When a PF is being reset, it has to reset all of the corresponding VFs of that PF.

3.20.2 FLR Operation

1. When an FLR is initiated, the targeted Function must first return the Completion for the configuration write that initiated the FLR operation and then initiate the FLR.
2. While an FLR is in progress (between the time an FLR has been initiated and the completion of the FLR by the targeted Function):
 - a. If a Posted Request arrives, the Request is silently discarded (following update of flow control credits) without logging or signaling it as an error.
 - b. If a Non-Posted Request arrives, the Request is handled as an Unsupported Request (UR).
 - c. If a Completion arrives, the Completion is silently discarded (following update of flow control credits) without logging or signaling it as an error.
 - d. Once the FLR has completed, received Completions corresponding to Requests issued prior to the FLR must be handled as Unexpected Completions, unless the Function has been re-enabled to issue Requests.
3. If software initiates an FLR when the Device Status Transactions Pending bit is 1b, then software must not initialize the Function until allowing adequate time for any associated Completions to arrive, or to achieve reasonable certainty that any remaining Completions will never arrive. For this purpose, it is recommended that software allow as much time as provided by the pre-FLR value for Completion Timeout on the device. If Completion Timeouts were disabled on the Function when FLR was issued, then the delay is system dependent but must be no less than 100 ms.
4. Note that upon receipt of an FLR, the device Function will clear all transaction status including Transactions Pending. This process is completed in $(2 * CX_MAX_TAG * \text{clock_period})$.
5. While a Function is required to complete the FLR operation within 100 ms, the subsequent Function-specific initialization sequence may require additional time. If additional time is required, the Function must return a Configuration Request Retry Status (CRS) Completion Status when a Configuration Request is received after the time limit above. After the Function responds to a Configuration Request with a Completion other than CRS, it is not permitted to return CRS until it is reset again.
6. Any outstanding INTx interrupt asserted by the Function must be de-asserted by sending the corresponding Deassert_INTx Message prior to starting the FLR.
7. Note that when the FLR is initiated to a Function of a multi-Function device, if another Function continues to assert a matching INTx, no Deassert_INTx Message will be transmitted.

4

Registers

This chapter provides information on the following registers:

- ❖ PCIe Registers (DM in EP Mode / EP)
- ❖ PCIe Registers (DM in RC Mode / RC / SW)
- ❖ PCIe Registers: Port Logic

4.1 PCIe Registers (DM in EP Mode / EP)

This section describes the core implementation of the PCI Express configuration space in the EP core and the DM core is configured to operate in EP mode. The topics for this section are:

- ❖ Register Space Layout
- ❖ PF Register Maps
- ❖ VF Register Maps
- ❖ Accessing Configuration Registers
- ❖ Register Default Values
- ❖ PF PCI-Compatible Configuration Header Register Details
- ❖ PF PCI Standard Capability Structures Register Details
 - ◆ PF PCI Power Management Capability Register Details
 - ◆ PF MSI Capability Register Details
 - ◆ PF PCI Express Capability Register Details
 - ◆ PF MSI-X Capability Register Details
 - ◆ PF VPD Capability Register Details
- ❖ PF PCI Express Extended Capability Register Details
 - ◆ PF Advanced Error Reporting Capability Registers
 - ◆ PF Virtual Channel Capability Registers
 - ◆ PF Device Serial Number Capability Register
 - ◆ PF Power Budgeting Capability
 - ◆ PF ARI Capability Registers
 - ◆ PF SR-IOV Capability Registers

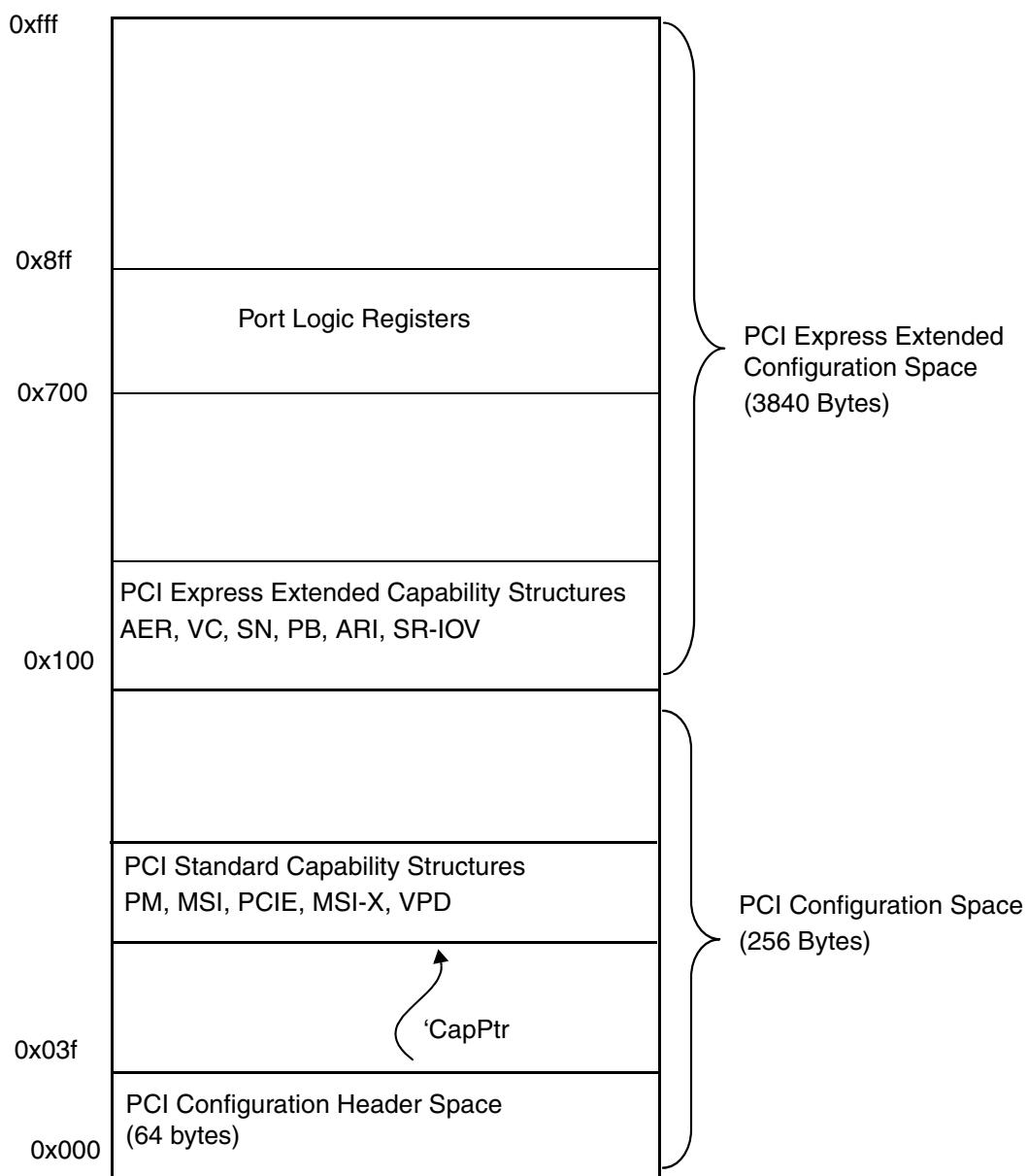
- ❖ VF PCI-Compatible Configuration Header Register Details
- ❖ VF PCI Standard Capability Structures Register Details
 - ◆ VF PCI Express Capability Register Details
 - ◆ VF MSI-X Capability Registers

4.1.1 Register Space Layout

The core has 4096 bytes of PCI Express configuration space per function. For each function, the PCI Express configuration space is divided into:

- ❖ 256 bytes of PCI Configuration Space, containing:
 - ◆ 64 bytes of PCI 3.0 Compatible Configuration Space Header (type 0 in EP mode)
 - ◆ PCI Standard Capabilities Structures linked list, which can start anywhere after offset 0x40
- ❖ 3840 bytes of PCI Express Extended Configuration Space (which starts at offset 0x100), containing:
 - ◆ PCI Express Extended Capabilities Structures linked list, which starts at offset 0x100
 - ◆ Port Logic registers (vendor-specific registers), which start at offset 0x700. The Port Logic registers have specific pre-defined usages, mostly for test purposes, and can optionally be removed from the core hardware configuration. The usage of the Port Logic registers is the same in both EP mode and RC mode. See “[PCIe Registers: Port Logic](#)” on page [504](#) for details.

[Figure 4-1](#) shows the EP mode layout of the core configuration space.

**Figure 4-1 Core Configuration Space Layout (EP Mode)**

4.1.2 PF Register Maps

Capability configuration registers are in structures (groups) identified by a Capability ID. Groups are linked together as in PCI. Register locations within a group are specified, but the starting location of each group must be found by traversing the linked list. There are two linked lists of register groups: PCI Compatible Capability registers and PCI Express Extended Capability registers. PCI Compatible Capability register groups begin at the configuration address stored in the capability pointer register at 0x34. PCI Express Extended Capability register groups begin at address 0x100.

4.1.2.1 PF PCI Configuration Space Header – Type 0

[Table 4-1](#) shows the configuration field register definitions for PCI Express Type 0 Configuration Space header. Most PCI-compatible register fields have the same software interpretation in PCI 3.0 and PCI Express. Refer to the indicated page numbers for detailed register descriptions.

Table 4-1 PF PCI Configuration Space Header – Type 0

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x00	Device ID		Vendor ID		336
0x04	Status Register		Command Register		337
0x08	Class Code			Revision ID	339
0x0C	BIST(0x00)	Header Type	Latency Timer	Cache Line Size	342
0x10	Base Address Register 0				345
0x14	Base Address Register 1				345
0x18	Base Address Register 2				345
0x1C	Base Address Register 3				345
0x20	Base Address Register 4				345
0x24	Base Address Register 5				345
0x28	CardBus CIS Pointer				
0x2C	Subsystem ID		Subsystem Vendor ID		359
0x30	Expansion ROM Base Address				360
0x34	Reserved			CapPtr	358
0x38	Reserved				
0x3C	Max_Latency ¹	Min_Grant ¹	Interrupt Pin	Interrupt Line	362

1. The Max_Latency and Min_Grant registers do not apply to PCI Express and are read-only registers with values hardwired to 0x00.

4.1.2.2 PF PCI Standard Capability Structures Register Maps

The Capability Pointer register points to the next item in the linked list of capabilities, which by default is the PCI Power Management Capability register space.



Note Even though there is an unique standard capabilities linked lists provided per function, specific capabilities cannot be enabled/disabled on a per-function basis; e.g., MSI-X capability is enabled/disabled for all functions (PF) at the same time through the coreConsultant GUI. Each function (PF) may have a different configuration of that capability structure once it is enabled, although some features/settings are common across all functions.

[Table 4-2](#) lists the capabilities supported by the core and their respective default address offsets and next capability pointers. The starting addresses are defined by parameters that are not visible in coreConsultant. [Table 4-3](#) provides the default values of these parameters.

Table 4-2 PF Configuration Structure: Starting Addresses and Next Capability Pointers

Start Address Offset	Item	Next Pointer
0x00	PCI-Compatible Header (Type 0)	`CFG_NEXT_PTR
`CFG_PM_CAP	PCI Power Management	`PM_NEXT_PTR
`CFG_MSI_CAP	Message Signaled Interrupt (MSI)	`MSI_NEXT_PTR
`CFG_PCIE_CAP	PCI Express Capabilities	`PCIE_NEXT_PTR
`CFG_MSIX_CAP	MSI-X	`MSIX_NEXT_PTR
`CFG_VPD_CAP	VPD	`VPD_NEXT_PTR
NOTE: These parameters not visible in coreConsultant.		

Table 4-3 PF Default Values of Parameters that Define Starting Addresses

Parameter	Default Value
CFG_PM_CAP	8'h40
CFG_MSI_CAP	8'h50
CFG_PCIE_CAP	8'h70
CFG_MSIX_CAP	8'hB0
CFG_VPD_CAP	8'hD0

The following tables show the PCI Standard Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-4 PF Power Management Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_PM_CAP	Power Management Capabilities (PMC)		Next Capability Pointer (`PM_NEXT_PTR)	Capability ID (0x01)	364
+0x4	Data	PMCSR_BSE Bridge Extensions	Power Management Control Status Register (PMCSR)		366

Table 4-5 PF MSI Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_MSI_CAP	Message Control Register		Next Capability Pointer (`MSI_NEXT_PTR)	Capability ID (0x05)	368
+0x4	MSI Lower 32-bit Address Register				368
+0x8	MSI Upper 32-bit Address Register				368
+0xC	Reserved		MSI Data		369
+0x10	Mask Bits Register				369
+0x14	Pending Bits Register				369

Table 4-6 PF PCI Express Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_PCIE_CAP	PCI Express Capabilities Register		Next Capability Pointer (`PCIE_NEXT_PTR)	Capability ID (0x10)	371
+0x4	Device Capabilities				372
+0x8	Device Status		Device Control		374
+0xC	Link Capabilities				375
+0x10	Link Status		Link Control		376
+0x24	Device Capabilities 2				378
+0x28			Device Control 2		379
+0x30	Link Status 2		Link Control 2		380

Table 4-7 PF MSI-X Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_MSIX_CAP	MSI-X Control Register		Next Capability Pointer (`MSIX_NEXT_PTR)	Capability ID (0x11)	384
+0x4	Table Offset (31:3) and BIR (2:0)				385
+0x8	PBA Offset (31:3) and BIR (2:0)				385

Table 4-8 PF VPD Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_VPD_CAP	VPD Capability ID				387
+00h	VPD Control and Capabilities register				387
+04h	VPD Data register				387

4.1.2.3 PF PCI Express Extended Capability Register Maps

The PCI Express Extended Capabilities registers are located in device configuration space at offsets 0x100 or higher. As with PCI Standard Capability Structures, the PCI Express Extended Capability structures are allocated using a linked list with a similar method and format to those of PCI.



Note Even though there is an unique standard capabilities linked lists provided per function, specific capabilities cannot be enabled/disabled on a per function basis; e.g., MSI-X capability is enabled/disabled for all functions (PF) at the same time through the coreConsultant GUI.

Each function (PF) may have a different configuration of that capability structure once it is enabled, although some features/settings are common across all functions.

The Advanced Error Reporting (AER) Capability and Virtual Channel (VC) Capability are optional extended capabilities that may be implemented by PCI Express devices supporting advanced error control and reporting and multiple VCs, respectively. The Advanced Error Reporting Capability is required when the device supports ECRC generation/checking. The Virtual Channel Capability is required for any device that supports multiple VCs and/or multiple Traffic Classes (TCs).

The Next Capability Pointer register in the PCI Express Extended Capability Structures Register Maps points to the next item in the linked list of capabilities, which, by default, is the Advanced Error Reporting (AER) Capability register space.

Table 4-9 lists the capabilities supported by the core and their respective default address offsets and next capability pointers. The starting addresses are defined by parameters that are not visible in coreConsultant.

Table 4-9 PF Configuration Structure: Starting Addresses and Next Capability Pointers

Start Address Offset	Item	Next Pointer
0x00	PCI-Compatible Header (Type 0)	`CFG_NEXT_PTR
`AER_PTR	Advanced Error Reporting	`AER_NEXT_PTR
`VC_PTR	Virtual Channel	`VC_NEXT_PTR
`SN_PTR	Device Serial Number	`SN_NEXT_PTR
`PB_PTR	Power Budgeting	`PB_NEXT_PTR
`ARI_PTR	Alternate Routing-ID (ARI)	`ARI_NEXT_PTR
`SRIOV_PTR	Single Root I/O Virtualization (SR-IOV)	`SRIOV_NEXT_PTR

NOTE: These parameters not visible in coreConsultant.

The following tables outline the PCI Express Extended Capabilities structures. Refer to the indicated pages for detailed register descriptions.

Table 4-10 PF Advanced Error Reporting (AER) Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`AER_PTR	PCI Express Extended Capability Header				389
+0x4	Uncorrectable Error Status Register				389
+0x8	Uncorrectable Error Mask Register				390
+0xC	Uncorrectable Error Severity Register				391
+0x10	Correctable Error Status Register				391
+0x14	Correctable Error Mask Register				392
+0x18	Advanced Error Capabilities and Control Register				392
+0x1C through +0x28	Header Log Registers				393

Table 4-11 PF Virtual Channel Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VC_PTR	PCI Express Extended Capability Header				394
+0x4	Port VC Capability Register 1				394
+0x8	Port Capability Register 2				395
+0xC	Port VC Status Register		Port VC Control Register		395
+0x10	VC Resource Capability Register (0)				396
+0x14	VC Resource Control Register (0)				396
+0x18	VC Resource Status Register (0)	RsvdP			397
0x10+(N*0x0C)	VC Resource Capability Register (N) ¹				396
0x14+(N*0x0C)	VC Resource Control Register (N)				396
0x18+(N*0x0C)	VC Resource Status Register (N)	RsvdP			397

1. There is one VC Resource Capability/Control/Status Register N set for each configured VC (in addition to VC0).

Table 4-12 PF Device Serial Number Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`SN_PTR ¹	Device Serial Number Extended Capability Header				400

1. Depends on the number of VCs

Table 4-12 PF Device Serial Number Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0x4	Serial Number Register (1st DWORD)				400
+0x8	Serial Number Register (2nd DWORD)				400

1. Depends on the number of VCs

Table 4-13 PF Power Budgeting Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`PB_PTR	Power Budgeting Extended Capability Header				401
	Data Select Register				402
	Data Register				402
	Power Budget Capability Register				403

Table 4-14 PF Alternate Routing-ID (ARI) Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`ARI_PTR	ARI Capability Header				404
+0x04	ARI Control Register		ARI Capability Register		404

Table 4-15 PF Single Root I/O Virtualization (SR-IOV) Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`SRIOV_PTR	SRIOV Capability Header				406
+0x04	SRIOV Capabilities				407
+0x08	SRIOV Status		SRIOV Control		407
+0x0C	TotalVFs		InitialVFs		407
+0x10	RsvdP	Function Dependency Link	NumVFs		408
+0x14	VF Stride		First VF Offset		408
+0x18	VF Device ID		RsvdP		409
+0x1C	Supported Page Sizes				409
+0x20	System Page Size				409



Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0x24	VF BAR0				409
+0x28	VF BAR1				409
+0x2C	VF BAR2				410
+0x30	VF BAR3				410
+0x34	VF BAR4				410
+0x38	VF BAR5				410
+0x3C	VF Migration State Array Offset				410



4.1.3 VF Register Maps

Capability configuration registers are in structures (groups) identified by a Capability ID. Groups are linked together as in PCI. Register locations within a group are specified, but the starting location of each group must be found by traversing the linked list. There is one linked lists of register groups for Virtual Functions (VFs), which is the group of PCI Compatible Capability registers. The PCI Compatible Capability register groups contain two capabilities, PCI Express and MSI-X, with PCI Express being first in the list. PCI Compatible Capability register groups begin at the configuration address stored in the capability pointer register at 0x34.

4.1.3.1 VF PCI Configuration Space Header – Type 0

Table 4-16 shows the configuration field register definitions for a Virtual Function PCI Express Type 0 Configuration Space header. Most PCI-compatible register fields have the same software interpretation in PCI 3.0 and PCI Express. Refer to the indicated page numbers for detailed register descriptions.

Table 4-16 VF PCI Configuration Space Header – Type 0

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x00	Device ID		Vendor ID		416
0x04	Status Register		Command Register		416
0x08	Class Code			Revision ID	418
0x0C	BIST(0x00)	Header Type	Latency Timer	Cache Line Size	418
0x10	Base Address Register 0				419
0x14	Base Address Register 1				419
0x18	Base Address Register 2				419
0x1C	Base Address Register 3				419
0x20	Base Address Register 4				419
0x24	Base Address Register 5				419
0x28	CardBus CIS Pointer				419
0x2C	Subsystem ID		Subsystem Vendor ID		419
0x30	Expansion ROM Base Address				420
0x34	Reserved			CapPtr	420
0x38	Reserved				
0x3C	Max_Latency ¹	Min_Grant ¹	Interrupt Pin	Interrupt Line	420

1. The Max_Latency and Min_Grant registers do not apply to PCI Express and are read-only registers with values hardwired to 0x00.

4.1.3.2 VF Capability Structure Register Maps



All VFs share a single capabilities structure linked list.

The Capability Pointer register in the VF PCI Register Maps points to the next item in the linked list of capabilities, which, by default, is the PCI Express Capabilities register space.

There is one linked lists of register groups for Virtual Functions (VFs), which is the group of PCI Compatible Capability registers. The PCI Compatible Capability register groups contain two capabilities, PCI Express and MSI-X, with PCI Express being first in the list. PCI Compatible Capability register groups begin at the configuration address stored in the capability pointer register at 0x34.

Table 4-17 lists the capabilities supported by the core and their respective default address offsets and next capability pointers. The starting addresses are defined by parameters that are not visible in coreConsultant. **Table 4-18** provides the default values of these parameters.

Table 4-17 VF Configuration Structure: Starting Addresses and Next Capability Pointers

Start Address Offset	Item	Next Pointer
0x00	PCI-Compatible Header (Type 0)	`VF_CFG_NEXT_PTR
`VF_CFG_PCIE_CAP	PCI Express Capabilities	`VF_PCIE_NEXT_PTR
`VF_CFG_MSIX_CAP	MSI-X	`VF_MSIX_NEXT_PTR

NOTE: These parameters not visible in coreConsultant.

Table 4-18 VF Default Values of Parameters that Define Starting Addresses

Parameter	Default Value
`VF_CFG_PCIE_CAP	8'h70
`VF_CFG_MSIX_CAP	8'hB0

The following tables show the PCI Standard Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-19 VF PCI Express Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VF_CFG_PCIE_CAP	PCI Express Capabilities Register		Next Capability Pointer (`VF_PCIE_NEXT_PTR)	Capability ID (0x10)	422
+0x4	Device Capabilities				423

Table 4-19 VF PCI Express Capability Structure (Continued)

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0x8	Device Status		Device Control		424
+0xC	Link Capabilities				425
+0x10	Link Status		Link Control		426
+0x24	Device Capabilities 2				428
+0x28			Device Control 2		428
+0x30	Link Status 2		Link Control 2		429

Table 4-20 VF MSI-X Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VF_CFG_MSIX_C AP	MSI-X Control Register		Next Capability Pointer (`VF_MSIX_NEXT_P TR)	Capability ID (0x11)	432
+0x4	Table Offset (31:3) and BIR (2:0)				433
+0x8	PBA Offset (31:3) and BIR (2:0)				434

4.1.4 Accessing Configuration Registers

The application can access the configuration space through the DBI. Bits [11:0] of the DBI address bus select the target register. Bits [18:16] of the DBI address bus select the target physical function when not configured to support SR-IOV. When SR-IOV is supported, the physical function is indicated on the dbi_func_num and the virtual function is indicated on dbi_vfunc_num. Host software accesses the configuration registers through PCI Express Configuration Requests.

The Attribute (Attr) column in each register description indicates the read/write access for the register or bit. [Table 4-21](#) defines the read/write attribute abbreviations that are used in the register and bit descriptions throughout this chapter.

Table 4-21 Configuration Register Bit-Field Types

Attribute	Description
HwInit	Hardware Initialized HwInit bits are controlled by core hardware and are read-only by host system software. Some HwInit bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions. If `CX_DBI_RO_WR_EN = 0, none of the HwInit bits are writable through the DBI.
RO	Read-Only Some RO bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions. If `CX_DBI_RO_WR_EN = 0, none of the RO bits are writable through the DBI.
RW	Read-Write Register bits are read-write and may be read and written normally by the host and the application. Writing from the application side (if any) requires careful synchronization with the host software.
RW1C	Read-Only Status/Write-1-to-Clear Status Register Register bits indicate status when read. A set bit indicating a status event may be cleared by writing a 1. Writing 0 to RW1C bits has no effect. Writing from the application side (if any) requires careful synchronization with host software.
ROS	Sticky Read-Only Registers are read-only and cannot be altered by host or application software, except as noted. Registers are not initialized or modified by a hot reset. A few bits designated as very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.
RWS	Sticky Read-Write Registers are read-write and may be either set or cleared by host or application software to the desired state. Bits are not initialized or modified by a hot reset. A few bits designated very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.

**Table 4-21 Configuration Register Bit-Field Types (Continued)**

Attribute	Description
RW1CS	Sticky Read-Only Status/Write-1-to-Clear Status A 1-write clears status for the host and the application, except as noted. Registers indicate status when read. A set bit indicating a status event may be cleared by writing a 1. Writing 0 to RW1CS bits has no effect. Bits are not initialized or modified by a hot reset. A few bits designated very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.
RsvdP	Reserved and Preserved Reserved for future RW implementations: software must preserve the value read when writing to other bits in the same register.
RsvdZ	Reserved and Zero Reserved for future RW1C implementations: software must write 0 to these bits when writing to other bits in the same register.



Note Some RO bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions.



4.1.5 Register Default Values

You can select the default reset values of several of the configuration registers through design configuration parameters. The register descriptions in this chapter indicate the default reset value of the register, either as a specific numerical value or as the name of the configuration parameter that sets the default reset value.

For several of the configuration registers, you can set the default reset values on a per-function basis. Configuration parameters that apply to specific functions include _N at the end of the parameter name. For example, the `CX_DEVICE_ID_0 parameter sets the default value of the Device ID register for function 0, `CX_DEVICE_ID_1 sets the default of the Device ID register for function 1, and so on for each function in your core configuration.

4.1.6 PF PCI-Compatible Configuration Header Register Details

4.1.6.1 PF PCI-Compatible Configuration Space Header – Type 0

Table 4-22 shows the configuration field register definitions for PCI Express Type 0 Configuration Space header. Most PCI-compatible register fields have the same software interpretation in PCI 3.0 and PCI Express. Refer to the indicated page numbers for detailed register descriptions.

Table 4-22 PF PCI-Compatible Configuration Space Header – Type 0

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x00	Device ID		Vendor ID		336
0x04	Status Register		Command Register		337
0x08	Class Code			Revision ID	339
0x0C	BIST(0x00)	Header Type	Latency Timer	Cache Line Size	342
0x10	Base Address Register 0				345
0x14	Base Address Register 1				345
0x18	Base Address Register 2				345
0x1C	Base Address Register 3				345
0x20	Base Address Register 4				345
0x24	Base Address Register 5				345
0x28	CardBus CIS Pointer				358
0x2C	Subsystem ID		Subsystem Vendor ID		359
0x30	Expansion ROM Base Address				360
0x34	Reserved			CapPtr	358
0x38	Reserved				
0x3C	Max_Latency ¹	Min_Grant ¹	Interrupt Pin	Interrupt Line	362

1. The Max_Latency and Min_Grant registers do not apply to PCI Express and are read-only registers with values hardwired to 0x00.

4.1.6.2 Device ID and Vendor ID Register

Offset: 0x00

[Table 4-23](#) defines the Device ID and Vendor ID register bit assignments. The default values of both Device ID and Vendor ID are hardware configuration parameters. The application can overwrite the default values of both Device ID and Vendor ID through the DBI.

Table 4-23 Device ID and Vendor ID Register

Bits	Default	Attr	Description
15:0	`CX_VENDOR_ID_N	RO	Vendor ID, writable through the DBI
31:16	`CX_DEVICE_ID_N	RO	Device ID, writable through the DBI



4.1.6.3 Command Register

Offset: 0x04

Bytes: 0–1

Table 4-24 defines the PCI-compatible Command register bit assignments.

Table 4-24 Command Register

Bits	Default	Attr	Description
0	0	RW	I/O Space Enable
1	0	RW	Memory Space Enable
2	0	RW	Bus Master Enable
3	0	RO	Special Cycle Enable Not applicable for PCI Express. Must be hardwired to 0.
4	0	RO	Memory Write and Invalidate Not applicable for PCI Express. Must be hardwired to 0.
5	0	RO	VGA Palette Snoop Not applicable for PCI Express. Must be hardwired to 0.
6	0	RW	Parity Error Response
7	0	RO	IDSEL Stepping/Wait Cycle Control Not applicable for PCI Express. Must be hardwired to 0
8	0	RW	SERR# Enable
9	0	RO	Fast Back-to-Back Enable Not applicable for PCI Express. Must be hardwired to 0.
10	0	RW	INTx Assertion Disable
15:11	0x0	RO	Reserved



4.1.6.4 Status Register

Offset: 0x04

Bytes: 2–3

Table 4-25 defines the PCI-compatible Status register bit assignments.

Table 4-25 Status Register

Bits	Default	Attr	Description
2:0	0x0	RO	Reserved
3	0	RO	INTx Status
4	1	RO	Capabilities List Indicates presence of an extended capability item. Hardwired to 1.
5	0	RO	66 MHz Capable Not applicable for PCI Express. Hardwired to 0.
6	0	RO	Reserved
7	0	RO	Fast Back-to-Back Capable Not applicable for PCI Express. Hardwired to 0.
8	0	RW1C	Master Data Parity Error
10:9	0x0	RO	DEVSEL Timing Not applicable for PCI Express. Hardwired to 0.
11	0	RW1C	Signaled Target Abort
12	0	RW1C	Received Target Abort
13	0	RW1C	Received Master Abort
14	0	RW1C	Signaled System Error
15	0	RW1C	Detected Parity Error



4.1.6.5 Revision ID Register

Offset: 0x08

Byte: 0

Table 4-26 Revision ID Register

Bits	Default	Attr	Description
7:0	`CX_REVISION_ID_N	RO	Revision ID, writable through the DBI



4.1.6.6 Class Code Register

Offset: 0x08

Bytes: 1–3

Table 4-27 Class Code Register

Bits	Default	Attr	Description
7:0	`IF_CODE_N	RO	Programming Interface, writable through the DBI
15:8	`SUB_CLASS_CODE_N	RO	Subclass Code, writable through the DBI
23:16	`BASE_CLASS_CODE_N	RO	Base Class Code, writable through the DBI



4.1.6.7 Cache Line Size Register

Offset: 0x0C

Byte: 0

Table 4-28 Cache Line Size Register

Bits	Default	Attr	Description
7:0	0x00	RW	<p>Cache Line Size</p> <p>The Cache Line Size register is RW for legacy compatibility purposes and is not applicable to PCI Express device functionality. Writing to the Cache Line Size register does not impact functionality of the core.</p>



4.1.6.8 Master Latency Timer Register

Offset: 0x0C

Byte: 1

Table 4-29 Master Latency Timer Register

Bits	Default	Attr	Description
7:0	0x00	RO	Master Latency Timer Not applicable for PCI Express, hardwired to 0.



4.1.6.9 Header Type Register

Offset: 0x0C

Byte: 2

Table 4-30 Header Type Register

Bits	Default	Attr	Description
6:0	0x0	RO	Configuration Header Format Hardwired to 0 for type 0.
7	(`CX_NFUNC != 1)	RO	Multi Function Device The default value is 0 for a single function device (`CX_NFUNC = 1) or 1 for a multi-function device (`CX_NFUNC != 1). The Multi Function Device bit is writable through the DBI.



4.1.6.10 BIST Register

Offset: 0x0C

Byte: 3

Table 4-31 BIST Register

Bits	Default	Attr	Description
7:0	0x00	RO	The BIST register functions are not supported by the core. All 8 bits of the BIST register are hardwired to 0.



4.1.6.11 Base Address Registers

Offset: 0x10–0x24

The core provides three pairs of 32-bit BARs for each implemented function. Each pair (BARs 0 and 1, BARs 2 and 3, BARs 4 and 5) can be configured as follows:

- ❖ One 64-bit BAR: For example, BARs 0 and 1 are combined to form a single 64-bit BAR.
- ❖ Two 32-bit BARs: For example, BARs 0 and 1 are two independent 32-bit BARs.
- ❖ One 32-bit BAR: For example, BAR 0 is a 32-bit BAR and BAR 1 is either disabled or removed from the core altogether to reduce gate count.

In addition, you can configure each BAR to have its incoming Requests routed to either:

- ❖ RTRGT1
- ❖ RTRGT0 for local application register access on the ELBI

The following sections describe how to set up the BAR types and sizes by programming values into the base address registers. For more information about routing Requests to either RTRGT1 or RTRGT0 on a BAR-by-BAR basis, see “[Receive Filtering](#)” on page [60](#).

4.1.6.11.1 General Rules for BAR Setup

The contents of the six BARs determine the BAR configuration. The reset values of the BARs are determined by hardware configuration options.

At runtime, application software can overwrite the BAR contents to reconfigure the BARs (unless the affected BAR is removed during hardware configuration). Application software must observe the rules listed below when writing to the BARs.

The rules for BAR configuration are the same for all three pairs. Using BARs 0 and 1 as the example pair, the rules for BAR configuration are:

- ❖ Any pair (for example, BARs 0 and 1) can be configured as one 64-bit BAR, two 32-bit BARs, or one 32-bit BAR.
- ❖ BAR pairs cannot overlap to form a 64-bit BAR. For example, you cannot combine BARs 1 and 2 to form a 64-bit BAR.
- ❖ Any 32-bit BAR that is not needed can be removed during core hardware configuration to reduce gate count.
- ❖ An I/O BAR must be a 32-bit BAR and cannot be prefetchable.
- ❖ If the device is configured as a PCI Express Endpoint (not a Legacy Endpoint), then any memory that is configured as prefetchable must be a 64-bit memory BAR.
- ❖ If BAR 0 is configured as a 64-bit BAR:
 - ◆ BAR 1 is the upper 32 bits of the combined 64-bit BAR formed by BARs 0 and 1. Therefore, BAR 1 must be disabled and cannot be configured independently.
 - ◆ BAR 0 must be a memory BAR and can be either prefetchable or non-prefetchable.
 - ◆ The contents of the BAR 0 Mask register determine the number of writable bits in the 64-bit BAR, subject to the restrictions described in “[BAR Mask Registers](#)” on page [350](#). The BAR 1 Mask register contains the upper 32 bits of the BAR 0 Mask value.
 - ◆ BAR 0 can be disabled by writing 0 to bit 0 of the BAR 0 Mask register (if `BAR0_MASK_WRITABLE_N = 1`).

- ❖ If BAR 0 is configured as a 32-bit BAR:
 - ◆ You can configure BAR 1 as an independent 32-bit BAR or remove BAR 1 from the core hardware configuration.
 - ◆ BAR 0 can be configured as a memory BAR or an I/O BAR.
 - ◆ The contents of the BAR 0 Mask register determine the number of writable bits in the 32-bit BAR 0, subject to the restrictions described in “[BAR Mask Registers](#)” on page 350.
 - ◆ BAR 0 can be disabled by writing 0 to bit 0 of the BAR 0 Mask register (if `BAR0_MASK_WRITABLE_N = 1).
- ❖ When BAR 0 is configured as a 32-bit BAR, BAR 1 is available as an independent 32-bit BAR according to the following rules:
 - ◆ BAR 1 can be configured as a memory BAR or an I/O BAR.
 - ◆ The contents of the BAR 1 Mask register determine the number of writable bits in the 32-bit BAR 1, subject to the restrictions described in “[BAR Mask Registers](#)” on page 350.
 - ◆ BAR 1 can be disabled by writing 0 to bit 0 of the BAR 1 Mask register (if `BAR1_MASK_WRITABLE_N = 1).
 - ◆ If BAR 1 is not required in your design, you can remove BAR 1 from the hardware configuration by setting both `BAR1_ENABLED_N and `BAR1_MASK_WRITABLE_N to 0.

The same rules apply for pairs 2/3 and 4/5.

4.1.6.11.2 Base Address Register 0 (Optional)

Offset: 0x10 (if included in the core hardware configuration)

Table 4-32 Base Address Register 0

Bits	Default	Attr	Description
31:4	0x0000000	RO	BAR 0 base address bits (for a 64-bit BAR, the remaining upper address bits are in BAR 1). The BAR 0 Mask value determines which address bits are masked.
3	`PREFETCHABLE0_N for memory BAR 0 for I/O BAR	RO	If BAR 0 is a memory BAR, bit 3 indicates if the memory region is prefetchable: <ul style="list-style-type: none"> • 0 = Non-prefetchable • 1 = Prefetchable If BAR 0 is an I/O BAR, bit 3 is the second least significant bit of the base address. Bits [3:0] are writable through the DBI.
2:1	`BAR0_TYPE_N for memory BAR 00 for I/O BAR	RO	If BAR 0 is a memory BAR, bits [2:1] determine the BAR type: <ul style="list-style-type: none"> • 00 = 32-bit BAR • 10 = 64-bit BAR If BAR 0 is an I/O BAR, bit 2 the least significant bit of the base address and bit 1 is 0. Bits [3:0] are writable through the DBI.

Table 4-32 Base Address Register 0 (Continued)

Bits	Default	Attr	Description
0	`MEM0_SPACE_DECODER_N	RO	<ul style="list-style-type: none"> • 0 = BAR 0 is a memory BAR • 1 = BAR 0 is an I/O BAR Bits [3:0] are writable through the DBI.

4.1.6.11.3 Base Address Register 1 (Optional)

Address: 0x14 (if included in the core hardware configuration)

Table 4-33 Base Address Register 1

Bits	Default	Attr	Description
31:0	Configuration-dependent	RO	If BAR 0 is a 64-bit BAR, BAR 1 contains the upper 32 bits of the BAR 0 base address (bits [63:32]). If BAR 0 is a 32-bit BAR, BAR 1 can be independently programmed as an additional 32-bit BAR or can be excluded from the core hardware configuration. If programmed as an independent 32-bit BAR, the BAR 1 bit definitions are the same as the BAR 0 bit definitions.

4.1.6.11.4 Base Address Register 2 (Optional)

Offset: 0x18 (if included in the core hardware configuration)

Table 4-34 Base Address Register 2

Bits	Default	Attr	Description
31:4	0x00000000	RO	BAR 2 base address bits (for a 64-bit BAR, the remaining upper address bits are in BAR 3). The BAR 2 Mask value determines which address bits are masked.
3	`PREFETCHABLE2_N for memory BAR 0 for I/O BAR	RO	If BAR 2 is a memory BAR, bit 3 indicates if the memory region is prefetchable: <ul style="list-style-type: none"> • 0 = Non-prefetchable • 1 = Prefetchable If BAR 2 is an I/O BAR, bit 3 is the second least significant bit of the base address. Bits [3:0] are writable through the DBI.
2:1	`BAR2_TYPE_N for memory BAR 00 for I/O BAR	RO	If BAR 2 is a memory BAR, bits [2:1] determine the BAR type: <ul style="list-style-type: none"> • 00 = 32-bit BAR • 10 = 64-bit BAR If BAR 2 is an I/O BAR, bit 2 the least significant bit of the base address and bit 1 is 0. Bits [3:0] are Writable through the DBI.

Table 4-34 Base Address Register 2 (Continued)

Bits	Default	Attr	Description
0	`MEM2_SPACE_DECODER_N	RO	<ul style="list-style-type: none"> • 0 = BAR 2 is a memory BAR • 1 = BAR 2 is an I/O BAR • Bits [3:0] are writable through the DBI.

4.1.6.11.5 Base Address Register 3 (Optional)

Address: 0x1C (if included in the core hardware configuration)

Table 4-35 Base Address Register 3

Bits	Default	Attr	Description
31:0	Configuration-dependent	RO	<p>If BAR 2 is a 64-bit BAR, BAR 3 contains the upper 32 bits of the BAR 2 base address (bits [63:32]).</p> <p>If BAR 2 is a 32-bit BAR, BAR 3 can be independently programmed as an additional 32-bit BAR or can be excluded from the core hardware configuration.</p> <p>If programmed as an independent 32-bit BAR, the BAR 3 bit definitions are the same as the BAR 2 bit definitions.</p>

4.1.6.11.6 Base Address Register 4 (Optional)

Offset: 0x20 (if included in the core hardware configuration)

Table 4-36 Base Address Register 4

Bits	Default	Attr	Description
31:4	0x0000000	RO	BAR 4 base address bits (for a 64-bit BAR, the remaining upper address bits are in BAR 5). The BAR 4 Mask value determines which address bits are masked.
3	`PREFETCHABLE4_N for memory BAR 0 for I/O BAR	RO	<p>If BAR 4 is a memory BAR, bit 3 indicates if the memory region is prefetchable:</p> <ul style="list-style-type: none"> • 0 = Non-prefetchable • 1 = Prefetchable <p>If BAR 4 is an I/O BAR, bit 3 is the second least significant bit of the base address.</p> <p>Bits [3:0] are writable through the DBI.</p>
2:1	`BAR4_TYPE_N for memory BAR 00 for I/O BAR	RO	<p>If BAR 4 is a memory BAR, bits [2:1] determine the BAR type:</p> <ul style="list-style-type: none"> • 00 = 32-bit BAR • 10 = 64-bit BAR <p>If BAR 4 is an I/O BAR, bit 2 the least significant bit of the base address and bit 1 is 0.</p> <p>Bits [3:0] are writable through the DBI.</p>

Table 4-36 Base Address Register 4 (Continued)

Bits	Default	Attr	Description
0	`MEM4_SPACE_DECODER_N	RO	<ul style="list-style-type: none"> • 0 = BAR 4 is a memory BAR • 1 = BAR 4 is an I/O BAR Bits [3:0] are writable through the DBI.

4.1.6.11.7 Base Address Register 5 (Optional)

Address: 0x24 (if included in the core hardware configuration)

Table 4-37 Base Address Register 5

Bits	Default	Attr	Description
31:0	Configuration-dependent	RO	<p>If BAR 4 is a 64-bit BAR, BAR 5 contains the upper 32 bits of the BAR 4 base address (bits 63:32).</p> <p>If BAR 4 is a 32-bit BAR, BAR 5 can be independently programmed as an additional 32-bit BAR or can be excluded from the core hardware configuration.</p> <p>If programmed as an independent 32-bit BAR, the BAR 5 bit definitions are the same as the BAR 4 bit definitions.</p>

4.1.6.12 BAR Mask Registers

The BAR masks are used for indicating the amount of memory each BAR requests from host software. Their default values are derived from the corresponding coreConsultant parameters; e.g. `BAR0_MASK_0. The application logic can overwrite the default values via the DBI.

The BAR Mask registers determine which bits in each BAR are non-writable by host software, which determines the size of the address space claimed by each BAR.

A BAR Mask register exists only if the corresponding `BARn_MASK_WRITABLE_N value is 1. Otherwise, the `BARn_MASK_N value sets the BAR Mask value in hardware.

The BAR Mask values indicate the range of low-order bits in each implemented BAR *not to use* for address matching. The BAR Mask value also indicates the range of low-order bits in the BAR that *cannot be written* from the host. The application can write to all BAR bits to allow setting of memory, I/O, and other standard BAR options.

To disable any BAR, the application can write a 0 to bit 0 of the corresponding BAR Mask register. To change the BAR Mask value for a disabled BAR, the application must first enable the BAR by writing 1 to bit 0. After enabling the BAR, the application can then write a new value to the BAR Mask register.

The BAR Mask registers are accessible through the same address as the corresponding BAR registers, but requires dbi_cs2 assertions instead. The BAR Mask registers are writable only, not readable.

If the BAR Mask value for a BAR is less than that required for the BAR type, the core automatically uses the minimum value for the BAR type:

- ❖ BAR bits [11:0] are always masked for a memory BAR. The core requires each memory BAR to claim at least 4 KB.
- ❖ BAR bits [7:0] are always masked for an I/O BAR. The core requires each I/O BAR to claim at least 256 bytes.

The BAR Mask registers are accessible through the same address as the corresponding BAR registers, but BAR Mask registers requires dbi_cs2 assertions instead. The BAR Mask registers are writable only, not readable.

The aperture of the BAR is actually the larger of the written size (or default vf_bar*_mask) or the system page size (set by the operating system). In the case where the system page size is larger than the requested bar size, the BAR is actually sized to the system page size (see *Single Root I/O Virtualization and Sharing Specification (SR-IOV)*, paragraph 2 on page 46). This means that when the OS writes all 1s then reads back to determine the size of the BAR, the OS will see the BAR size to be the system page size. The application logic, most likely, will only have the original requested amount of physical memory. A transaction will receive a UR if the transaction is from the RC and it targets an address that is within the range of the allocated system page size but above the implemented application memory.

4.1.6.12.1 BAR 0 Mask Register

Offset: 0x10 (same as Base Address Register 0, but requires dbi_cs2 for write access)

Table 4-38 BAR 0 Mask Register

Bits	Default	Attr	Description
31:1	`BAR0_MASK_N	Appl. write access only; not readable	<p>Indicates which BAR 0 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the BAR 0 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 0 is a 64-bit BAR, then the BAR 1 Mask register contains the upper bits of the BAR 0 Mask. The BAR 0 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `BAR0_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `BAR0_MASK_WRITABLE_N = 1, the BAR 0 Mask register is writable through the DBI. • If `BAR0_MASK_WRITABLE_N = 0, the BAR 0 Mask register is not writable through the DBI.
0	`BAR0_ENABLED_N	Same as bits [31:1]	<p>BAR 0 Enable</p> <ul style="list-style-type: none"> • 0: BAR 0 is disabled • 1: BAR 0 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.1.6.12.2 BAR 1 Mask Register

Offset: 0x14 (same as Base Address Register 1, but requires dbi_cs2 for write access)

Table 4-39 BAR 1 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 0 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 1 Mask value, interpreted the same way as BAR 0 Mask. Default value is `BAR1_MASK_N. • Bit 0: BAR 1 Enable (0 = BAR 1 is disabled; 1= BAR 1 is enabled). Default value is `BAR1_ENABLED_N. • `BAR1_MASK_WRITABLE_N controls application write access to the BAR 1 Mask register. <p>If BAR 0 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 0 Mask value. • `BAR0_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 0 Mask register.

4.1.6.12.3 BAR 2 Mask Register

Offset: 0x18 (same as Base Address Register 2, but requires dbi_cs2 for write access)

Table 4-40 BAR 2 Mask Register

Bits	Default	Attr	Description
31:1	`BAR2_MASK_N	Appl. access only	<p>Indicates which BAR 2 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the BAR 2 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 2 is a 64-bit BAR, then the BAR 3 Mask register contains the upper bits of the BAR 2 Mask.</p> <p>The BAR 2 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `BAR2_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `BAR2_MASK_WRITABLE_N = 1, the BAR 2 Mask register is writable through the DBI. • If `BAR2_MASK_WRITABLE_N = 0, the BAR 2 Mask register is not writable through the DBI.

**Table 4-40 BAR 2 Mask Register (Continued)**

Bits	Default	Attr	Description
0	`BAR2_ENABLED_N	Same as bits [31:1]	<p>BAR 2 Enable</p> <ul style="list-style-type: none">• 0: BAR 2 is disabled• 1: BAR 2 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>



4.1.6.12.4 BAR 3 Mask Register

Offset: 0x1C (same as Base Address Register 3, but requires dbi_cs2 for write access)

Table 4-41 BAR 3 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 2 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 3 Mask value, interpreted the same way as BAR 2 Mask. Default value is `BAR3_MASK_N. • Bit 0: BAR 3 Enable (0 = BAR 3 is disabled; 1= BAR 3 is enabled). Default value is `BAR3_ENABLED_N. • `BAR3_MASK_WRITABLE_N controls application write access to the BAR 3 Mask register. <p>If BAR 2 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 2 Mask value. • `BAR2_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 2 Mask register.

4.1.6.12.5 BAR 4 Mask Register

Offset: 0x20 (same as Base Address Register 4, but requires dbi_cs2 for write access)

Table 4-42 BAR 4 Mask Register

Bits	Default	Attr	Description
31:1	`BAR4_MASK_N	Appl. access only	<p>Indicates which BAR 4 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the BAR 4 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 4 is a 64-bit BAR, then the BAR 5 Mask register contains the upper bits of the BAR 4 Mask.</p> <p>The BAR 4 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `BAR4_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `BAR4_MASK_WRITABLE_N = 1, the BAR 4 Mask register is writable through the DBI. • If `BAR4_MASK_WRITABLE_N = 0, the BAR 4 Mask register is not writable through the DBI.

Table 4-42 BAR 4 Mask Register (Continued)

Bits	Default	Attr	Description
0	`BAR4_ENABLED_N	Same as bits [31:1]	<p>BAR 4 Enable</p> <ul style="list-style-type: none"> • 0: BAR 4 is disabled • 1: BAR 4 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.1.6.12.6 BAR 5 Mask Register

Offset: 0x24 (same as Base Address Register 5, but requires dbi_cs2 for write access)

Table 4-43 BAR 5 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 4 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 5 Mask value, interpreted the same way as BAR 5 Mask. Default value is `BAR5_MASK_N. • Bit 0: BAR 5 Enable (0 = BAR 5 is disabled; 1= BAR 5 is enabled). Default value is `BAR5_ENABLED_N. • `BAR5_MASK_WRITABLE_N controls application write access to the BAR 5 Mask register. <p>If BAR 4 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 4 Mask value. • `BAR4_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 4 Mask register.

4.1.6.12.7 Expansion ROM BAR Mask Register

Offset: 0x30 (same as the Expansion ROM BAR, but requires dbi_cs2 for write access)

Table 4-44 Expansion ROM BAR Mask Register

Bits	Default	Attr	Description
31:1	`ROM_MASK_N	Appl. access only	<p>Indicates which Expansion ROM BAR bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the Expansion ROM BAR Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>The maximum value is 0xFFFFFFFF because the maximum space that can be claimed by an Expansion ROM BAR is 16 MB.</p> <p>The Expansion ROM BAR Mask register is invisible to host software and not readable from the application. Application access depends on the value of `ROM_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `ROM_MASK_WRITABLE_N = 1, the Expansion ROM BAR Mask register is writable through the DBI. • If `ROM_MASK_WRITABLE_N = 0, the Expansion ROM BAR Mask register is not writable through the DBI.
0	`ROM_BAR_ENABLED_N	Same as bits [31:1]	<p>Expansion ROM BAR Enable</p> <ul style="list-style-type: none"> • 0: Expansion ROM BAR is disabled • 1: Expansion ROM BAR is enabled

4.1.6.12.8 Example BAR Setup

Figure 4-2 shows an example configuration of the six BARs and their corresponding BAR Mask registers. The example configuration includes:

- ❖ One 64-bit memory BAR (non-prefetchable)
- ❖ One 32-bit memory BAR (non-prefetchable)
- ❖ One 32-bit I/O BAR

Figure 4-2 Example Base Address Register Configuration

Base Address Registers

31		0
BAR 5 (24h)	Disabled	
BAR 4 (20h)	Writable Base Address Bits (31:8)	Masked Base Addr. Bits (7:2)
BAR 3 (1Ch)	Disabled	
BAR 2 (18h)	Writable Base Address Bits (31:20)	Masked Base Addr. Bits (19:4)
BAR 1 (14h)	Upper Address Bits of BAR 0 (non-masked)	
BAR 0 (10h)	Writable Base Address Bits (31:20)	Masked Base Addr. Bits (19:4)

32-bit I/O BAR

32-bit Memory BAR

64-bit Memory BAR

BAR Mask Registers

31		0
BAR 5 Mask (24h)	0x0	
BAR 4 Mask (20h)	0x000000FF	
BAR 3 Mask (1Ch)	0x0	
BAR 2 Mask (18h)	0x000FFFFF	
BAR 1 Mask (14h)	0x00000000	
BAR 0 Mask (10h)	0x000FFFFFFF	

4.1.6.13 CardBus CIS Pointer Register

Offset: 0x28

Table 4-45 CardBus CIS Pointer Register

Bits	Default	Attr	Description
31:0	`CARDBUS_CIS_PTR_N	RO	CardBus CIS Pointer Optional, writable through the DBI.



4.1.6.14 Subsystem ID and Subsystem Vendor ID Register

Offset: 0x2C

Table 4-46 Subsystem ID and Subsystem Vendor ID Register

Bits	Default	Attr	Description
15:0	`SUBSYS_VENDOR_ID_N	RO	Subsystem Vendor ID Writable through the DBI.
31:16	`SUBSYS_DEV_ID_N	RO	Subsystem ID Writable through the DBI.



4.1.6.15 Expansion ROM Base Address Register

Offset: 0x30

Table 4-47 Expansion ROM Base Address Register

Bits	Default	Attr	Description
31:11	0x00000	RW	Expansion ROM Address
10:1	0x000	RO	Reserved
0	0x0	RW	Expansion ROM Enable



4.1.6.16 Capability Pointer Register

Offset: 0x34

Byte: 0

Table 4-48 Capability Pointer Register

Bits	Default	Attr	Description
7:0	`CFG_NEXT_PTR	RO	First Capability Pointer. Points to Power Management Capability structure by default, writable through the DBI



4.1.6.17 Interrupt Line Register

Offset: 0x3C

Byte: 0

Table 4-49 Interrupt Line Register

Bits	Default	Attr	Description
7:0	0xFF	RW	<p>Interrupt Line</p> <p>Value in this register is system architecture specific. POST software will write the routing information into this register as it initializes and configures the system.</p>

4.1.6.18 Interrupt Pin Register

Offset: 0x3C

Byte: 1

Table 4-50 Interrupt Pin Register

Bits	Default	Attr	Description
7:0	`INT_PIN_MAPPING_N	RO	<p>Interrupt Pin</p> <p>Identifies the legacy interrupt Message that the device (or device function) uses. Valid values are:</p> <ul style="list-style-type: none"> • 00h: The device (or function) does not use legacy interrupt • 01h: The device (or function) uses INTA • 02h: The device (or function) uses INTB • 03h: The device (or function) uses INTC • 04h: The device (or function) uses INTD <p>In a single-function configuration, the core only uses INTA.</p> <p>The Interrupt Pin register is writable through the DBI.</p>

4.1.7 PF PCI Standard Capability Structures Register Details

4.1.7.1 PF PCI Power Management Capability Register Details

The core implements power management capabilities. The Capability Pointer field in the configuration header points to the PCI Power Management registers as the first extended capability by default.

The extent of the power management implementation in the core includes:

- ❖ Power Management register space
- ❖ Link state information (provided to both the application logic and PHY interfaces)
- ❖ Power management-ready clock and reset implementation

The following sections describe the PCI Power Management registers implemented in the core. See the *PCI Power Management specification* and the *PCI Express 2.0 specification* for more details.

Table 4-51 shows the PF Power Management Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-51 PF Power Management Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`CFG_PM_CAP	Power Management Capabilities (PMC)		Next Capability Pointer (`PM_NEXT_PTR)	Capability ID (0x01)	
+0x4	Data	PMCSR_BSE Bridge Extensions	Power Management Control Status Register (PMCSR)		

4.1.7.1.1 Power Management Capability ID Register

Offset: `CFG_PM_CAP

Table 4-52 Capability ID Register

Bits	Default	Attr	Description
7:0	0x01	RO	Power Management Capability ID

4.1.7.1.2 Power Management Next Item Pointer

Offset: `CFG_PM_CAP + 0x01

Table 4-53 Next Item Pointer

Bits	Default	Attr	Description
7:0	`PM_NEXT_PTR	RO	Next Capability Pointer Points to the MSI capabilities by default, writable through the DBI.

4.1.7.1.3 Power Management Capabilities Register

Offset: `CFG_PM_CAP + 0x02

Table 4-54 Power Management Capabilities Register

Bits	Default	Attr	Description
2:0	3'b011	RO	Power Management specification version, writable through the DBI
3	0	RO	PME Clock, hardwired to 0
4	0	RsvdP	Reserved
5	`DEV_SPEC_INIT_N	RO	Device Specific Initialization (DSI), writable through the DBI
8:6	`AUX_CURRENT_N	RO	AUX Current, writable through the DBI
9	`D1_SUPPORT_N	RO	D1 Support, writable through the DBI
10	`D2_SUPPORT_N	RO	D2 Support, writable through the DBI
15:11	`PME_SUPPORT_N	RO	<p>PME_Support</p> <p>Identifies the power states from which the core can generate PME Messages. A value of 0 for any bit indicates that the device (or function) is not capable of generating PME Messages while in that power state:</p> <ul style="list-style-type: none"> • Bit 11: If set, PME Messages can be generated from D0 • Bit 12: If set, PME Messages can be generated from D1 • Bit 13: If set, PME Messages can be generated from D2 • Bit 14: If set, PME Messages can be generated from D3_{hot} • Bit 15: If set, PME Messages can be generated from D3_{cold} <p>The PME_Support field is writable through the DBI.</p>

4.1.7.1.4 Power Management Control and Status Register

Offset: `CFG_PM_CAP + 0x04

Table 4-55 Power Management Control and Status Register

Bits	Default	Attr	Description
1:0	0x0	RW	<p>Power State Controls the device power state:</p> <ul style="list-style-type: none"> • 00b: D0 • 01b: D1 • 10b: D2 • 11b: D3 <p>The written value is ignored if the specific state is not supported.</p>
2	0x0	RsvdP	Reserved
3	`DEFAULT_NO_SOFT_RESET_N	RO	No Soft Reset, writable through the DBI
7:4	0x0	RsvdP	Reserved
8	0x0	RWS	<p>PME Enable (sticky bit) A value of 1 indicates that the device is enabled to generate PME.</p>
12:9	0x0	RO	Data Select (not supported)
14:13	0x0	RO	Data Scale (not supported)
15	0x0	RW1CS	<p>PME Status Indicates if a previously enabled PME event occurred or not.</p>
21:16	0x0	RsvdP	Reserved
22	0x0	RO	B2/B3 Support, hardwired to 0
23	0x0	RO	Bus Power/Clock Control Enable, hardwired to 0
31:24	0x00	RO	Data register for additional information (not supported)

4.1.7.2 PF MSI Capability Register Details

The MSI Capability structure is required for all PCI Express devices that are capable of generating interrupts. The definition of the MSI register structure is compatible with the PCI 3.0 specification.

[Table 4-56](#) shows the PF MSI Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-56 PF MSI Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`CFG_MSI_CAP	Message Control Register		Next Capability Pointer (`MSI_NEXT_PTR)	Capability ID (0x05)	
+0x4	MSI Lower 32-bit Address Register				
+0x8	MSI Upper 32-bit Address Register				
+0xC	Reserved		MSI Data		
+0x10	Mask Bits Register				
+0x14	Pending Bits Register				

4.1.7.2.1 MSI Capability ID

Offset: `CFG_MSI_CAP + 0x00

Table 4-57 MSI Capability ID Register

Bits	Default	Attr	Description
7:0	0x05	RO	MSI Capability ID

4.1.7.2.2 MSI Next Item Pointer

Offset: `CFG_MSI_CAP + 0x01

Table 4-58 MSI Next Item Pointer

Bits	Default	Attr	Description
7:0	'	RO	Next Capability Pointer Points to PCI Express Capabilities by default, writable through the DBI.

4.1.7.2.3 MSI Control Register

Offset: `CFG_MSI_CAP + 0x02

Table 4-59 MSI Control Register

Bits	Default	Attr	Description
0	0	RW	MSI Enabled When set, INTx must be disabled.
3:1	`DEFAULT_MULTI_MSI_CAPABLE	RO	Multiple Message Capable, writable through the DBI
6:4	0x0	RW	Multiple Message Enabled Indicates that multiple Message mode is enabled by system software. The number of Messages enabled must be less than or equal to the Multiple Message Capable value.
7	`MSI_64_EN	RO	64-bit Address Capable, writable through the DBI
8	`PVM_EN	RO	MSI Per Vector Masking (PVM) supported
15:8	0x00	RO	Reserved

4.1.7.2.4 MSI Lower 32 Bits Address Register

Offset: `CFG_MSI_CAP + 0x04

Table 4-60 MSI Lower 32 Bits Address Register

Bits	Default	Attr	Description
1:0	0x0	RO	Reserved
31:2	0x00000000	RW	Lower 32-bit Address

4.1.7.2.5 MSI Upper 32 bits Address Register

Offset: `CFG_MSI_CAP + 0x08 (if `MSI_64_EN = 1)

Table 4-61 MSI Upper 32 bits Address Register

Bits	Default	Attr	Description
31:0	0x00000000	RW	Upper 32-bit Address Optional, used only if `MSI_64_EN = 1. If `MSI_64_EN = 0, then the Upper 32-bit Address register functions as the MSI Data register and register address `CFG_MSI_CAP + 0x0C is not used.

4.1.7.2.6 MSI Data Register

Offset: `CFG_MSI_CAP + 0x0C if `MSI_64_EN = 1;
`CFG_MSI_CAP + 0x08 if `MSI_64_EN = 0

Table 4-62 MSI Data Register

Bits	Default	Attr	Description
15:0	0x0000	RW	MSI Data Pattern assigned by system software, bits [4:0] are OR-ed with MSI_VECTOR to generate 32 MSI Messages per function.
31:16	0x0000	RO	Reserved

4.1.7.2.7 MSI Mask Bits Register

Offset: `CFG_MSI_CAP + 0x10 if `MSI_64_EN = 1;
`CFG_MSI_CAP + 0x0C if `MSI_64_EN = 0

Table 4-63 MSI Mask Bit Register

Bits	Default	Attr	Description
31:0	0x0	RW	MSI Mask For each Mask bit that is set, the function is prohibited from sending the associated message.

4.1.7.2.8 MSI Pending Bits Register

Offset: `CFG_MSI_CAP + 0x14 if `MSI_64_EN = 1;
`CFG_MSI_CAP + 0x10 if `MSI_64_EN = 0

Table 4-64 MSI Pending Bits Register

Bits	Default	Attr	Description
31:0	0x0	RO	MSI Pending Bits For each Pending bit that is set, the function has a pending associated message.

4.1.7.3 PF PCI Express Capability Register Details

The core implements the PCI Express Capability Structure as defined in the *PCI Express 2.0 Specification* except for Root Port registers.

Table 4-65 shows the PF PCI Express Standard Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-65 PF PCI Express Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`CFG_PCIE_CAP	PCI Express Capabilities Register		Next Capability Pointer (`PCIE_NEXT_PTR)	Capability ID (0x10)	
+0x4	Device Capabilities				
+0x8	Device Status		Device Control		
+0xC	Link Capabilities				
+0x10	Link Status		Link Control		
+0x24	Device Capabilities 2				
+0x28			Device Control 2		
+0x30	Link Status 2		Link Control 2		

4.1.7.3.1 PCI Express Capability List Register

Offset: `CFG_PCIE_CAP + 0x00

Table 4-66 PCI Express Capability List Register

Bits	Default	Attr	Description
7:0	0x10	RO	PCI Express Capability ID
15:8	PCIE_NEXT_PTR	RO	Next Capability Pointer Points to the MSI-X capability by default, writable through the DBI.



4.1.7.3.2 PCI Express Capabilities Register

Offset: `CFG_PCIE_CAP + 0x02

Table 4-67 PCI Express Capabilities Register

Bits	Default	Attr	Description
3:0	0x2	RO	PCI Express Capability Version
7:4	`CX_DEVICE_TYPE	RO	Device Port Type
8	0x0	HwInit	Slot Implemented This bit is writable through the DBI. However, it must 0 for an Endpoint device. Therefore, the application must not write a 1 to this bit.
13:9	`PCIE_CAP_INT_MSG_NUM_N	RO	Interrupt Message Number Updated by hardware, writable through the DBI.
15:14	0x0	RO	RsvdP



4.1.7.3.3 Device Capabilities Register

Offset: `CFG_PCIE_CAP + 0x04

Table 4-68 Device Capabilities Register

Bits	Default	Attr	Description
2:0	Automatically derived from the value you specify for `CX_MAX_MTU	RO	Max_Payload_Size Supported, writable through the DBI
4:3	0x0	RO	Phantom Function Supported This field is writable through the DBI. However, Phantom Function is not supported. Therefore, the application must not write any value other than 0x0 to this field.
5	`DEFAULT_EXT_TAG_FIELD_SUPPORTED	RO	Extended Tag Field Supported This bit is writable through the DBI. However, if the core supports only 5 bits of TAG, then the application must not write a 1 to this bit because the hardware required to support more than 32 tags is not implemented.
8:6	`DEFAULT_EP_L0S_ACCPT_LATENCY	RO	Endpoint L0s Acceptable Latency, writable through the DBI
11:9	`DEFAULT_EP_L1_ACCPT_LATENCY	RO	Endpoint L1 Acceptable Latency, writable through the DBI
12	0x0	RO	Undefined for PCI Express 1.1 (Was Attention Button Present for PCI Express 1.0a)
13	0x0	RO	Undefined for PCI Express 1.1 (Was Attention Indicator Present for PCI Express 1.0a)
14	0x0	RO	Undefined for PCI Express 1.1 (Was Power Indicator Present for PCI Express 1.0a)
15	0x1	RO	Role-Based Error Reporting, writable through the DBI. Required to be set for device compliant to 1.1 spec and later.
17:16	0x0	RsvdP	Reserved
25:18	0x00	RO	Captured Slot Power Limit Value From Message from RC, upstream port only.
27:26	0x0	RO	Captured Slot Power Limit Scale From Message from RC, upstream port only.

**Table 4-68 Device Capabilities Register (Continued)**

Bits	Default	Attr	Description
28	`FLR_EN	RO	Function Level Reset Capability
31:29	0x0	RsvdP	Reserved



4.1.7.3.4 Device Control Register

Offset: `CFG_PCIE_CAP + 0x08

Table 4-69 Device Control Register

Bits	Default	Attr	Description
0	0x0	RW	Correctable Error Reporting Enable
1	0x0	RW	Non-Fatal Error Reporting Enable
2	0x0	RW	Fatal Error Reporting Enable
3	0x0	RW	Unsupported Request Reporting Enable
4	0x1	RW	Enable Relaxed Ordering
7:5	0x0	RW	Max_Payload_Size
8	0x0	RW	Extended Tag Field Enable
9	0x0	RW	Phantom Function Enable
10	0x0	RWS	AUX Power PM Enable
11	`DEFAULT_NO_SNOOP_SUPPORTED_N	RW	Enable No Snoop
14:12	0x2	RW	Max_Read_Request_Size
15	0x0	RW	Initiate FLR

4.1.7.3.5 Device Status Register

Offset: `CFG_PCIE_CAP + 0x0A

Table 4-70 Device Status Register

Bits	Default	Attr	Description
0	0	RW1C	Correctable Error Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
1	0	RW1C	Non-Fatal Error detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
2	0	RW1C	Fatal Error Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
3	0	RW1C	Unsupported Request Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.

**Table 4-70 Device Status Register (Continued)**

Bits	Default	Attr	Description
4	0	RO	Aux Power Detected From sys_aux_pwr_det input port.
5	0	RO	Transaction Pending Set to 1 when Non-Posted Requests are not yet completed and clear when they are completed.
15:6	0x000	RsvdZ	Reserved

4.1.7.3.6 Link Capabilities Register

Offset: `CFG_PCIE_CAP + 0x0C

Table 4-71 Link Capabilities Register

Bits	Default	Attr	Description
3:0	0x1	RO	Maximum Link Speed Default value is 0x1 for 2.5 Gbps Link. This field is writable through the DBI. However, 0x1 is the only supported value. Therefore, the application must not write any value other than 0x1 to this field. Note: For cores supporting Gen2 speed, the following values are accepted: 0001b: 2.5 GHz supported 0010b: 5.0 GHz and 2.5 GHz supported
9:4	`CX_NL	RO	Maximum Link Width Default value is the value you specify during hardware configuration (x1, x4, x8, or x16), writable through the DBI.
11:10	`AS_LINK_PM_SUPT	RO	Active State Link PM Support Default value is the value you specify during hardware configuration, writable through the DBI.
14:12	`DEFAULT_L0S_EXIT_LATENCY	RO	L0s Exit Latency The default value is the value you specify during hardware configuration, writable through the DBI. This parameter is not visible in coreConsultant.
14:12	`DEFAULT_COMM_L0S_EXIT_LATENCY	RO	L0s Exit Latency; when common clock is used The default value is the value you specify during hardware configuration, writable through the DBI2. This parameter is not visible in coreConsultant.



Table 4-71 Link Capabilities Register (Continued)

Bits	Default	Attr	Description
17:15	`DEFAULT_L1_EXIT_LATENCY	RO	L1 Exit Latency The default value is the value you specify during hardware configuration, writable through the DBI. This parameter is not visible in coreConsultant.
17:15	`DEFAULT_COMM_L1_EXIT_LATENCY	RO	L1 Exit Latency; when common clock is used The default value is the value you specify during hardware configuration, writable through the DBI2. This parameter is not visible in coreConsultant.
18	`DEFAULT_CLK_PM_CAP	RO	Clock Power Management The default value is the value you specify during hardware configuration, writable through the DBI. This parameter is not visible in coreConsultant.
19	0x0	RO	Surprise Down Error Reporting Capable Not supported, hardwired to 0x0.
20	0x0	RO	Data Link Layer Active Reporting Capable
21	0x0	RO	Link Bandwidth Notification Capability
23:22	0x00	RsvdP	Reserved
31:24	0x0	HwInit	Port Number, writable through the DBI

4.1.7.3.7 Link Control Register

Offset: `CFG_PCIE_CAP + 0x10

Table 4-72 Link Control Register

Bits	Default	Attr	Description
1:0	0x0	RW	Active State Link PM Control
2	0x0	RsvdP	Reserved
3	0x0	RW	Read Completion Boundary (RCB)
4	0x0	RO	Link Disable Note: This bit is not applicable and is reserved for Endpoints.
5	0x0	RO	Retrain Link Note: This bit is not applicable and is reserved for Endpoints.
6	0x0	RW	Common Clock Configuration
7	0x0	RW	Extended Synch

Table 4-72 Link Control Register (Continued)

Bits	Default	Attr	Description
8	0x0	RW	Enable Clock Power Management Hardwired to 0 if Clock Power Management is disabled in the Link Capabilities register.
9	0x0	RO	Hardware Autonomous Width Disable Not supported, hardwired to 0.
10	0x0	RW	Link Bandwidth Management Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Bandwidth Management Status bit has been set. Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.
11	0x0	RW	Link Autonomous Bandwidth Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Autonomous Bandwidth Status bit has been set. Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.
15:12	0x00	Rsvd	Reserved

4.1.7.3.8 Link Status Register

Offset: `CFG_PCIE_CAP + 0x12

Table 4-73 Link Status Register

Bits	Default	Attr	Description
3:0	0x1	RO	Link Speed The negotiated Link speed.
9:4	0x1	RO	Negotiated Link Width Set automatically by hardware after Link initialization. Value is undefined when link is not up.
10	0x0	RO	Undefined for PCI Express 1.1 (Was Training Error for PCI Express 1.0a)
11	0	RO	Link Training Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.

Table 4-73 Link Status Register (Continued)

Bits	Default	Attr	Description
12	`SLOT_CLK_CONFIG	HwInit	Slot Clock Configuration Indicates that the component uses the same physical reference clock that the platform provides on the connector. The default value is the value you select during hardware configuration, writable through the DBI.
13	0x0	RO	Data Link Layer Active Note: This bit is not applicable and is reserved for Upstream Ports.
14	0x0	RO	Link Bandwidth Management Status Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.
15	0x0	RO	Link Autonomous Bandwidth Status Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.

4.1.7.3.9 Device Capabilities 2 Register

Offset: `CFG_PCIE_CAP + 0x24

Table 4-74 Slot Status Register

Bits	Default	Attr	Description
3:0	0x0	HWInit	Completion Timeout Ranges Supported This field is applicable only to Root Ports, Endpoints that issue Requests on their own behalf, and PCI Express to PCI/PCI-X Bridges that take ownership of Requests issued on PCI Express. If `CX_CPL_TO_RANGES_ENABLE is defined, then the default value is 0xf (A, B, C and D ranges supported). If `CX_CPL_TO_RANGES_ENABLE is not defined, the default value is 0x0.
4	0x1	RO	Completion Timeout Disable Supported

4.1.7.3.10 Device Control 2 Register

Offset: `CFG_PCIE_CAP + 0x28

Table 4-75 Slot Status Register

Bits	Default	Attr	Description
3:0	0x0	RW	<p>Completion Timeout Value</p> <p>If `CX_CPL_TO_RANGES_ENABLE is defined, the following encodings apply:</p> <ul style="list-style-type: none"> • 0000b Default range: 50 µs to 50 ms • 0001b 50 µs to 100 µs • 0010b 1 ms to 10 ms • 0101b 16 ms to 55 ms • 0110b 65 ms to 210 ms • 1001b 260 ms to 900 ms • 1010b 1 s to 3.5 s • 1101b 4 s to 13 s • 1110b 17 s to 64 s <p>Values not defined above are reserved.</p> <p>If the default range is chosen, the core will have a timeout in the range of 16ms to 55ms.</p> <p>If `CX_CPL_TO_RANGES_ENABLE is not defined, the core will have a timeout in the range of 16ms to 55ms.</p>
4	0x0	RW	Completion Timeout Disable

4.1.7.3.11 Link Control 2 Register

Offset: `CFG_PCIE_CAP + 30

Table 4-76 Link Control 2 Register

Bits	Default	Attr	Description
3:0	*	RW	<p>Target Link Speed</p> <p>For Downstream ports, this field sets an upper limit on link operational speed by restricting the values advertised by the upstream component in its training sequences:</p> <ul style="list-style-type: none"> • 0001: 2.5Gb/s Target Link Speed • 0010: 5Gb/s Target Link Speed <p>All other encodings are reserved.</p> <p>If a value is written to this field that does not correspond to a speed included in the Supported Link Speeds field, the result is undefined.</p> <p>*The default value of this field is the highest link speed supported by the component (as reported in the Supported Link Speeds field of the Link Capabilities Register) unless the corresponding platform / form factor requires a different default value.</p> <p>For both Upstream and Downstream ports, this field is used to set the target compliance mode speed when software is using the Enter Compliance bit to force a link into compliance mode.</p>
4	0x0	RWS	<p>Enter Compliance</p> <p>Software is permitted to force a link to enter Compliance mode at the speed indicated in the Target Link Speed field by setting this bit to 1b in both components on a link and then initiating a hot reset on the link.</p> <p>The default value of this field following Fundamental Reset is 0b.</p>
5	0x0	RO	<p>Hardware Autonomous Speed Disable</p> <p>When cfg_hw_auto_sp_dis signal is asserted, the application must disable hardware from changing the Link speed for device-specific reasons other than attempting to correct unreliable Link operation by reducing Link speed. Initial transition to the highest supported common link speed is not blocked by this signal.</p>

Table 4-76 Link Control 2 Register (Continued)

Bits	Default	Attr	Description
6	`SEL_DE_EMPHASIS	HwInit	Selectable De-emphasis When the Link is operating at 5.0 GT/s speed, selects the level of de-emphasis: <ul style="list-style-type: none">• 1: -3.5 dB• 0: -6 dB When the Link is operating at 2.5 GT/s speed, the setting of this bit has no effect. Components that support only the 2.5 GT/s speed are permitted to hardwire this bit to 0b. Not applicable for an upstream Port or Endpoint device. Hardwired to 0.
9:7	0x000	RW	Transmit Margin This field controls the value of the non-de-emphasized voltage level at the Transmitter pins: <ul style="list-style-type: none">• 000: 800-1200 mV for full swing 400-600 mV for half-swing• 001-010: values must be monotonic with a non-zero slope• 011: 200-400 mV for full-swing and 100-200 mV for halfswing• 100-111: reserved This field is reset to 000b on entry to the LTSSM Polling. Compliance substate. Components that support only the 2.5 GT/s speed are permitted to hard-wire this bit to 0b. When operating in 5.0 GT/s mode with full swing, the de-emphasis ratio must be maintained within +/- 1 dB from the specification-defined operational value (either -3.5 or -6 dB).
10	0x0	RW	Enter Modified Compliance When this bit is set to 1b, the device transmits a modified compliance pattern if the LTSSM enters Polling. Compliance state.
11	0x0	RWS	Compliance SOS When set to 1b, the LTSSM is required to send SKP Ordered Sets periodically in between the (modified) compliance patterns. Note: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect.

Table 4-76 Link Control 2 Register (Continued)

Bits	Default	Attr	Description
12	0x0	RWS	<p>Compliance De-emphasis</p> <p>This bit sets the de-emphasis level in Polling.Compliance state if the entry occurred due to the Tx Compliance Receive bit being 1b. Encodings:</p> <ul style="list-style-type: none"> • 1b: -3.5 dB • 0b: -6 dB <p>Note: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect.</p>
15:13	0x000	Rsvd	Reserved

4.1.7.3.12 Link Status 2 Register

Offset: `CFG_PCIE_CAP + 32

Table 4-77 Link Control 2 Register

Bits	Default	Attr	Description
0	0x0	RO	<p>Current De-emphasis Level</p> <p>When the Link is operating at 5 GT/s speed, this bit reflects the level of de-emphasis. Encodings:</p> <ul style="list-style-type: none"> • 1b: -3.5 dB • 0b: -6 dB <p>Note: The value in this bit is undefined when the Link is operating at 2.5 GT/s speed and permitted to hardwire this bit to 0b</p>

4.1.7.4 PF MSI-X Capability Register Details

The MSI-X Capability structure is an optional capability that can be supported either instead of or in addition to the MSI Capability structure. Even though both structures can exist in the same design, only one can be enabled at a time by software.

MSI-X increases the flexibility of MSI by allowing multiple vectors with different destination addresses. This overcomes the 32-vector limit of MSI and provides additional flexibility.

The MSI-X Capability structure points to an MSI-X Table and Pending Bit Array (PBA) structure that resides in memory space under a particular BAR (user configurable).

The MSI-X Capability structure is implemented inside the CDM as part of the capabilities linked list. The application designer is responsible for configuring the core with values for Table and PBA Offset and BIR numbers.

[Table 4-78](#) shows the PF MSI-X Standard Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-78 PF MSI-X Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`CFG_MSIX_CAP	MSI-X Control Register		Next Capability Pointer (`MSIX_NEXT_PTR)	Capability ID (0x11)	
+0x4	Table Offset (31:3) and BIR (2:0)				
+0x8	PBA Offset (31:3) and BIR (2:0)				

4.1.7.4.1 MSI-X Capability ID

Offset: `CFG_MSIX_CAP + 0x00

Table 4-79 MSI-X Capability ID Register

Bits	Default	Attr	Description
7:0	0x11	RO	MSI-X Capability ID

4.1.7.4.2 MSI-X Next Item Pointer

Offset: `CFG_MSIX_CAP + 0x01

Table 4-80 MSI-X Next Item Pointer

Bits	Default	Attr	Description
7:0	`MSIX_NEXT_PTR	RO	Next Capability Pointer Points to the VPD capability by default, writable through the DBI.

4.1.7.4.3 MSI-X Control Register

Offset: `CFG_MSIX_CAP + 0x02

Table 4-81 MSI-X Control Register

Bits	Default	Attr	Description
10:0	`MSIX_TABLE_SIZE_N	RO	MSI-X Table Size Encoded as (Table Size - 1). For example, a value of 0x003 indicates the MSI-X Table Size is 4.
13:11	000b	RsvdZ	Reserved
14	0	RW	Function Mask <ul style="list-style-type: none"> • 1: All vectors associated with the function are masked, regardless of their respective per-vector Mask bits. • 0: Each vector's Mask bit determines whether the vector is masked or not.
15	0	RW	MSI-X Enable If MSI-X is enabled, MSI and INTx must be disabled.

4.1.7.4.4 MSI-X Table Offset and BIR Register

Offset: `CFG_MSIX_CAP + 0x04

Table 4-82 MSI-X Table Offset and BIR Register

Bits	Default	Attr	Description
2:0	`MSIX_TABLE_BIR_N	RO	Table BAR Indicator Register (BIR) Indicates which BAR is used to map the MSI-X Table into memory space: <ul style="list-style-type: none"> • 000: BAR0 • 001: BAR1 • 010: BAR2 • 011: BAR3 • 100: BAR4 • 101: BAR5 • 110: Reserved • 111: Reserved
31:3	`MSIX_TABLE_OFFSET_N	RO	Table Offset Base address of the MSI-X Table, as an offset from the base address of the BAR indicated by the Table BIR bits.

4.1.7.4.5 MSI-X PBA Offset and BIR Register

Offset: `CFG_MSIX_CAP + 0x08

Table 4-83 MSI-X PBA Offset and BIR Register

Bits	Default	Attr	Description
2:0	`MSIX_PBA_BIR_N	RO	<p>Pending Bit Array (PBA) BIR Indicates which BAR is used to map the MSI-X PBA into memory space:</p> <ul style="list-style-type: none"> • 000: BAR0 • 001: BAR1 • 010: BAR2 • 011: BAR3 • 100: BAR4 • 101: BAR5 • 110: Reserved • 111: Reserved
31:3	`MSIX_PBA_OFFSET_N	RO	<p>PBA Offset Base address of the MSI-X PBA, as an offset from the base address of the BAR indicated by the PBA BIR bits.</p>

4.1.7.5 PF VPD Capability Register Details

The VPD interface follows the *PCI Base Specification*. The cfg_vpd_int signal pulses when the host has written to this register.

Table 4-84 shows the PF MSI-X Standard Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-84 PF VPD Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`CFG_VPD_CAP	VPD Capability ID				
+00h	VPD Control and Capabilities register				
+04h	VPD Data register				

4.1.7.5.1 VPD Control and Capabilities Register

Offset: `CFG_VPD_CAP + 00h

Table 4-85 VPD Control and Capabilities Register

Bit(s)	Type	Reset	Description
31	RW	0	VPD Flag
30:16	RW	0	VPD Address
15:8	HwInit	0	Next Capability Pointer Points to end of capabilities by default, writable through the DBI.
7:0	HwInit	03h	VPD Capability ID

4.1.7.5.2 VPD Data Register

Offset: `CFG_VPD_CAP + 04h

VPD data is updated by way of a host-application handshake, as described in “[Vital Product Data \(VPD\) Support \(optional\)](#)” on page [193](#).

Table 4-86 VPD Control and Capabilities Register

Bit(s)	Type	Reset	Description
31:0	RW	0	VPD Data

4.1.8 PF PCI Express Extended Capability Register Details

The core implements the following PCI Express Extended Capabilities registers:

- ❖ Advanced Error Reporting (AER) Capability register set
- ❖ Virtual Channel (VC) Capability register set – only exists in the first function of a multi-function device
- ❖ Device Serial Number Capability register set – only exists in the first function of a multi-function device
- ❖ Power Budgeting Capability register set
- ❖ ARI Extended Capability register set
- ❖ SR-IOV Extended Capability register set

The following sections describe the individual registers in each group. For register maps, see “[PF Register Maps](#)” on page [320](#).

4.1.8.1 PF Advanced Error Reporting Capability Registers

[Table 4-87](#) shows the PF Advanced Error Reporting (AER) Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-87 PF Advanced Error Reporting (AER) Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`AER_PTR	PCI Express Extended Capability Header				
+0x4	Uncorrectable Error Status Register				
+0x8	Uncorrectable Error Mask Register				
+0xC	Uncorrectable Error Severity Register				
+0x10	Correctable Error Status Register				
+0x14	Correctable Error Mask Register				
+0x18	Advanced Error Capabilities and Control Register				
+0x1C through +0x28	Header Log Registers				

4.1.8.1.1 PF PCI Express Extended Capability Header

Address: 0x100

Table 4-88 PCI Express Extended Capability Header

Bits	Default	Attr	Description
15:0	0x1	RO	PCI Express Extended Capability ID Default value is 0x1 for Advanced Error Reporting.
19:16	0x1	RO	Capability Version
31:20	0x140	RO	Next Capability Offset Points to the Virtual Channel Capability structure by default.

4.1.8.1.2 Uncorrectable Error Status Register

Offset: 0x04

Table 4-89 Uncorrectable Error Status Register

Bits	Default	Attr	Description
0	0x0	Undefined	Undefined for PCI Express 1.1 (Was Training Error Status for PCI Express 1.0a)
3:1	0x0	RsvdZ	Reserved

Table 4-89 Uncorrectable Error Status Register (Continued)

Bits	Default	Attr	Description
4	0x0	RW1CS	Data Link Protocol Error Status
5	0x0	RO	Surprise Down Error Status (not supported)
11:6	0x00	RsvdZ	Reserved
12	0x0	RW1CS	Poisoned TLP Status
13	0x0	RW1CS	Flow Control Protocol Error Status
14	0x0	RW1CS	Completion Timeout Status
15	0x0	RW1CS	Completer Abort Status
16	0x0	RW1CS	Unexpected Completion Status
17	0x0	RW1CS	Receiver Overflow Status
18	0x0	RW1CS	Malformed TLP Status
19	0x0	RW1CS	ECRC Error Status
20	0x0	RW1CS	Unsupported Request Error Status
31:21	0x000	RsvdZ	Reserved

4.1.8.1.3 Uncorrectable Error Mask Register

Offset: 0x08

Table 4-90 Uncorrectable Error Mask Register

Bits	Default	Attr	Description
0	0	Undefined	Undefined for PCI Express 1.1 (Was Training Error Mask for PCI Express 1.0a)
3:1	0x0	RsvdP	Reserved
4	0	RWS	Data Link Protocol Error Mask
5	0x0	RO	Surprise Down Error Mask (not supported)
11:6	0x00	RsvdP	Reserved
12	0	RWS	Poisoned TLP Mask
13	0	RWS	Flow Control Protocol Error Mask
14	0	RWS	Completion Timeout Mask
15	0	RWS	Completer Abort Mask
16	0	RWS	Unexpected Completion Mask

Table 4-90 Uncorrectable Error Mask Register (Continued)

Bits	Default	Attr	Description
17	0	RWS	Receiver Overflow Mask
18	0	RWS	Malformed TLP Mask
19	0	RWS	ECRC Error Mask
20	0	RWS	Unsupported Request Error Mask
31:21	0x000	RsvdP	Reserved

4.1.8.1.4 Uncorrectable Error Severity Register

Offset: 0x0C

Table 4-91 Uncorrectable Error Severity Register

Bits	Default	Attr	Description
0	0x1	Undefined	Undefined for PCI Express 1.1 (Was Training Error Severity for PCI Express 1.0a)
3:1	0x0	RsvdP	Reserved
4	0x1	RWS	Data Link Protocol Error Severity
5	0x1	RO	Surprise Down Error Severity (not supported)
11:6	0x00	RsvdP	Reserved
12	0x0	RWS	Poisoned TLP Severity
13	0x1	RWS	Flow Control Protocol Error Severity
14	0x0	RWS	Completion Timeout Severity
15	0x0	RWS	Completer Abort Severity
16	0x0	RWS	Unexpected Completion Severity
17	0x1	RWS	Receiver Overflow Severity
18	0x1	RWS	Malformed TLP Severity
19	0x0	RWS	ECRC Error Severity
20	0x0	RWS	Unsupported Request Error Severity
31:21	0x000	RsvdP	Reserved

4.1.8.1.5 Correctable Error Status Register

Offset: 0x10

Table 4-92 Correctable Error Status Register

Bits	Default	Attr	Description
0	0x0	RW1CS	Receiver Error Status
5:1	0x00	RsvdZ	Reserved
6	0x0	RW1CS	Bad TLP Status
7	0x0	RW1CS	Bad DLLP Status
8	0x0	RW1CS	REPLAY_NUM Rollover Status
11:9	0x00	RsvdZ	Reserved
12	0x0	RW1CS	Reply Timer Timeout Status
13	0x0	RW1CS	Advisory Non-Fatal Error Status
31:14	0x00000	RsvdZ	Reserved

4.1.8.1.6 Correctable Error Mask Register

Offset: 0x14

Table 4-93 Correctable Error Mask Register

Bits	Default	Attr	Description
0	0	RWS	Receiver Error Mask
5:1	0x00	RsvdP	Reserved
6	0	RWS	Bad TLP Mask
7	0	RWS	Bad DLLP Mask
8	0	RWS	REPLAY_NUM Rollover Mask
11:9	0x0	RsvdP	Reserved
12	0	RWS	Reply Timer Timeout Mask
13	0x1	RWS	Advisory Non-Fatal Error Mask
31:14	0x00000	RsvdP	Reserved

4.1.8.1.7 Advanced Capabilities and Control Register

Offset: 0x18

**Table 4-94 Advanced Capabilities and Control Register**

Bits	Default	Attr	Description
4:0	0x00	ROS	First Error Pointer
5	`DEFAULT_ECRC_GEN_CAP_N	RO	ECRC Generation Capability
6	0	RWS	ECRC Generation Enable
7	`DEFAULT_ECRC_CHK_CAP_N	RO	ECRC Check Capable
8	0	RWS	ECRC Check Enable
31:9	0x0000000	RsvdP	Reserved

4.1.8.1.8 Header Log Registers

Offset: 0x1C

The Header Log registers collect the header for the TLP corresponding to a detected error. See the *PCI Express 2.0 specification* for details. Each of the Header Log registers is type ROS; the default reset value of each Header Log register is 0x00000000.

Table 4-95 Header Log Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Header Log Register (first DWORD)			
0x20	Header Log Register (second DWORD)			
0x24	Header Log Register (third DWORD)			
0x28	Header Log Register (fourth DWORD)			

4.1.8.2 PF Virtual Channel Capability Registers

Table 4-96 shows the PF Virtual Channel Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-96 PF Virtual Channel Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VC_PTR	PCI Express Extended Capability Header				394
+0x4	Port VC Capability Register 1				394
+0x8	Port Capability Register 2				395
+0xC	Port VC Status Register		Port VC Control Register		395
+0x10	VC Resource Capability Register (0)				396
+0x14	VC Resource Control Register (0)				396
+0x18	VC Resource Status Register (0)		RsvdP		397
0x10+(N*0x0C)	VC Resource Capability Register (N) ¹				396
0x14+(N*0x0C)	VC Resource Control Register (N)				396
0x18+(N*0x0C)	VC Resource Status Register (N)		RsvdP		397

1. There is one VC Resource Capability/Control/Status Register N set for each configured VC (in addition to VC0).

4.1.8.2.1 VC Extended Capability Header

Offset: 0x140

Table 4-97 VC Extended Capability Header

Bits	Default	Attr	Description
15:0	0x2	RO	PCI Express Extended Capability The default value is 0x2 for VC Capability.
19:16	0x1	RO	Capability Version
31:20	0x000	RO	Next Capability Offset Points to the Device Serial Number Capability structure by default, if enabled. Default value should be `VC_NEXT_PTR.

4.1.8.2.2 Port VC Capability Register 1

Offset: 0x140 + 0x4

Table 4-98 Port VC Capability Register 1

Bits	Default	Attr	Description
2:0	`DEFAULT_EXT_VC_CNT	RO	Extended VC Count The default value is the one less than the number of VCs that you specify during hardware configuration (`CX_NVC - 1).
3	0x0	RsvdP	Reserved
6:4	`DEFAULT_LOW_PRI_EXT_VC_CNT	RO	Low Priority Extended VC Count, writable through the DBI
7	0x0	RsvdP	Reserved
9:8	0x0	RO	Reference Clock Not applicable for Endpoint, hardwired to 0.
11:10	0x0	RO	Port Arbitration Table Entry Size Not applicable for Endpoint, hardwired to 0.
31:12	0x0	RsvdP	Reserved

4.1.8.2.3 Port VC Capability Register 2**Offset:** 0x140 + 0x8**Table 4-99 Port VC Capability Register 2**

Bits	Default	Attr	Description
7:0	`DEFAULT_VC_ARB_32	RO	VC Arbitration Capability Indicates which VC arbitration mode(s) the device supports, writable through the DBI: <ul style="list-style-type: none">• Bit 0: Device supports hardware fixed arbitration scheme. For the core, the scheme is 16-phase weighted round robin (WRR).• Bit 1: Device supports 32-phase WRR.• Bit 2: Device supports 64-phase WRR.• Bit 3: Device supports 128-phase WRR.• Bits 4–7: Reserved
23:8		RsvdP	Reserved
31:24	0x00	RO	VC Arbitration Table Offset (not supported) The default value is 0x00 (no arbitration table present).

4.1.8.2.4 Port VC Control Register**Offset:** 0x140 + 0xC**Bytes:** 0–1

Table 4-100 Port VC Control Register

Bits	Default	Attr	Description
0	0x0	RW	Load VC Arbitration Table
3:1	0x0	RW	VC Arbitration Select
15:4	0x0	RsvdP	Reserved

4.1.8.2.5 Port VC Status Register**Offset:** 0x140 + 0xC**Bytes:** 2–3**Table 4-101 Port VC Status Register**

Bits	Default	Attr	Description
0	0x0	RO	Arbitration Table Status
15:1	0x0	RsvdZ	Reserved

4.1.8.2.6 VC Resource Capability Register (0)**Offset:** 0x140 + 0x10**Table 4-102 VC Resource Capability Register (0)**

Bits	Default	Attr	Description
7:0	0x00	RO	Port Arbitration Capability
13:8	0x00	RsvdP	Reserved
14	0x0	RO	Undefined for PCI Express 1.1 (Was Advanced Packet Switching for PCI Express 1.0a)
15	0x0	HwInit	Reject Snoop Transactions Not valid for Endpoints, must be 0x0.
22:16	0x0	HwInit	Maximum Time Slots Not valid for Endpoints.
23	0x0	RsvdP	Reserved
31:24	0x00	RO	Port Arbitration Table Offset Hardwired to 0x00, not applicable for Endpoints or devices that do not implement a port arbitration table.

4.1.8.2.7 VC Resource Control Register (0)**Offset:** 0x140 + 0x14

Table 4-103 VC Resource Control Register (0)

Bits	Default	Attr	Description
7:0	0xFF	RW	TC/VC Map Bit 0 is hardwired to 1; bits 7:1 are RW.
15:8		RsvdP	Reserved
16	0x0	RW	Load Port Arbitration Table Not applicable for Endpoint.
19:17	0x0	RW	Port Arbitration Select Hardwired to 0 for Endpoint.
23:20	0x0	RsvdP	Reserved
26:24	0x0	RO	VC ID Hardwired to 0 for VC0.
30:27		RsvdP	Reserved
31	0x1	RO	VC Enable Hardwired to 1 for the first VC.

4.1.8.2.8 VC Resource Status Register (0)

Offset: 0x140 + 0x18

Table 4-104 VC Resource Status Register (0)

Bits	Default	Attr	Description
15:0	0x0	RsvdP	Reserved
16	0x0	RO	Port Arbitration Table Status
17	0x1	RO	VC Negotiation Pending
31:18	0x00	RsvdZ	Reserved

4.1.8.2.9 VC Resource Capability Register (N)

Offset: 0x140 + 0x10+(N*0x0C)

There is a VC Resource Capability Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-105 VC Resource Capability Register (N)

Bits	Default	Attr	Description
7:0	0x01	RO	Port Arbitration Capability The default value is 0x01 for hardwired, fixed arbitration scheme.
13:8	0x00	RsvdP	Reserved

Table 4-105 VC Resource Capability Register (N) (Continued)

Bits	Default	Attr	Description
14	0x0	RO	Undefined for PCI Express 1.1 (Was Advanced Packet Switching for PCI Express 1.0a)
15	0x0	HwInit	Reject Snoop Transactions
22:16	0x0	HwInit	Maximum Time Slots
23		RsvdP	Reserved
31:24	0x00	RO	Port Arbitration Table Offset Hardwired to 0x00, not applicable for Endpoints or devices that do not implement a port arbitration table.

4.1.8.2.10 VC Resource Control Register (N)**Offset:** 0x140 + 0x14+(N*0x0C)

There is a VC Resource Control Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-106 VC Resource Control Register (N)

Bits	Default	Attr	Description
7:0	0x00	RW	TC/VC Map
15:8		RsvdP	Reserved
16	0x0	RW	Load Port Arbitration Table
19:17	0x0	RW	Port Arbitration Select Hardwired to 0 for Endpoint.
23:20	0x0	RsvdP	Reserved
26:24	0x0	RW	VC ID
30:27		RsvdP	Reserved
31	0x0	RW	VC Enable

4.1.8.2.11 VC Resource Status Register (N)**Offset:** 0x140 + 0x18+(N*0x0C)

There is a VC Resource Status Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-107 VC Resource Status Register (N)

Bits	Default	Attr	Description
15:0	0x0	RsvdP	Reserved
16	0x0	RO	Arbitration Table Status

**Table 4-107 VC Resource Status Register (N) (Continued)**

Bits	Default	Attr	Description
17	0x0	RO	VC Negotiation Pending
31:18	0x00	RsvdZ	Reserved



4.1.8.3 PF Device Serial Number Capability Register

[Table 4-108](#) shows the PF Device Serial Number Capability Register Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-108 PF Device Serial Number Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	
`SN_PTR ¹	Device Serial Number Extended Capability Header				
+0x4	Serial Number Register (1st DWORD)				
+0x8	Serial Number Register (2nd DWORD)				

1. Depends on the number of VCs

4.1.8.3.1 Device Serial Number Extended Capability Header

Offset: `SN_PTR (Depends on the number of VCs)

Table 4-109 Device Serial Number Extended Capability Header

Bit(s)	Default	Attr	Description
15:0	0x0003	HwInit	Extended Capability ID
19:16	0x1	HwInit	Capability Version
31:20	0x0	HwInit	Points to end of capabilities by default, writable through the DBI

4.1.8.3.2 Serial Number Register

Offset: `SN_PTR + 0x4 / +0x8

Table 4-110 Serial Number Registers

Offset	Default	Attr	Description
0x4	Parameter	RO	Serial Number register (1st DWORD), writable through the DBI
0x8	Parameter	RO	Serial Number register (2nd DWORD) writable through the DBI

4.1.8.4 PF Power Budgeting Capability

The PCI Power Budgeting Capability allows the system to properly allocate power to devices that are added to the system at runtime. Through this capability, a device can report the power it consumes on a variety of power rails, in a variety of device power management states, and in a variety of operating conditions. The system uses this information to ensure that the system is capable of providing the proper power and cooling levels to the device. Failure to properly indicate device power consumption may risk device or system failure.

Implementation of the Power Budgeting Capability is optional for PCIe devices that are implemented either in a form factor that does not require Hot-Plug support, or that are integrated on the system board. PCIe form factor specifications may require support for power budgeting.

[Table 4-111](#) shows the PF Power Budgeting Capability Register Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-111 PF Power Budgeting Capability Register

Table 4-112

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`PB_PTR	Power Budgeting Extended Capability Header				401
	Data Select Register				402
	Data Register				402
	Power Budget Capability Register				403

4.1.8.4.1 Power Budgeting Extended Capability Header

Offset: `PB_PTR

Refer to section 7.15 of the *PCI Express 2.0 specification* for a description of the Power Budgeting Capability.

Table 4-113 Power Budgeting Extended Capability Header

Bits	Default	Attr	Description
15:0	0x0004	RO	Extended Capability ID This field is a PCI-SIG defined ID number that indicates the nature and format of the extended capability. Extended Capability ID for Power Budgeting capability is 0x0004.
19:16	0x1	RO	Capability Version This field is a PCI-SIG defined version number that indicates the version of the capability structure present. Must be 0x1 as per the <i>PCI Express 2.0 specification</i> .
31:20	`PB_NEXT_PTR	RO	Next Capability Offset Points to the next capability structure. Default value should be `PB_NEXT_PTR

4.1.8.4.2 Data Select Register

Offset: `PB_PTR + 0x4

This read/write register indexes the Power Budgeting data reported through the Data register and selects the DWORD of Power Budgeting Data that should appear in the Data register. Index values for this register start at 0 to select the first DWORD of Power budgeting Data; subsequent DWORDs of power budgeting data are selected by increasing index values.

4.1.8.4.3 Data Register

Offset: `PB_PTR + 0x8

Table 4-114 Data Select Register

Bits	Default	Attr	Description
7:0	0x0	RO	<p>Base Power</p> <p>Specifies (in watts) the base power value in the given operating condition. This value must be multiplied by the data scale to produce the actual power consumption value.</p>
9:8	0x0	RO	<p>Data Scale</p> <p>Specifies the scale to apply to the Base Power value. The device power consumption is determined by multiplying the contents of the Base Power register field with the value corresponding to the encoding returned by this field:</p> <ul style="list-style-type: none"> 00: 1.0x 01: 0.1x 10: 0.01x 11: 0.001x
12:10	0x0	RO	<p>PM Sub State</p> <p>Specifies the power management sub state of the operating condition being described:</p> <ul style="list-style-type: none"> 000: Default sub state 001 - 111: Device-specific sub state
14:13	0x0	RO	<p>PM State</p> <p>Specifies the power management state of the operating condition being described:</p> <ul style="list-style-type: none"> 00: D0 01: D1 10: D2 11: D3 <p>A device returns 11b in this field and Aux or PME Aux in the Type register to specify the D3-Cold PM state. An encoding of 11b along with any other Type register value specifies the D3-Hot state.</p>

Table 4-114 Data Select Register (Continued)

Bits	Default	Attr	Description
17:15	0x0	RO	Type Specifies the power rail of the operating condition being described: 000: Power (12V) 001: Power (3.3V) 010: Power (1.8V) 111: Thermal All other encodings are reserved.
20:18	0x0	RO	Power Rail Specifies the type of the operating condition being described: 000: PME Aux 001: Auxiliary 010: Idle 011: Sustained 111: Maximum All other encodings are reserved.
31:21	0x0	RsvdP	Reserved

4.1.8.4.4 Power Budget Capability Register

Offset: `PB_PTR + 0xC

This register indicates the power budgeting capability of a device.

Table 4-115 Power Budget Capability Register

Bits	Default	Attr	Description
0	DEFAULT_ PWR_BUDGET_ SYS_ALLOC	HwInit	System Allocated This bit when set, indicates that the power budget for the device is included within the system power budget. Reported power budgeting data for this device should be ignored by software for power budgeting decisions if this bit is set.
7:1	0x0	RsvdP	Reserved

4.1.8.5 PF ARI Capability Registers

[Table 4-116](#) shows the PF Alternate Routing-ID (ARI) Capability Register Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-116 PF Alternate Routing-ID (ARI) Capability Register

Table 4-117

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`ARI_PTR	ARI Capability Header				404
+0x04	ARI Control Register		ARI Capability Register		404

4.1.8.5.1 ARI Capability Header

Offset: Offset : 0x0

Table 4-118 ARI Capability Header

Bits	Default	Attr	Description
15:0	0x000E	RO	PCI Express Extended Capability ID
19:16	0x1	RO	Capability Version
31:20	`ARI_NEXT_PTR	RO	Next Capability Offset

4.1.8.5.2 ARI Capability Register

Offset: Offset : 0x4

Table 4-119 ARI Capability Register

Bits	Default	Attr	Description
0	0	RO	MFVC Function Groups Capability (M)
1	0	RO	ACS Function Groups Capability (A)
7:2		RsvdP	Reserved
15:8	Eqn1	RO	Next Function Number
Eqn1 = {FUNC_NUM+1} when (FUNC_NUM+1 /= `CX_NFUNC) else 0			

4.1.8.5.3 ARI Control Register

Offset: Offset : 0x6

Table 4-120 ARI Control Register

Bits	Default	Attr	Description
0	0	RW	MFVC Function Groups Enable (M)
1	0	RW	ACS Function Groups Enable (A)

**Table 4-120 ARI Control Register (Continued)**

Bits	Default	Attr	Description
3:2		RsvdP	Reserved
6:4	0x0	RW	Function Group
15:7		RsvdP	Reserved



4.1.8.6 PF SR-IOV Capability Registers

[Table 4-121](#) shows the PF Single Root I/O Virtualization (SR-IOV) Capability Register Capability Structure. Refer to the indicated pages for detailed register descriptions.

Table 4-121 PF Single Root I/O Virtualization (SR-IOV) Capability Register

Table 4-122

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`SRIOV_PTR	SRIOV Capability Header				406
+0x04	SRIOV Capabilities				407
+0x08	SRIOV Status		SRIOV Control		407
+0x0C	TotalVFs		InitialVFs		407
+0x10	RsvdP	Function Dependency Link	NumVFs		
+0x14	VF Stride		First VF Offset		408
+0x18	VF Device ID		RsvdP		409
+0x1C	Supported Page Sizes				409
+0x20	System Page Size				409
+0x24	VF BAR0				409
+0x28	VF BAR1				409
+0x2C	VF BAR2				410
+0x30	VF BAR3				410
+0x34	VF BAR4				410
+0x38	VF BAR5				410
+0x3C	VF Migration State Array Offset				410

4.1.8.6.1 SR-IOV Capability Header

Offset: Offset : 0x0

Table 4-123 SR-IOV Capability Header

Bits	Default	Attr	Description
15:0	0x000E	RO	PCI Express Extended Capability ID
19:16	0x1	RO	Capability Version
31:20	`SRIOV_NEXT_PTR	RO	Next Capability Offset



4.1.8.6.2 SR-IOV Capability Register

Offset: 0x4

Table 4-124 SR-IOV Capability Register

Bits	Default	Attr	Description
0	0	RO	VF Migration Capable
20:1		RsvdP	Reserved
31:21	0x0	RO	VF Migration Interrupt Message Number

4.1.8.6.3 SR-IOV Control Register

Offset: Offset : 0x8

Table 4-125 SR-IOV Control Register

Bits	Default	Attr	Description
0	0	RW	VF Enable
1	0	RO	VF Migration Enable
2	0	RO	VF Migration Interrupt Enable
3	0	RW	MSE
4		RW/RO	ARI Capable Hierarchy
15:5		RsvdP	Reserved

4.1.8.6.4 SR-IOV Status Register

Offset: Offset : 0xA

Table 4-126 SR-IOV Status Register

Bits	Default	Attr	Description
0	0	RO	VF Migration Status
15:1		RsvdP	Reserved

4.1.8.6.5 InitialVFs Register

Offset: Offset : 0x0C

Table 4-127 InitialVFs Register

Bits	Default	Attr	Description
15:0	HwInit	RO	InitialVFs



4.1.8.6.6 TotalVFs Register

Offset: Offset : 0x0E

Table 4-128 TotalVFs Register

Bits	Default	Attr	Description
15:0	HwInit	RO	TotalVFs

4.1.8.6.7 NumVFs Register

Offset: Offset : 0x10

Table 4-129 NumVFs Register

Bits	Default	Attr	Description
15:0	undefined	RO	NumVFs, writable through the DBI

4.1.8.6.8 Function Dependency Link

Offset: Offset : 0x12

Table 4-130 Function Dependency Link

Bits	Default	Attr	Description
7:0	HwInit	RO	Function Dependency Link

4.1.8.6.9 First VF Offset

Offset: Offset : 0x14

Table 4-131 First VF Offset

Bits	Default	Attr	Description
15:0	HwInit	RO	First VF Offset, writable through the DBI

4.1.8.6.10 VF Stride

Offset: Offset : 0x16

Table 4-132 VF Stride

Bits	Default	Attr	Description
15:0	HwInit	RO	VF Stride, writable through the DBI

4.1.8.6.11 VF Device ID

Offset: Offset : 0x1A

Table 4-133 VF Device ID

Bits	Default	Attr	Description
15:0	HwInit	RO	VF Device ID, writable through the DBI

4.1.8.6.12 Supported Page Sizes

Offset: Offset : 0x1C

Table 4-134 Supported Page Sizes

Bits	Default	Attr	Description
31:0	HwInit	RO	Supported Page Sizes, writable through the DBI

4.1.8.6.13 System Page Size

Offset: Offset : 0x20

Table 4-135 System Page Size

Bits	Default	Attr	Description
310	0x00000001	RW	System Page Size

4.1.8.6.14 VF BAR0

Offset: Offset : 0x24

Table 4-136 VF BAR0

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR0

4.1.8.6.15 VF BAR1

Offset: Offset : 0x28

Table 4-137 VF BAR1

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR1

4.1.8.6.16 VF BAR2

Offset: Offset : 0x2C

Table 4-138 VF BAR2

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR2

4.1.8.6.17 VF BAR3

Offset: Offset : 0x30

Table 4-139 VF BAR3

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR3

4.1.8.6.18 VF BAR4

Offset: Offset : 0x34

Table 4-140 VF BAR4

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR4

4.1.8.6.19 VF BAR5

Offset: Offset : 0x38

Table 4-141 VF BAR5

Bits	Default	Attr	Description
31:0	HwInit	RW	VF BAR5

4.1.8.6.20 VF Migration State Array Offset

Offset: Offset : 0x3C

Table 4-142 VF BAR5

Bits	Default	Attr	Description
2:0	0x0	RO	VF Migration State BIR
31:3	0x0	RO	VF Migration State Offset

4.1.8.6.21 VF BAR 0 Mask Register

Offset: 0x24 (same as Base Address Register 0, but requires dbi_cs2 for write access)

Table 4-143 VF BAR 0 Mask Register

Bits	Default	Attr	Description
31:1	`VF_BAR0_MASK_N	Appl. write access only; not readable	<p>Indicates which BAR 0 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the BAR 0 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 0 is a 64-bit BAR, then the BAR 1 Mask register contains the upper bits of the BAR 0 Mask. The BAR 0 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `VF_BAR0_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `VF_BAR0_MASK_WRITABLE_N = 1, the BAR 0 Mask register is writable through the DBI. • If `VF_BAR0_MASK_WRITABLE_N = 0, the BAR 0 Mask register is not writable through the DBI.
0	`VF_BAR0_ENABLED_N	Same as bits [31:1]	<p>BAR 0 Enable</p> <ul style="list-style-type: none"> • 0: BAR 0 is disabled • 1: BAR 0 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.1.8.6.22 VF BAR 1 Mask Register

Offset: 0x28 (same as Base Address Register 1, but requires dbi_cs2 for write access)

Table 4-144 VF_BAR 1 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 0 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 1 Mask value, interpreted the same way as BAR 0 Mask. Default value is `VF_BAR1_MASK_N. • Bit 0: BAR 1 Enable (0 = BAR 1 is disabled; 1= BAR 1 is enabled). Default value is `VF_BAR1_ENABLED_N. • `VF_BAR1_MASK_WRITABLE_N controls application write access to the BAR 1 Mask register. <p>If BAR 0 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 0 Mask value. • `VF_BAR0_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 0 Mask register.

4.1.8.6.23 VF_BAR 2 Mask Register

Offset: 0x2C (same as Base Address Register 2, but requires dbi_cs2 for write access)

Table 4-145 VF_BAR 2 Mask Register

Bits	Default	Attr	Description
31:1	`VF_BAR2_MASK_N	Appl. access only	<p>Indicates which BAR 2 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the BAR 2 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 2 is a 64-bit BAR, then the BAR 3 Mask register contains the upper bits of the BAR 2 Mask. The BAR 2 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `VF_BAR2_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `VF_BAR2_MASK_WRITABLE_N = 1, the BAR 2 Mask register is writable through the DBI. • If `VF_BAR2_MASK_WRITABLE_N = 0, the BAR 2 Mask register is not writable through the DBI.

Table 4-145 VF_BAR2 Mask Register (Continued)

Bits	Default	Attr	Description
0	`VF_BAR2_ENABLED_N	Same as bits [31:1]	<p>BAR 2 Enable</p> <ul style="list-style-type: none"> • 0: BAR 2 is disabled • 1: BAR 2 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.1.8.6.24 VF_BAR3 Mask Register

Offset: 0x30 (same as Base Address Register 3, but requires dbi_cs2 for write access)

Table 4-146 VF_BAR3 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 2 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 3 Mask value, interpreted the same way as BAR 2 Mask. Default value is `VF_BAR3_MASK_N. • Bit 0: BAR 3 Enable (0 = BAR 3 is disabled; 1= BAR 3 is enabled). Default value is `VF_BAR3_ENABLED_N. • `VF_BAR3_MASK_WRITABLE_N controls application write access to the BAR 3 Mask register. <p>If BAR 2 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 2 Mask value. • `VF_BAR2_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 2 Mask register.

4.1.8.6.25 VF_BAR4 Mask Register

Offset: 0x34 (same as Base Address Register 4, but requires dbi_cs2 for write access)

Table 4-147 VF_BAR 4 Mask Register

Bits	Default	Attr	Description
31:1	`VF_BAR4_MASK_N	Appl. access only	<p>Indicates which BAR 4 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFF to the BAR 4 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 4 is a 64-bit BAR, then the BAR 5 Mask register contains the upper bits of the BAR 4 Mask.</p> <p>The BAR 4 Mask register is invisible to host software and not readable from the application. Application write access depends on the value of `VF_BAR4_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `VF_BAR4_MASK_WRITABLE_N = 1, the BAR 4 Mask register is writable through the DBI. • If `VF_BAR4_MASK_WRITABLE_N = 0, the BAR 4 Mask register is not writable through the DBI.
0	`VF_BAR4_ENABLED_N	Same as bits [31:1]	<p>BAR 4 Enable</p> <ul style="list-style-type: none"> • 0: BAR 4 is disabled • 1: BAR 4 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.1.8.6.26 VF_BAR 5 Mask Register

Offset: 0x38 (same as Base Address Register 5, but requires dbi_cs2 for write access)

Table 4-148 VF_BAR 5 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 4 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 5 Mask value, interpreted the same way as BAR 5 Mask. Default value is `VF_BAR5_MASK_N. • Bit 0: BAR 5 Enable (0 = BAR 5 is disabled; 1= BAR 5 is enabled). Default value is `VF_BAR5_ENABLED_N. • `VF_BAR5_MASK_WRITABLE_N controls application write access to the BAR 5 Mask register. <p>If BAR 4 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 4 Mask value. • `VF_BAR4_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 4 Mask register.

4.1.9 VF PCI-Compatible Configuration Header Register Details



Note In the tables below, a default value of “PF” means that the default value of the VF is the same as the default value of the PF that owns this VF.

4.1.9.1 VF PCI Configuration Space Header – Type 0

Table 4-149 shows the configuration field register definitions for a Virtual Function PCI Express Type 0 Configuration Space header. Most PCI-compatible register fields have the same software interpretation in PCI 3.0 and PCI Express. Refer to the indicated page numbers for detailed register descriptions.

Table 4-149 VF PCI Configuration Space Header – Type 0

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x00	Device ID		Vendor ID		416
0x04	Status Register		Command Register		416
0x08	Class Code			Revision ID	418
0x0C	BIST(0x00)	Header Type	Latency Timer	Cache Line Size	418
0x10	Base Address Register 0				419
0x14	Base Address Register 1				419
0x18	Base Address Register 2				419
0x1C	Base Address Register 3				419
0x20	Base Address Register 4				419
0x24	Base Address Register 5				419
0x28	CardBus CIS Pointer				419
0x2C	Subsystem ID		Subsystem Vendor ID		419
0x30	Expansion ROM Base Address				420
0x34	Reserved			CapPtr	420
0x38	Reserved				
0x3C	Max_Latency ¹	Min_Grant ¹	Interrupt Pin	Interrupt Line	420

1. The Max_Latency and Min_Grant registers do not apply to PCI Express and are read-only registers with values hardwired to 0x00.

4.1.9.2 VF Device ID and Vendor ID Register

Offset: 0x00

[Table 4-23](#) defines the Device ID and Vendor ID register bit assignments.

Table 4-150 VF Device ID and Vendor ID Register

Bits	Default	Attr	Description
15:0	0xFFFF	RO	Vendor ID
31:16	0xFFFF	RO	Device ID

4.1.9.3 VF Command Register

Offset: 0x04

Bytes: 0–1

[Table 4-24](#) defines the PCI-compatible Command register bit assignments.

Table 4-151 VF Command Register

Bits	Default	Attr	Description
0	0	RO	I/O Space Enable Not applicable for SR-IOV. Must be hardwired to 0.
1	0	RO	Memory Space Enable Not applicable for SR-IOV. Must be hardwired to 0.
2	0	RW	Bus Master Enable
3	0	RO	Special Cycle Enable Not applicable for PCI Express. Must be hardwired to 0.
4	0	RO	Memory Write and Invalidate Not applicable for PCI Express. Must be hardwired to 0.
5	0	RO	VGA Palette Snoop Not applicable for PCI Express. Must be hardwired to 0.
6	PF	RO	Parity Error Response
7	0	RO	IDSEL Stepping/Wait Cycle Control Not applicable for PCI Express. Must be hardwired to 0
8	PF	RO	SERR# Enable
9	0	RO	Fast Back-to-Back Enable Not applicable for PCI Express. Must be hardwired to 0.
10	0	RO	INTx Assertion Disable Not applicable for SR-IOV. Must be hardwired to 0.
15:11	0x0	RO	Reserved

4.1.9.4 VF Status Register

Offset: 0x04

Bytes: 2–3

Table 4-25 defines the PCI-compatible Status register bit assignments.

Table 4-152 VF Status Register

Bits	Default	Attr	Description
2:0	0x0	RO	Reserved
3	0	RO	INTx Status Not applicable for SR-IOV. Must be hardwired to 0.
4	1	RO	Capabilities List Indicates presence of an extended capability item. Hardwired to 1. Not applicable for SR-IOV. Must be hardwired to 0.
5	0	RO	66 MHz Capable Not applicable for PCI Express. Hardwired to 0. Not applicable for SR-IOV. Must be hardwired to 0.
6	0	RO	Reserved Not applicable for SR-IOV. Must be hardwired to 0.
7	0	RO	Fast Back-to-Back Capable Not applicable for PCI Express. Hardwired to 0. Not applicable for SR-IOV. Must be hardwired to 0.
8	0	RW1C	Master Data Parity Error
10:9	0x0	RO	DEVSEL Timing Not applicable for PCI Express. Hardwired to 0.
11	0	RW1C	Signaled Target Abort
12	0	RW1C	Received Target Abort
13	0	RW1C	Received Master Abort
14	0	RW1C	Signaled System Error
15	0	RW1C	Detected Parity Error

4.1.9.5 VF Revision ID Register

Offset: 0x08

Byte: 0

Table 4-153 VF Revision ID Register

Bits	Default	Attr	Description
7:0	PF	RO	Revision ID.

4.1.9.6 VF Class Code Register

Offset: 0x08

Bytes: 1–3

Table 4-154 VF Class Code Register

Bits	Default	Attr	Description
7:0	PF	RO	Programming Interface.
15:8	PF	RO	Subclass Code.
23:16	PF	RO	Base Class Code.

4.1.9.7 VF Cache Line Size Register

Offset: 0x0C

Byte: 0

Table 4-155 VF Cache Line Size Register

Bits	Default	Attr	Description
7:0	0	RO	Cache Line Size.
	0x00 (PFs)	RW (PFs)	

4.1.9.8 VF Master Latency Timer Register

Offset: 0x0C

Byte: 1

Table 4-156 VF Master Latency Timer Register

Bits	Default	Attr	Description
7:0	0x00	RO	Master Latency Timer Not applicable for PCI Express, hardwired to 0.

4.1.9.9 VF Header Type Register

Offset: 0x0C

Byte: 2

Table 4-157 VF Header Type Register

Bits	Default	Attr	Description
6:0	0x0	RO	Configuration Header Format Hardwired to 0 for type 0.
7	0 0x0 (PFs)	RO	Multi Function Device

4.1.9.10 VF BIST Register

Offset: 0x0C

Byte: 3

Table 4-158 VF BIST Register

Bits	Default	Attr	Description
7:0	0x00	RO	The BIST register functions are not supported by the core. All 8 bits of the BIST register are hardwired to 0.

4.1.9.11 VF Base Address Registers

For all details on the base address registers in the VF PCI-Compatible Configuration Header Register, refer to the *Single Root I/O Virtualization and Sharing Specification (SR-IOV)*.

A Read request in these base address registers returns all 0s.

4.1.9.12 VF CardBus CIS Pointer Register

Offset: 0x28

Table 4-159 VF CardBus CIS Pointer Register

Bits	Default	Attr	Description
31:0	0 0x0 (PFs)	RO	CardBus CIS Pointer

4.1.9.13 VF Subsystem ID and Subsystem Vendor ID Register

Offset: 0x2C

Table 4-160 VF Subsystem ID and Subsystem Vendor ID Register

Bits	Default	Attr	Description
15:0	PF	RO	Subsystem Vendor ID
31:16	PF	RO	Subsystem ID

4.1.9.14 VF Expansion ROM Base Address Register

Offset: 0x30

Table 4-161 VF Expansion ROM Base Address Register

Bits	Default	Attr	Description
31:11	0 0x00000 (PFs)	RO RW (PFs)	Expansion ROM Address
10:1	0x000	RO	Reserved
0	0 0x0 (PFs)	RO RW (PFs)	Expansion ROM Enable

4.1.9.15 VF Capability Pointer Register



The First Capability Pointer points to the PCI Express Capability structure. This applies for all VFs.

Offset: 0x34

Byte: 0

Table 4-162 VF Capability Pointer Register

Bits	Default	Attr	Description
7:0	`VF_CFG_NEXT_PTR	RO	First Capability Pointer. Points to PCI Express Capability structure.

4.1.9.16 VF Interrupt Line Register

Offset: 0x3C

Byte: 0

Table 4-163 VF Interrupt Line Register

Bits	Default	Attr	Description
7:0	0 0x0 (PFs)	RO RW (PFs)	Interrupt Line



4.1.9.17 VF Interrupt Pin Register

Offset: 0x3C

Byte: 1

Table 4-164 VF Interrupt Pin Register

Bits	Default	Attr	Description
7:0	0 0x0 (PFs)	RO RW (PFs)	Interrupt Pin



4.1.10 VF PCI Standard Capability Structures Register Details

4.1.10.1 VF PCI Express Capability Register Details

The core implements the VF PCI Express Capability Structure as defined in the *Single Root I/O Virtualization and Sharing Specification (SR-IOV)* and the *PCI Express Base 2.0 Specification* except for Root Port registers in the PCIe spec.



In the tables below, a default value of “PF” means that the default value of the VF is the same as the default value of the PF that owns this VF.

Table 4-165 shows the VF PCI Standard Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-165 VF PCI Express Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VF_CFG_PCIE_CAP	PCI Express Capabilities Register		Next Capability Pointer (`VF_PCIE_NEXT_PTR)	Capability ID (0x10)	422
+0x4	Device Capabilities				423
+0x8	Device Status		Device Control		424
+0xC	Link Capabilities				425
+0x10	Link Status		Link Control		426
+0x24	Device Capabilities 2				428
+0x28			Device Control 2		428
+0x30	Link Status 2		Link Control 2		429

4.1.10.1.1 VF PCI Express Capability List Register

Offset: `VF_CFG_PCIE_CAP + 0x00

Table 4-166 VF PCI Express Capability List Register

Bits	Default	Attr	Description
7:0	0x10	RO	PCI Express Capability ID
15:8	`VF_PCIE_NEXT_PTR	RO	Next Capability Pointer Points to the MSI-X capability by default, writable through the DBI.

4.1.10.1.2 VF PCI Express Capabilities Register

Offset: `VF_CFG_PCIE_CAP + 0x02

Table 4-167 VF PCI Express Capabilities Register

Bits	Default	Attr	Description
3:0	0x2	RO	PCI Express Capability Version
7:4	PF	RO	Device Port Type
8	PF	HwInit	Slot Implemented This bit is writable through the DBI. However, it must 0 for an Endpoint device. Therefore, the application must not write a 1 to this bit.
13:9	PF	RO	Interrupt Message Number Updated by hardware, writable through the DBI.
15:14	0x0	RO	RsvdP

4.1.10.1.3 VF Device Capabilities Register

Offset: `VF_CFG_PCIE_CAP + 0x04

Table 4-168 VF Device Capabilities Register

Bits	Default	Attr	Description
2:0	Same for all VF and PF.	RO	Max_Payload_Size Supported
4:3	0x0	RO	Phantom Function Supported
5	PF	RO	Extended Tag Field Supported
8:6	PF	RO	Endpoint L0s Acceptable Latency
11:9	PF	RO	Endpoint L1 Acceptable Latency
12	0x0	RO	Undefined for PCI Express 1.1 (Was Attention Button Present for PCI Express 1.0a)
13	0x0	RO	Undefined for PCI Express 1.1 (Was Attention Indicator Present for PCI Express 1.0a)
14	0x0	RO	Undefined for PCI Express 1.1 (Was Power Indicator Present for PCI Express 1.0a)
15	PF	RO	Role-Based Error Reporting Required to be set for device compliant to 1.1 spec and later.
17:16	0x0	RsvdP	Reserved

Table 4-168 VF Device Capabilities Register (Continued)

Bits	Default	Attr	Description
25:18	0x00	RO	Captured Slot Power Limit Value From Message from RC, upstream port only.
27:26	0x0	RO	Captured Slot Power Limit Scale From Message from RC, upstream port only.
28	0x1	RO	Function Level Reset Capability
31:29	0x0	RsvdP	Reserved

4.1.10.1.4 VF Device Control Register

Offset: `VF_CFG_PCIE_CAP + 0x08

Table 4-169 VF Device Control Register

Bits	Default	Attr	Description
0	PF	RW	Correctable Error Reporting Enable
1	PF	RW	Non-Fatal Error Reporting Enable
2	PF	RW	Fatal Error Reporting Enable
3	PF	RW	Unsupported Request Reporting Enable
4	PF	RW	Enable Relaxed Ordering
7:5	PF	RW	Max_Payload_Size
8	PF	RW	Extended Tag Field Enable
9	PF	RW	Phantom Function Enable
10	PF	RWS	AUX Power PM Enable
11	PF	RW	Enable No Snoop
14:12	PF	RW	Max_Read_Request_Size
15	0x0	RW	Initiate FLR

4.1.10.1.5 VF Device Status Register

Offset: `VF_CFG_PCIE_CAP + 0x0A

Table 4-170 VF Device Status Register

Bits	Default	Attr	Description
0	0	RO	Correctable Error Detected All correctable errors are non-function specific, so they get reported only in the PF.

Table 4-170 VF Device Status Register (Continued)

Bits	Default	Attr	Description
1	0	RO	Non-Fatal Error detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
2	0	RO	Fatal Error Detected All fatal errors are non-function specific, so they get reported only in the PF.
3	0	RO	Unsupported Request Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
4	0	RO	Aux Power Detected
5	0	RO	Transaction Pending Set to 1 when Non-Posted Requests are not yet completed and clear when they are completed.
15:6	0x000	RsvdZ	Reserved

4.1.10.1.6 VF Link Capabilities Register

Offset: `VF_CFG_PCIE_CAP + 0x0C

Table 4-171 VF Link Capabilities Register

Bits	Default	Attr	Description
3:0	PF	RO	Maximum Link Speed Default value is 0x1 for 2.5 Gbps Link. This field is writable through the DBI for the PF. However, 0x1 is the only supported value. Therefore, the application must not write any value other than 0x1 to this field. Note: For cores supporting Gen2 speed, the following values are accepted: 0001b: 2.5 GHz supported 0010b: 5.0 GHz and 2.5 GHz supported
9:4	PF	RO	Maximum Link Width Default value is the value you specify during hardware configuration (x1, x4, x8, or x16), writable through the DBI for the PF.
11:10	PF	RO	Active State Link PM Support Default value is the value you specify during hardware configuration, writable through the DBI for the PF.

Table 4-171 VF Link Capabilities Register (Continued)

Bits	Default	Attr	Description
14:12	PF	RO	L0s Exit Latency The default value is the value you specify during hardware configuration, writable through the DBI for the PF. This parameter is not visible in coreConsultant.
14:12	PF	RO	L0s Exit Latency; when common clock is used The default value is the value you specify during hardware configuration, writable through the DBI2 for the PF. This parameter is not visible in coreConsultant.
17:15	PF	RO	L1 Exit Latency The default value is the value you specify during hardware configuration, writable through the DBI for the PF. This parameter is not visible in coreConsultant.
17:15	PF	RO	L1 Exit Latency; when common clock is used The default value is the value you specify during hardware configuration, writable through the DBI2 for the PF. This parameter is not visible in coreConsultant.
18	PF	RO	Clock Power Management The default value is the value you specify during hardware configuration, writable through the DBI for the PF. This parameter is not visible in coreConsultant.
19	0x0	RO	Surprise Down Error Reporting Capable Not supported, hardwired to 0x0.
20	PF	RO	Data Link Layer Active Reporting Capable
21	PF	RO	Link Bandwidth Notification Capability
23:22	PF	RsvdP	Reserved
31:24	PF	HwInit	Port Number, writable through the DBI for the PF

4.1.10.1.7 VF Link Control Register

Offset: `VF_CFG_PCIE_CAP + 0x10

Table 4-172 VF Link Control Register

Bits	Default	Attr	Description
1:0	PF	RO	Active State Link PM Control
2	0x0	RO	Reserved
3	0x0	RW	Read Completion Boundary (RCB)
4	0x0	RO	Link Disable Note: This bit is not applicable and is reserved for Endpoints.
5	0x0	RO	Retrain Link Note: This bit is not applicable and is reserved for Endpoints.
6	PF	RO	Common Clock Configuration
7	PF	RO	Extended Synch
8	PF	RO	Enable Clock Power Management
9	0x0	RO	Hardware Autonomous Width Disable Not supported, hardwired to 0.
10	0x0	RO	Link Bandwidth Management Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Bandwidth Management Status bit has been set. Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.
11	0x0	RO	Link Autonomous Bandwidth Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Autonomous Bandwidth Status bit has been set. Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.
15:12	0x00	Rsvd	Reserved

4.1.10.1.8 VF Link Status Register

Offset: `VF_CFG_PCIE_CAP + 0x12

Table 4-173 VF Link Status Register

Bits	Default	Attr	Description
3:0	0	RO	Link Speed The negotiated Link speed.

Table 4-173 VF Link Status Register (Continued)

Bits	Default	Attr	Description
9:4	0	RO	Negotiated Link Width Set automatically by hardware after Link initialization. Value is undefined when link is not up.
10	0	RO	Undefined for PCI Express 1.1 (Was Training Error for PCI Express 1.0a)
11	0	RO	Link Training Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.
12	0	HwInit	Slot Clock Configuration Indicates that the component uses the same physical reference clock that the platform provides on the connector. The default value is the value you select during hardware configuration, writable through the DBI.
13	0	RO	Data Link Layer Active Note: This bit is not applicable and is reserved for Upstream Ports.
14	0	RO	Link Bandwidth Management Status Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.
15	0	RO	Link Autonomous Bandwidth Status Note: This bit is not applicable and is reserved for Endpoints and PCI Express-to-PCI/PCI-X bridges.

4.1.10.1.9 VF Device Capabilities 2 Register

Offset: `VF_CFG_PCIE_CAP + 0x24

Table 4-174 VF Device Capabilities 2 Register

Bits	Default	Attr	Description
3:0	PF	RO	Completion Timeout Ranges Supported This field is applicable only to Root Ports, Endpoints that issue Requests on their own behalf, and PCI Express to PCI/PCI-X Bridges that take ownership of Requests issued on PCI Express.
4	PF	RO	Completion Timeout Disable Supported

4.1.10.1.10 VF Device Control 2 Register

Offset: `CFG_PCIE_CAP + 0x28

Table 4-175 VF Device Control 2 Register

Bits	Default	Attr	Description
3:0	PF	RO	<p>Completion Timeout Value</p> <p>If `CX_CPL_TO_RANGES_ENABLE is defined, the following encodings apply:</p> <ul style="list-style-type: none"> • 0000b Default range: 50 µs to 50 ms • 0001b 50 µs to 100 µs • 0010b 1 ms to 10 ms • 0101b 16 ms to 55 ms • 0110b 65 ms to 210 ms • 1001b 260 ms to 900 ms • 1010b 1 s to 3.5 s • 1101b 4 s to 13 s • 1110b 17 s to 64 s <p>Values not defined above are reserved.</p> <p>If the default range is chosen, the core will have a timeout in the range of 16ms to 55ms.</p> <p>If `CX_CPL_TO_RANGES_ENABLE is not defined, the core will have a timeout in the range of 16ms to 55ms.</p>
4	PF	RO	Completion Timeout Disable

4.1.10.1.11VF Link Control 2 Register

Offset: `VF_CFG_PCIE_CAP + 0x30

Table 4-176 VF Link Control 2 Register

Bits	Default	Attr	Description
3:0	0	RO	<p>Target Link Speed</p> <p>For Downstream ports, this field sets an upper limit on link operational speed by restricting the values advertised by the upstream component in its training sequences:</p> <ul style="list-style-type: none"> • 0001: 2.5Gb/s Target Link Speed • 0010: 5Gb/s Target Link Speed <p>All other encodings are reserved.</p> <p>If a value is written to this field that does not correspond to a speed included in the Supported Link Speeds field, the result is undefined.</p> <p>For both Upstream and Downstream ports, this field is used to set the target compliance mode speed when software is using the Enter Compliance bit to force a link into compliance mode.</p>

Table 4-176 VF Link Control 2 Register (Continued)

Bits	Default	Attr	Description
4	0	RO	<p>Enter Compliance</p> <p>Software is permitted to force a link to enter Compliance mode at the speed indicated in the Target Link Speed field by setting this bit to 1b in both components on a link and then initiating a hot reset on the link.</p> <p>The default value of this field following Fundamental Reset is 0b.</p>
5	0	RO	<p>Hardware Autonomous Speed Disable</p> <p>When cfg_hw_auto_sp_dis signal is asserted, the application must disable hardware from changing the Link speed for device-specific reasons other than attempting to correct unreliable Link operation by reducing Link speed. Initial transition to the highest supported common link speed is not blocked by this signal.</p>
6	0	RO	<p>Selectable De-emphasis</p> <p>When the Link is operating at 5.0 GT/s speed, selects the level of de-emphasis:</p> <ul style="list-style-type: none"> • 1: -3.5 dB • 0: -6 dB <p>When the Link is operating at 2.5 GT/s speed, the setting of this bit has no effect. Components that support only the 2.5 GT/s speed are permitted to hardwire this bit to 0b.</p> <p>Not applicable for an upstream Port or Endpoint device. Hardwired to 0.</p>
9:7	0	RO	<p>Transmit Margin</p> <p>This field controls the value of the non-de-emphasized voltage level at the Transmitter pins:</p> <ul style="list-style-type: none"> • 000: 800-1200 mV for full swing 400-600 mV for half-swing • 001-010: values must be monotonic with a non-zero slope • 011: 200-400 mV for full-swing and 100-200 mV for halfswing • 100-111: reserved <p>This field is reset to 000b on entry to the LTSSM Polling. Compliance substate.</p> <p>Components that support only the 2.5 GT/s speed are permitted to hard-wire this bit to 0b. When operating in 5.0 GT/s mode with full swing, the de-emphasis ratio must be maintained within +/- 1 dB from the specification-defined operational value (either -3.5 or -6 dB).</p>

Table 4-176 VF Link Control 2 Register (Continued)

Bits	Default	Attr	Description
10	0	RO	<p>Enter Modified Compliance When this bit is set to 1b, the device transmits a modified compliance pattern if the LTSSM enters Polling Compliance state.</p>
11	0	RO	<p>Compliance SOS When set to 1b, the LTSSM is required to send SKP Ordered Sets periodically in between the (modified) compliance patterns. Note: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect.</p>
12	0	RO	<p>Compliance De-emphasis This bit sets the de-emphasis level in Polling.Compliance state if the entry occurred due to the Tx Compliance Receive bit being 1b. Encodings:<ul style="list-style-type: none">• 1b: -3.5 dB• 0b: -6 dBNote: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect.</p>
15:13	0	RO	Reserved

4.1.10.1.12VF Link Status 2 Register

Offset: `VF_CFG_PCIE_CAP + 0x32

Table 4-177 VF Link Control 2 Register

Bits	Default	Attr	Description
0	PF	RO	<p>Current De-emphasis Level When the Link is operating at 5 GT/s speed, this bit reflects the level of de-emphasis. Encodings:<ul style="list-style-type: none">• 1b: -3.5 dB• 0b: -6 dBNote: The value in this bit is undefined when the Link is operating at 2.5 GT/s speed and permitted to hardwire this bit to 0b</p>

4.1.10.2 VF MSI-X Capability Registers

The MSI-X Capability structure is an optional capability that can be supported either instead of or in addition to the MSI Capability structure. Even though both MSI-X and MSI structures can exist in the same design, only one can be enabled at a time by software. All VFs in a particular PF must have MSI-X either enabled or disabled, so the MSI-X Capability structure is not optional on a per-VF basis.

The MSI-X Capability structure points to an MSI-X Table and Pending Bit Array (PBA) structure that resides in memory space under a particular BAR (user configurable).



The MSI-X Capability structure is the same structure for all VFs.



In the tables below, a default value of “PF” means that the default value of the VF is the same as the default value of the PF that owns this VF.

Table 4-178 shows the PCI Standard Capability Structures. Refer to the indicated pages for detailed register descriptions.

Table 4-178 VF MSI-X Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VF_CFG_MSIX_CAP	MSI-X Control Register		Next Capability Pointer (`VF_MSIX_NEXT_PTR)	Capability ID (0x11)	432
+0x4	Table Offset (31:3) and BIR (2:0)				433
+0x8	PBA Offset (31:3) and BIR (2:0)				434

4.1.10.2.1 VF MSI-X Capability ID

Offset: `VF_CFG_MSIX_CAP + 0x00

Table 4-179 VF MSI-X Capability ID Register

Bits	Default	Attr	Description
7:0	0x11	RO	MSI-X Capability ID

4.1.10.2.2 VF MSI-X Next Item Pointer

Offset: `VF_CFG_MSIX_CAP + 0x01

Table 4-180 VF MSI-X Next Item Pointer

Bits	Default	Attr	Description
7:0	`VF_MSIX_NEXT_PTR	RO	Next Capability Pointer Points to the VPD capability by default, writable through the DBI.

4.1.10.2.3 VF MSI-X Control Register

Offset: `VF_CFG_MSIX_CAP + 0x02

Table 4-181 VF MSI-X Control Register

Bits	Default	Attr	Description
10:0	`VF_MSIX_TABLE_SIZE	RO	MSI-X Table Size Encoded as (Table Size - 1). For example, a value of 0x003 indicates the MSI-X Table Size is 4.
13:11	000b	RsvdZ	Reserved
14	0	RW	Function Mask <ul style="list-style-type: none"> • 1: All vectors associated with the function are masked, regardless of their respective per-vector Mask bits. • 0: Each vector's Mask bit determines whether the vector is masked or not.
15	0	RW	MSI-X Enable If MSI-X is enabled, MSI and INTx must be disabled.

4.1.10.2.4 VF MSI-X Table Offset and BIR Register

Offset: `VF_CFG_MSIX_CAP + 0x04

Table 4-182 VF MSI-X Table Offset and BIR Register

Bits	Default	Attr	Description
2:0	`VF_MSIX_TABLE_BIR	RO	Table BAR Indicator Register (BIR) Indicates which BAR is used to map the MSI-X Table into memory space: <ul style="list-style-type: none"> • 000: BAR0 • 001: BAR1 • 010: BAR2 • 011: BAR3 • 100: BAR4 • 101: BAR5 • 110: Reserved • 111: Reserved

Table 4-182 VF MSI-X Table Offset and BIR Register (Continued)

Bits	Default	Attr	Description
31:3	`VF_MSIX_TABLE_OFFSET	RO	Table Offset Base address of the MSI-X Table, as an offset from the base address of the BAR indicated by the Table BIR bits.

4.1.10.2.5 VF MSI-X PBA Offset and BIR Register

Offset: `VF_CFG_MSIX_CAP + 0x08

Table 4-183 VF MSI-X PBA Offset and BIR Register

Bits	Default	Attr	Description
2:0	`VF_MSIX_PBA_BIR	RO	Pending Bit Array (PBA) BIR Indicates which BAR is used to map the MSI-X PBA into memory space: <ul style="list-style-type: none">• 000: BAR0• 001: BAR1• 010: BAR2• 011: BAR3• 100: BAR4• 101: BAR5• 110: Reserved• 111: Reserved
31:3	`VF_MSIX_PBA_OFFSET	RO	PBA Offset Base address of the MSI-X PBA, as an offset from the base address of the BAR indicated by the PBA BIR bits.



4.2 PCIe Registers (DM in RC Mode / RC / SW)

This section describes the core implementation of the PCI Express configuration space as an RC core, a DM core configured to operate in RC mode, and a Switch (upstream and downstream ports). Register definition applies to all cases unless otherwise specified.

The topics in this section are:

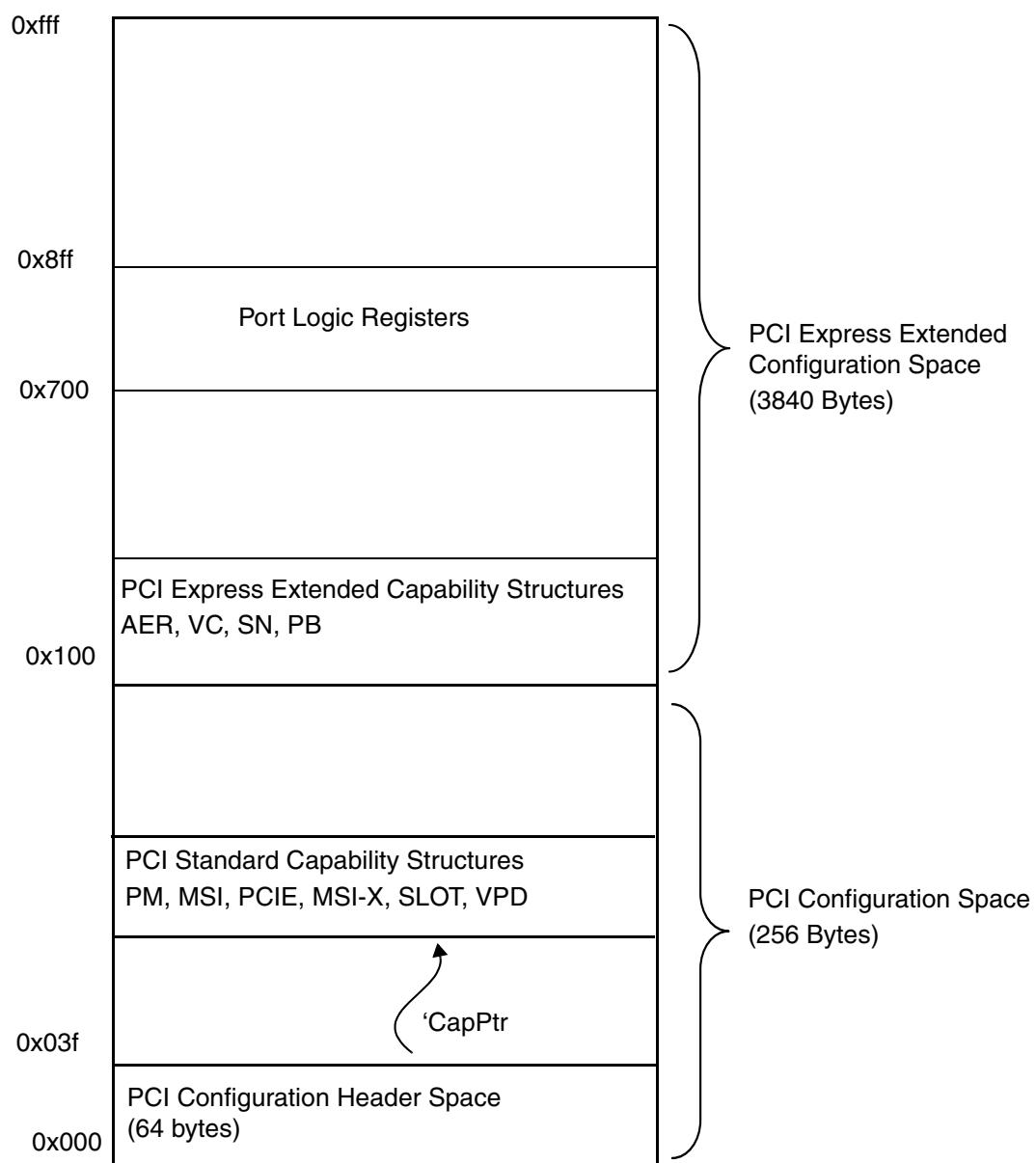
- ❖ [Register Space Layout](#)
- ❖ [Register Maps](#)
- ❖ [Accessing Configuration Registers](#)
- ❖ [Register Default Values](#)
- ❖ [PCI-Compatible Configuration Header Register Details](#)
- ❖ [PCI Standard Capability Structures Register Details](#)
 - ◆ [PCI Power Management Capability Register Details](#)
 - ◆ [MSI Capability Register Details \(DM / RC / SW\)](#)
 - ◆ [PCI Express Capability Register Details \(DM / RC / SW\)](#)
 - ◆ [MSI-X Capability Register Details \(DM / RC / SW\)](#)
 - ◆ [Slot Numbering Capabilities Register Details \(SW\)](#)
 - ◆ [VPD Capabilities Register Details \(DM / RC / SW\)](#)
- ❖ [PCI Express Extended Capabilities Register Details \(DM / RC / SW\)](#)
 - ◆ [Advanced Error Reporting Capability Registers](#)
 - ◆ [Virtual Channel Capability Registers](#)
 - ◆ [Device Serial Number Capability Registers](#)
 - ◆ [Power Budgeting Capability](#)

4.2.1 Register Space Layout

The core has 4096 bytes of PCI Express configuration space per function. For each function, the PCI Express configuration space is divided into:

- ❖ 256 bytes of PCI 3.0 Compatible Configuration Space Header
- ❖ PCI Capabilities structures, which start at offset 0x40
- ❖ PCI Express Extended Configuration Space, which starts at offset 0x100
- ❖ Port Logic registers (vendor-specific registers), which start at offset 0x700. The Port Logic registers have specific pre-defined usages, mostly for test purposes, and can optionally be removed from the core hardware configuration. The usage of the Port Logic registers is the same in both EP mode and RC mode. See “[PCIe Registers: Port Logic](#)” on page [504](#) for details.

[Figure 4-3](#) shows the layout of the core configuration space.

**Figure 4-3 Core Configuration Space Layout: RC Mode and Switch**

4.2.2 Register Maps

Configuration registers are in structures (groups) identified by a Capability ID. Groups are linked together as in PCI. Register locations within a group are specified, but the starting location of each group must be found by traversing the linked list. There are two linked lists of register groups: PCI-compatible base registers and PCI Express Extended Capability registers. PCI-compatible base register groups begin at configuration address stored in capability pointer register at 0x34. PCI Express Extended Capability register groups begin at address 0x100.

4.2.2.1 PCI Configuration Space Header – Type 1

Table 4-1 shows the configuration field register definitions for PCI Express Type 1 Configuration Space header. Most PCI-compatible register fields have the same software interpretation in PCI 3.0 and PCI Express. Refer to the indicated page numbers for detailed register descriptions.

Table 4-184 PCI Configuration Space Header – Type 1

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x00	Device ID		Vendor ID		448
0x04	Status Register		Command Register		448
0x08	Class Code			Revision ID	449
0x0C	BIST(0x00)	Header Type	Latency Timer	Cache Line Size	450
0x10	Base Address Register 0				452
0x14	Base Address Register 1				452
0x18	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	458
0x1C	Secondary Status		I/O Limit	I/O Base	458
0x20	Memory Limit		Memory Base		460
0x24	Prefetchable Memory Limit		Prefetchable Memory Base		460
0x28	Prefetchable Base Upper 32 Bits				461
0x2C	Prefetchable Limit 32 Upper Bits				461
0x30	I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		461
0x34	Reserved			CapPtr	461
0x38	Expansion ROM Base Address				462
0x3C	Bridge Control		Interrupt Pin	Interrupt Line	462

4.2.2.2 Capability Structures Register Maps

The Capability Pointer register in the PCI-compatible header register points to the next item in the linked list of capabilities, which, by default, is the PCI Power Management capabilities register space.

Table 4-2 lists the capabilities supported by the core and their respective default address offsets and next capability pointers.

Table 4-185 Configuration Structure: Starting Addresses and Next Capability Pointers

Start Address Offset	Item	Next Pointer
0x00	PCI-Compatible Header (Type 1)	`CFG_NEXT_PTR
`CFG_PM_CAP	PCI Power Management	`PM_NEXT_PTR
`CFG_MSI_CAP	Message Signaled Interrupt (MSI)	`MSI_NEXT_PTR
`CFG_PCIE_CAP	PCI Express Capabilities	`PCIE_NEXT_PTR
`CFG_MSIX_CAP	MSI-X	`MSIX_NEXT_PTR
`CFG_SLOT_CAP	Slot ID Capability (SW only)	`SLOT_NEXT_PTR
`CFG_VPD_CAP	VPD	`VPD_NEXT_PTR

The following tables show the PCI capability structures. Refer to the indicated pages for detailed register descriptions.

Table 4-186 Power Management Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_PM_CAP	Power Management Capabilities (PMC)		Next Capability Pointer (`PM_NEXT_PTR)	Capability ID (0x01)	464
+0x4	Data	PMCSR_BSE Bridge Extensions	Power Management Control Status Register (PMCSR)		465

Table 4-187 MSI Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_MSI_CAP	Message Control Register		Next Capability Pointer (`MSI_NEXT_PTR)	Capability ID (0x05)	467
+0x4	MSI Lower 32-bit Address Register				467
+0x8	MSI Upper 32-bit Address Register				468
+0xC			MSI Data		468

Table 4-188 PCI Express Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_PCIE_CAP	PCI Express Capabilities Register		Next Capability Pointer (`PCIE_NEXT_PTR)	Capability ID (0x10)	469
+0x4	Device Capabilities				469
+0x8	Device Status		Device Control		470
+0xC	Link Capabilities				472
+0x10	Link Status		Link Control		473
+0x14	Slot Capabilities				475
+0x18	Slot Status		Slot Control		476
+0x1C	Root Capabilities				477
+0x20	Root Status		Root Control		478
+0x24	Device Capabilities 2				478
+0x28	Device Status 2		Device Control 2		479
+0x2C	Link Capabilities 2				
+0x30	Link Status 2		Link Control 2		479
+0x34	Slot Capabilities 2				
+0x38	Slot Status 2		Slot Control 2		
1. Slot Capabilities apply only to downstream ports; for example, RC and downstream of SW.					

Table 4-189 MSI-X Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_MSIX_CAP	MSI-X Control Register		Next Capability Pointer (`MSIX_NEXT_PTR)	Capability ID (0x11)	483
+0x4	Table Offset (31:3) and BIR (2:0)				484
+0x8	PBA Offset (31:3) and BIR (2:0)				484

Table 4-190 Slot Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_SLOT_CAP	Slot Numbering Capabilities		Next Capability Pointer (`SLOT_NEXT_PTR)	Capability ID (0x04)	486

Table 4-191 VPD Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`CFG_VPD_CAP					
+00h	VPD Control and Capabilities register				487
+04h	VPD Data register				487

4.2.2.3 PCI Express Extended Capability Register Maps

The PCI Express Extended Capabilities registers are located in device configuration space at offsets 0x100 or higher. As with PCI Capabilities, the PCI Express Extended Capability structures are allocated using a linked list with a similar method and format to those of PCI.

The Advanced Error Reporting Capability and Virtual Channel Capability are optional extended capabilities that may be implemented by PCI Express devices supporting advanced error control and reporting and multiple VCs, respectively. The Advanced Error Reporting Capability is required when the device supports ECRC generation/checking. The Virtual Channel Capability is required for any device that supports multiple VCs and/or multiple Traffic Classes (TCs).

The following tables outline the PCI Express Extended Capabilities structures. Refer to the indicated pages for detailed register descriptions.

Table 4-192 Advanced Error Reporting Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`AER_PTR	PCI Express Extended Capability Header				489
+0x4	Uncorrectable Error Status Register				489
+0x8	Uncorrectable Error Mask Register				490
+0xC	Uncorrectable Error Severity Register				490
+0x10	Correctable Error Status Register				491
+0x14	Correctable Error Mask Register				491
+0x18	Advanced Error Capabilities and Control Register				492
+0x1C through +0x28	Header Log Registers				492
+0x2C	Root Error Command Register				493
+0x30	Root Error Status Register				493
+0x34	Error Source Identification Register				494

Table 4-193 Virtual Channel Capability Structure

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`VC_PTR	PCI Express Extended Capability Header				495
+0x4	Port VC Capability Register 1				495
+0x8	Port Capability Register 2				495
+0xC	Port VC Status Register		Port VC Control Register		496
+0x10	VC Resource Capability Register (0)				496

1. There is one VC Resource Capability/Control/Status Register N set for each configured VC (in addition to VC0).

Table 4-193 Virtual Channel Capability Structure (Continued)

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0x14	VC Resource Control Register (0)				497
+0x18	VC Resource Status Register (0)		RsvdP		497
0x10+(N*0x0C)	VC Resource Capability Register (N) ¹				498
0x14+(N*0x0C)	VC Resource Control Register (N)				498
0x18+(N*0x0C)	VC Resource Status Register (N)		RsvdP		499

1. There is one VC Resource Capability/Control/Status Register N set for each configured VC (in addition to VC0).

Table 4-194 Device Serial Number Capability Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
`SN_PTR ¹	Device Serial Number Extended Capability Header				500
+0x4	Serial Number Register (1st DWORD)				500
+0x8	Serial Number Register (2nd DWORD)				500

1. Depends on the number of VCs.



4.2.3 Accessing Configuration Registers

The application can access the configuration space through the DBI. Bits [11:0] of the DBI address bus select the target register. Bits [18:16] of the DBI address bus select the target function.

The Attribute column in each register description indicates the read/write access for the register or bit. [Table 4-21](#) defines the read/write attribute abbreviations that are used in the register and bit descriptions throughout this chapter. The definitions match the *PCI Express 2.0 specification*; in the case of a Root Port, all accesses are through the DBI so the host-access information does not apply.

Table 4-195 Configuration Register Bit-Field Types

Attribute	Description
HwInit	<p>Hardware Initialized</p> <p>HwInit bits are controlled by core hardware and are read-only by host system software. Some HwInit bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions. If `CX_DBI_RO_WR_EN = 0, none of the HwInit bits are writable through the DBI.</p>
RO	<p>Read-Only</p> <p>Some RO bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions. If `CX_DBI_RO_WR_EN = 0, none of the RO bits are writable through the DBI.</p>
RW	<p>Read-Write</p> <p>Register bits are read-write and may be read and written normally by the host and the application. Writing from the application side (if any) requires careful synchronization with the host software.</p>
RW1C	<p>Read-Only Status/Write-1-to-Clear Status Register</p> <p>Register bits indicate status when read. A set bit indicating a status event may be cleared by writing a 1. Writing 0 to RW1C bits has no effect. Writing from the application side (if any) requires careful synchronization with host software.</p>
ROS	<p>Sticky Read-Only</p> <p>Registers are read-only and cannot be altered by host or application software, except as noted. Registers are not initialized or modified by a hot reset. A few bits designated as very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.</p>
RWS	<p>Sticky Read-Write</p> <p>Registers are read-write and may be either set or cleared by host or application software to the desired state. Bits are not initialized or modified by a hot reset. A few bits designated very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.</p>
RW1CS	<p>Sticky Read-Only Status/Write-1-to-Clear Status</p> <p>A 1-write clears status for the host and the application, except as noted. Registers indicate status when read. A set bit indicating a status event may be cleared by writing a 1. Writing 0 to RW1CS bits has no effect. Bits are not initialized or modified by a hot reset. A few bits designated very sticky are not cleared by any type of core reset when auxiliary power is supplied and enabled.</p>



Table 4-195 Configuration Register Bit-Field Types (Continued)

Attribute	Description
RsvdP	Reserved and Preserved Reserved for future RW implementations: software must preserve the value read when writing to other bits in the same register.
RsvdZ	Reserved and Zero Reserved for future RW1C implementations: software must write 0 to these bits when writing to other bits in the same register.



Some RO bits are writable from the application through the DBI (if `CX_DBI_RO_WR_EN = 1), as indicated in the register descriptions.



4.2.4 Register Default Values

You can select the default reset values of several of the configuration registers through design configuration parameters. The register descriptions in this chapter indicate the default reset value of the register, either as a specific numerical value or as the name of the configuration parameter that sets the default reset value.

For several of the configuration registers, you can set the default reset values on a per-function basis. Configuration parameters that apply to specific functions include _N at the end of the parameter name. For example, the `CX_DEVICE_ID_0 parameter sets the default value of the Device ID register for function 0, `CX_DEVICE_ID_1 sets the default of the Device ID register for function 1, and so on for each function in your core configuration.



4.2.5 PCI-Compatible Configuration Header Register Details

4.2.5.1 Device ID and Vendor ID Register

Offset: 0x00

[Table 4-23](#) defines the Device ID and Vendor ID register bit assignments. The default values of both Device ID and Vendor ID are hardware configuration parameters. The application can overwrite the default values of both Device ID and Vendor ID through the DBI.

Table 4-196 Device ID and Vendor ID Register

Bits	Default	Attr	Description
15:0	`CX_VENDOR_ID_N	RO	Vendor ID, writable through the DBI
31:16	`CX_DEVICE_ID_N	RO	Device ID, writable through the DBI

4.2.5.2 Command Register

Offset: 0x04

Bytes: 0-1

[Table 4-24](#) defines the PCI-compatible Command register bit assignments.

Table 4-197 Command Register

Bits	Default	Attr	Description
0	0	RW	I/O Space Enable
1	0	RW	Memory Space Enable
2	0	RW	Bus Master Enable
3	0	RO	Special Cycle Enable Not applicable for PCI Express. Must be hardwired to 0.
4	0	RO	Memory Write and Invalidate Not applicable for PCI Express. Must be hardwired to 0.
5	0	RO	VGA Palette Snoop Not applicable for PCI Express. Must be hardwired to 0.
6	0	RW	Parity Error Response
7	0	RO	IDSEL Stepping/ Wait Cycle Control Not applicable for PCI Express. Must be hardwired to 0
8	0	RW	SERR# Enable
9	0	RO	Fast Back-to-Back Enable Not applicable for PCI Express. Must be hardwired to 0.
10	0	RW	INTx Assertion Disable
15:11	0x0	RO	Reserved

4.2.5.3 Status Register

Offset: 0x04

Bytes: 2–3

Table 4-25 defines the PCI-compatible Status register bit assignments.

Table 4-198 Status Register

Bits	Default	Attr	Description
2:0	0x0	RO	Reserved
3	0	RO	INTx Status
4	1	RO	Capabilities List Indicates presence of an extended capability item. Hardwired to 1.
5	0	RO	66 MHz Capable Not applicable for PCI Express. Hardwired to 0.
6	0	RO	Reserved
7	0	RO	Fast Back-to-Back Capable Not applicable for PCI Express. Hardwired to 0.
8	0	RW1C	Master Data Parity Error
10:9	0x0	RO	DEVSEL Timing Not applicable for PCI Express. Hardwired to 0.
11	0	RW1C	Signaled Target Abort
12	0	RW1C	Received Target Abort
13	0	RW1C	Received Master Abort
14	0	RW1C	Signaled System Error
15	0	RW1C	Detected Parity Error

4.2.5.4 Revision ID Register

Offset: 0x08

Byte: 0

Table 4-199 Revision ID Register

Bits	Default	Attr	Description
7:0	`CX_REVISION_ID_N	RO	Revision ID, writable through the DBI

4.2.5.5 Class Code Register

Offset: 0x08

Bytes: 1–3

Table 4-200 Class Code Register

Bits	Default	Attr	Description
7:0	`IF_CODE_N	RO	Programming Interface, writable through the DBI
15:8	`SUB_CLASS_CODE_N	RO	Subclass Code, writable through the DBI
23:16	`BASE_CLASS_CODE_N	RO	Base Class Code, writable through the DBI

4.2.5.6 Cache Line Size Register

Offset: 0x0C

Byte: 0

Table 4-201 Cache Line Size Register

Bits	Default	Attr	Description
7:0	0x00	RW	<p>Cache Line Size</p> <p>The Cache Line Size register is RW for legacy compatibility purposes and is not applicable to PCI Express device functionality. Writing to the Cache Line Size register does not impact functionality of the core.</p>

4.2.5.7 Master Latency Timer Register

Offset: 0x0C

Byte: 1

Table 4-202 Master Latency Timer Register

Bits	Default	Attr	Description
7:0	0x00	RO	<p>Master Latency Timer</p> <p>Not applicable for PCI Express, hardwired to 0.</p>

4.2.5.8 Header Type Register

Offset: 0x0C

Byte: 2

Table 4-203 Header Type Register

Bits	Default	Attr	Description
6:0	0x01	RO	<p>Configuration Header Format</p> <p>Hardwired to 0x01.</p>

**Table 4-203 Header Type Register (Continued)**

Bits	Default	Attr	Description
7	(`CX_NFUNC != 1)	RO	Multi Function Device The default value is 0 for a single function device (`CX_NFUNC = 1) or 1 for a multi-function device (`CX_NFUNC != 1). The Multi Function Device bit is writable through the DBI.

4.2.5.9 BIST Register

Offset: 0x0C

Byte: 3

Table 4-204 BIST Register

Bits	Default	Attr	Description
7:0	0x00	RO	The BIST register functions are not supported by the core. All 8 bits of the BIST register are hardwired to 0.

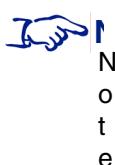
4.2.5.10 Base Address Registers

Offset: 0x10–0x14

The core provides one pair of 32-bit BARs (BAR 0 and BAR 1) for each implemented function. The BARs can be configured as follows:

- ❖ One 64-bit BAR: BARs 0 and 1 are combined to form a single 64-bit BAR.
- ❖ Two 32-bit BARs: BARs 0 and 1 are two independent 32-bit BARs.
- ❖ One 32-bit BAR: BAR 0 is a 32-bit BAR and BAR 1 is either disabled or removed from core altogether to reduce gate count.

The following sections describe how to set up the BAR types and sizes by programming values into the base address registers.



The base address registers (BAR 0 and BAR 1) are included in the core configuration space, but are not expected to be used in a typical Root Complex application.

4.2.5.10.1 General Rules for BAR Setup

The contents of the BARs determine the BAR configuration. The reset values of the BARs are determined by hardware configuration options.

At runtime, application software can overwrite the BAR contents to reconfigure the BARs (unless the affected BAR is removed during hardware configuration). Application software must observe the following rules when writing to the BARs:

- ❖ BARs 0 and 1 can be configured as one 64-bit BAR, two 32-bit BARs, or one 32-bit BAR.
- ❖ Any 32-bit BAR that is not needed can be removed during core hardware configuration to reduce gate count.
- ❖ An I/O BAR must be a 32-bit BAR and cannot be prefetchable.
- ❖ If BAR 0 is configured as a 64-bit BAR:
 - ◆ BAR 1 is the upper 32 bits of the combined 64-bit BAR formed by BARs 0 and 1. Therefore, BAR 1 must be disabled and cannot be configured independently.
 - ◆ BAR 0 must be a memory BAR and can be either prefetchable or non-prefetchable.
 - ◆ The contents of the BAR 0 Mask register determine the number of writable bits in the 64-bit BAR, subject to the restrictions described in “[BAR Mask Registers](#)” on page 455. The BAR 1 Mask register contains the upper 32 bits of the BAR 0 Mask value.
 - ◆ BAR 0 can be disabled by writing 0 to bit 0 of the BAR 0 Mask register (if `BAR0_MASK_WRITABLE_N = 1).
- ❖ If BAR 0 is configured as a 32-bit BAR:
 - ◆ You can configure BAR 1 as an independent 32-bit BAR or remove BAR 1 from the core hardware configuration.
 - ◆ BAR 0 can be configured as a memory BAR or an I/O BAR.



- ◆ The contents of the BAR 0 Mask register determine the number of writable bits in the 32-bit BAR 0, subject to the restrictions described in “[BAR Mask Registers](#)” on page 455.
- ◆ BAR 0 can be disabled by writing 0 to bit 0 of the BAR 0 Mask register (if `BAR0_MASK_WRITABLE_N = 1).
- ❖ When BAR 0 is configured as a 32-bit BAR, BAR 1 is available as an independent 32-bit BAR according to the following rules:
 - ◆ BAR 1 can be configured as a memory BAR or an I/O BAR.
 - ◆ The contents of the BAR 1 Mask register determine the number of writable bits in the 32-bit BAR 1, subject to the restrictions described in “[BAR Mask Registers](#)” on page 455.
 - ◆ BAR 1 can be disabled by writing 0 to bit 0 of the BAR 1 Mask register (if `BAR1_MASK_WRITABLE_N = 1).
 - ◆ If BAR 1 is not required in your design, you can remove BAR 1 from the hardware configuration by setting both `BAR1_ENABLED_N and `BAR1_MASK_WRITABLE_N to 0.

4.2.5.10.2 Base Address Register 0

Offset: 0x10 (if included in the core hardware configuration)

Table 4-205 Base Address Register 0

Bits	Default	Attr	Description
31:4	0x00000000	RO	BAR 0 base address bits (for a 64-bit BAR, the remaining upper address bits are in BAR 1). The BAR 0 Mask value determines which address bits are masked.
3	`PREFETCHABLE0_N for memory BAR 0 for I/O BAR	RO	If BAR 0 is a memory BAR, bit 3 indicates if the memory region is prefetchable: <ul style="list-style-type: none"> • 0 = Non-prefetchable • 1 = Prefetchable If BAR 0 is an I/O BAR, bit 3 is the second least significant bit of the base address. Bits [3:0] are writable through the DBI.
2:1	`BAR0_TYPE_N for memory BAR 00 for I/O BAR	RO	If BAR 0 is a memory BAR, bits [2:1] determine the BAR type: <ul style="list-style-type: none"> • 00 = 32-bit BAR • 10 = 64-bit BAR If BAR 0 is an I/O BAR, bit 2 the least significant bit of the base address and bit 1 is 0. Bits [3:0] are writable through the DBI.
0	`MEM0_SPACE_DECODER_N	RO	<ul style="list-style-type: none"> • 0 = BAR 0 is a memory BAR • 1 = BAR 0 is an I/O BAR Bits [3:0] are writable through the DBI.

4.2.5.10.3 Base Address Register 1

Address: 0x14 (if included in the core hardware configuration)



Table 4-206 Base Address Register 1

Bits	Default	Attr	Description
31:0	Configuration-dependent	RO	If BAR 0 is a 64-bit BAR, BAR 1 contains the upper 32 bits of the BAR 0 base address (bits [63:32]). If BAR 0 is a 32-bit BAR, BAR 1 can be independently programmed as an additional 32-bit BAR or can be excluded from the core hardware configuration. If programmed as an independent 32-bit BAR, the BAR 1 bit definitions are the same as the BAR 0 bit definitions.



4.2.5.11 BAR Mask Registers

The BAR Mask registers determine which bits in each BAR are non-writable by host software, which determines the size of the address space claimed by each BAR.

A BAR Mask register only exists if the corresponding `BARn_MASK_WRITABLE_N value is 1. Otherwise, the `BARn_MASK_N value sets the BAR Mask value in hardware.

The BAR Mask values indicate the range of low-order bits in each implemented BAR *not to use* for address matching. The BAR Mask value also indicates the range of low-order bits in the BAR that *cannot be written* from the host. The application can write to all BAR bits to allow setting of memory, I/O, and other standard BAR options.

To disable any BAR, the application can write a 0 to bit 0 of the corresponding BAR Mask register. To change the BAR Mask value for a disabled BAR, the application must first enable the BAR by writing 1 to bit 0. After enabling the BAR, the application can then write a new value to the BAR Mask register.

If the BAR Mask value for a BAR is less than that required for the BAR type, the core automatically uses the minimum value for the BAR type:

- ❖ BAR bits [11:0] are always masked for a memory BAR. The core requires each memory BAR to claim at least 4 KB.
- ❖ BAR bits [7:0] are always masked for an I/O BAR. The core requires each I/O BAR to claim at least 256 bytes.

4.2.5.11.1 BAR 0 Mask Register

Offset: 0x10 (same as Base Address Register 0, but requires dbi_cs2 for write access)

Table 4-207 BAR 0 Mask Register

Bits	Default	Attr	Description
31:1	`BAR0_MASK_N	Appl. write access only; not readable	<p>Indicates which BAR 0 bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFF to the BAR 0 Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>If BAR 0 is a 64-bit BAR, then the BAR 1 Mask register contains the upper bits of the BAR 0 Mask.</p> <p>The BAR 0 Mask register is invisible to host software and not readable from the application.</p> <p>Application write access depends on the value of `BAR0_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `BAR0_MASK_WRITABLE_N = 1, the BAR 0 Mask register is writable through the DBI. • If `BAR0_MASK_WRITABLE_N = 0, the BAR 0 Mask register is not writable through the DBI.

Table 4-207 BAR 0 Mask Register (Continued)

Bits	Default	Attr	Description
0	`BAR0_ENABLED_N	Same as bits [31:1]	<p>BAR 0 Enable</p> <ul style="list-style-type: none"> • 0: BAR 0 is disabled • 1: BAR 0 is enabled <p>Bit 0 is interpreted as BAR Enable when writing to the BAR Mask register rather than as a mask bit because bit 0 of a BAR is always masked from writing by host software.</p>

4.2.5.11.2 BAR 1 Mask Register

Offset: 0x14 (same as Base Address Register 1, but requires dbi_cs2 for write access)

Table 4-208 BAR 1 Mask Register

Bits	Default	Attr	Description
31:0	Configuration-dependent	Appl. access only	<p>If BAR 0 is a 32-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:1]: BAR 1 Mask value, interpreted the same way as BAR 0 Mask. Default value is `BAR1_MASK_N. • Bit 0: BAR 1 Enable (0 = BAR 1 is disabled; 1= BAR 1 is enabled). Default value is `BAR1_ENABLED_N. • `BAR1_MASK_WRITABLE_N controls application write access to the BAR 1 Mask register. <p>If BAR 0 is a 64-bit BAR:</p> <ul style="list-style-type: none"> • Bits [31:0] are the upper bits of the BAR 0 Mask value. • `BAR0_MASK_WRITABLE_N controls application write access to the full 64-bit BAR 0 Mask register.

4.2.5.11.3 Expansion ROM BAR Mask Register

Offset: 0x38 (same as the Expansion ROM BAR, but requires dbi_cs2 for write access)

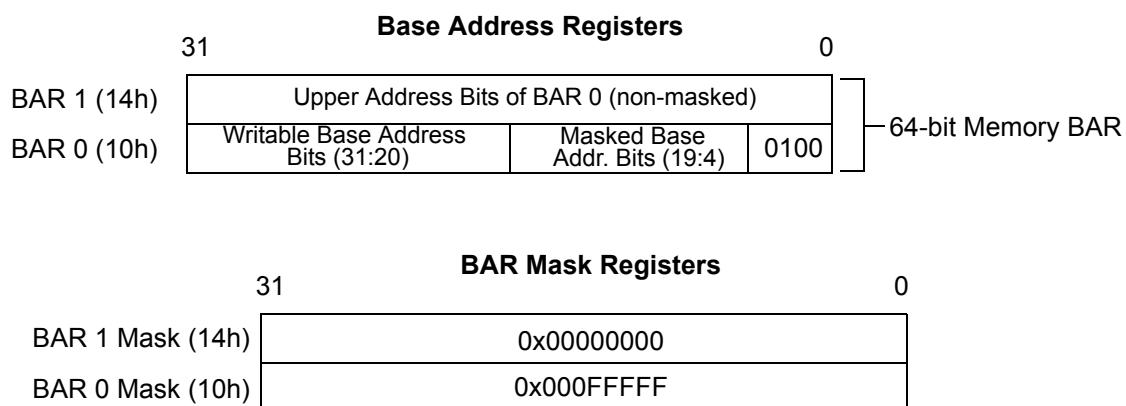
Table 4-209 Expansion ROM BAR Mask Register

Bits	Default	Attr	Description
31:1	`ROM_MASK_N	Appl. access only	<p>Indicates which Expansion ROM BAR bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, writing 0xFFFF to the Expansion ROM BAR Mask register claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.</p> <p>The maximum value is 0xFFFFFFFF because the maximum space that can be claimed by an Expansion ROM BAR is 16 MB.</p> <p>The Expansion ROM BAR Mask register is invisible to host software and not readable from the application. Application access depends on the value of `ROM_MASK_WRITABLE_N:</p> <ul style="list-style-type: none"> • If `ROM_MASK_WRITABLE_N = 1, the Expansion ROM BAR Mask register is writable through the DBI. • If `ROM_MASK_WRITABLE_N = 0, the Expansion ROM BAR Mask register is not writable through the DBI.
0	`ROM_BAR_ENABLED_N	Same as bits [31:1]	<p>Expansion ROM BAR Enable</p> <ul style="list-style-type: none"> • 0: Expansion ROM BAR is disabled • 1: Expansion ROM BAR is enabled

4.2.5.11.4 Example BAR Setup

Figure 4-2 shows an example configuration of the BARs and their corresponding BAR Mask registers. The example configuration includes one 64-bit memory BAR (non-prefetchable).

Figure 4-4 Example Base Address Register Configuration



4.2.5.12 Bus Number Registers

Offset: 0x18

Table 4-210 Bus Number Registers

Bits	Default	Attr	Description
7:0	0x00	RW	Primary Bus Number
15:8	0x00	RW	Secondary Bus Number
23:16	0x00	RW	Subordinate Bus Number
31:24	0x00	RO	Secondary Latency Timer Not applicable to PCI Express, hardwired to 0x00.

4.2.5.13 I/O Base and I/O Limit Register

Offset: 0x1C

Bytes: 0–1

Table 4-211 I/O Base and I/O Limit Register

Bits	Default	Attr	Description
0	`IO_DECODE_32_N	RO	32-Bit I/O Space <ul style="list-style-type: none"> • 0 = 16-bit I/O addressing • 1 = 32-bit I/O addressing This bit is writable through the DBI. When the application writes to this bit through the DBI, the same value is written to bit 8 of this register.
3:1	0x00	RO	Reserved
7:4	0x00	RW	I/O Space Base
8	`IO_DECODE_32_N	RO	32-Bit I/O Space <ul style="list-style-type: none"> • 0 = 16-bit I/O addressing • 1 = 32-bit I/O addressing
11:9	0x00	RO	Reserved
15:12	0x00	RW	I/O Space Limit



4.2.5.14 Secondary Status Register

Offset: 0x1C

Bytes: 2–3

Table 4-212 Secondary Status Register

Bits	Default	Attr	Description
4:0	0x0	RO	Reserved
5	0	RO	66 MHz Capable Not applicable to PCI Express, hardwired to 0.
6	0	RO	Reserved
7	0	RO	Fast Back-to-Back Capable Not applicable to PCI Express, hardwired to 0.
8	0	RW1C	Master Data Parity Error
10:9	0x0	RO	DEVSEL Timing Not applicable to PCI Express, hardwired to 0.
11	0	RW1C	Signaled Target Abort
12	0	RW1C	Received Target Abort
13	0	RW1C	Received Master Abort
14	0	RW1C	Received System Error
15	0	RW1C	Detected Parity Error



4.2.5.15 Memory Base and Memory Limit Register

Offset: 0x20

Table 4-213 Memory Base and Memory Limit Register

Bits	Default	Attr	Description
3:0	0x00	RO	Reserved
15:4	0x00	RW	Memory Base Address
19:16	0x00	RO	Reserved
31:20	0x00	RW	Memory Limit Address

4.2.5.16 Prefetchable Memory Base and Limit Register

Offset: 0x24

Table 4-214 Prefetchable Memory Base and Limit Register

Bits	Default	Attr	Description
0	`MEM_DECODE_64_N	RO	64-Bit Memory Addressing <ul style="list-style-type: none"> • 0 = 32-bit memory addressing • 1 = 64-bit memory addressing This bit is writable through the DBI. When the application writes to this bit through the DBI, the same value is written to bit 16 of this register.
3:1	0x0	RO	Reserved
15:4	0x000	RW	Upper 12 bits of 32-bit Prefetchable Memory Start Address
16	`MEM_DECODE_64_N	RO	64-Bit Memory Addressing <ul style="list-style-type: none"> • 0 = 32-bit memory addressing • 1 = 64-bit memory addressing
19:17	0x0	RO	Reserved
32:20	0x000	RW	Upper 12 bits of 32-bit Prefetchable Memory End Address

4.2.5.17 Prefetchable Base Upper 32 Bits Register

Offset: 0x28

Table 4-215 Prefetchable Base Upper 32 Bits Register

Bits	Default	Attr	Description
31:0	0x00000000	RW	Upper 32 Bits of Base Address of Prefetchable Memory Space Used only when 64-bit prefetchable memory addressing is enabled.

4.2.5.18 Prefetchable Limit Upper 32 Bits Register

Offset: 0x2C

Table 4-216 Prefetchable Limit Upper 32 Bits Register

Bits	Default	Attr	Description
31:0	0x00000000	RW	Upper 32 Bits of Limit Address of Prefetchable Memory Space Used only when 64-bit prefetchable memory addressing is enabled.

4.2.5.19 I/O Base and Limit Upper 16 Bits Register

Offset: 0x30

Table 4-217 I/O Base and Limit Upper 16 Bits Register

Bits	Default	Attr	Description
15:0	0x0000	RW	Upper 16 Bits of I/O Base (if 32-bit I/O decoding is supported for devices on the secondary side)
31:16	0x0000	RW	Upper 16 Bits of I/O Limit (if 32-bit I/O decoding is supported for devices on the secondary side)

4.2.5.20 Capability Pointer Register

Offset: 0x34

Byte: 0

Table 4-218 Capability Pointer Register

Bits	Default	Attr	Description
7:0	`CFG_NEXT_PTR	RO	First Capability Pointer. Points to Power Management Capability structure by default, writable through the DBI

4.2.5.21 Expansion ROM Base Address Register

Offset: 0x38

Table 4-219 Expansion ROM Base Address Register

Bits	Default	Attr	Description
31:11	0x00000	RW	Expansion ROM Address
10:1	0x000	RO	Reserved
0	0x0	RW	Expansion ROM Enable

4.2.5.22 Interrupt Line Register

Offset: 0x3C

Byte: 0

Table 4-220 Interrupt Line Register

Bits	Default	Attr	Description
7:0	0xFF	RW	<p>Interrupt Line</p> <p>Value in this register is system architecture specific. POST software will write the routing information into this register as it initializes and configures the system.</p>

4.2.5.23 Interrupt Pin Register

Offset: 0x3C

Byte: 1

The Interrupt Pin register exists in the core; however, the core does not generate Assert_INTERRUPT or Deassert_INTERRUPT Messages.

Table 4-221 Interrupt Pin Register

Bits	Default	Attr	Description
7:0	`INT_PIN_MAPPING_N	RO	<p>Interrupt Pin</p> <p>Identifies the legacy interrupt Message that the device (or device function) uses. Valid values are:</p> <ul style="list-style-type: none"> • 00h: The device (or function) does not use legacy interrupt • 01h: The device (or function) uses INTA • 02h: The device (or function) uses INTB • 03h: The device (or function) uses INTC • 04h: The device (or function) uses INTD <p>The Interrupt Pin register is writable through the DBI.</p>

4.2.5.24 Bridge Control Register

Offset: 0x3C

Bytes: 2–3

Table 4-222 Bridge Control Register

Bits	Default	Attr	Description
0	0	RW	Parity Error Response Enable
1	0	RW	SERR Enable
2	0	RW	ISA Enable
3	0	RW	VGA Enable
4	0	RW	VGA 16-Bit Decode
5	0	RO	Master Abort Mode Not applicable to PCI Express, hardwired to 0.
6	0	RW	Secondary Bus Reset
7	0	RO	Fast Back-to-Back Transactions Enable Not applicable to PCI Express, hardwired to 0.
8	0	RO	Primary Discard Timer Not applicable to PCI Express, hardwired to 0.
9	0	RO	Secondary Discard Timer Not applicable to PCI Express, hardwired to 0.
10	0	RO	Discard Timer Status Not applicable to PCI Express, hardwired to 0.
11	0	RO	Discard Timer SERR Enable Status Not applicable to PCI Express, hardwired to 0.
15:12	0x0	RO	Reserved

4.2.6 PCI Standard Capability Structures Register Details

4.2.6.1 PCI Power Management Capability Register Details

The core implements power management capabilities. The Capability Pointer field in the configuration header points to the PCI Power Management registers as the first extended capability by default.

The extent of the power management implementation in the core includes:

- ❖ Power Management register space
- ❖ Link state information (provided to both the application logic and PHY interfaces)
- ❖ Power management-ready clock and reset implementation

The following sections describe the PCI Power Management registers implemented in the core. See the *PCI Power Management Specification* and the *PCI Express 2.0 specification* for more details.

4.2.6.1.1 Power Management Capability ID Register

Offset: `CFG_PM_CAP

Table 4-223 Capability ID Register

Bits	Default	Attr	Description
7:0	0x01	RO	Power Management Capability ID

4.2.6.1.2 Power Management Next Item Pointer

Offset: `CFG_PM_CAP + 0x01

Table 4-224 Next Item Pointer

Bits	Default	Attr	Description
7:0	`PM_NEXT_PTR	RO	Next Capability Pointer Points to the MSI capabilities by default, writable through the DBI.

4.2.6.1.3 Power Management Capabilities Register

Offset: `CFG_PM_CAP + 0x02

Table 4-225 Power Management Capabilities Register

Bits	Default	Attr	Description
2:0	3'b011	RO	Power Management Specification Version, writable through the DBI
3	0	RO	PME Clock, hardwired to 0
4	0	RsvdP	Reserved
5	`DEV_SPEC_INIT_N	RO	Device Specific Initialization (DSI), writable through the DBI
8:6	`AUX_CURRENT_N	RO	AUX Current, writable through the DBI

Table 4-225 Power Management Capabilities Register (Continued)

Bits	Default	Attr	Description
9	`D1_SUPPORT_N	RO	D1 Support, writable through the DBI
10	`D2_SUPPORT_N	RO	D2 Support, writable through the DBI
15:11	`PME_SUPPORT_N	RO	<p>PME_Support</p> <p>Identifies the power states from which the core can generate PME Messages. A value of 0 for any bit indicates that the device (or function) is not capable of generating PME Messages while in that power state:</p> <ul style="list-style-type: none"> • Bit 11: If set, PME Messages can be generated from D0 • Bit 12: If set, PME Messages can be generated from D1 • Bit 13: If set, PME Messages can be generated from D2 • Bit 14: If set, PME Messages can be generated from D3_{hot} • Bit 15: If set, PME Messages can be generated from D3_{cold} <p>The PME_Support field is writable through the DBI. Bits 15, 14, and 11 must be Set for PCI-PCI Bridge structures representing Ports on Root Complexes/Switches to indicate that the Bridge will forward PME Messages.</p>

4.2.6.1.4 Power Management Control and Status Register

Offset: `CFG_PM_CAP + 0x04

Table 4-226 Power Management Control and Status Register

Bits	Default	Attr	Description
1:0	0x0	RW	<p>Power State</p> <p>Controls the device power state:</p> <ul style="list-style-type: none"> • 00b: D0 • 01b: D1 • 10b: D2 • 11b: D3 <p>The written value is ignored if the specific state is not supported.</p>
2	0x0	RsvdP	Reserved
3	`DEFAULT_NO_SOFT_RESET_N	RO	No Soft Reset
7:4	0x0	RsvdP	Reserved

Table 4-226 Power Management Control and Status Register (Continued)

Bits	Default	Attr	Description
8	0x0	RWS	PME Enable (sticky bit) A value of 1 indicates that the device is enabled to generate PME.
12:9	0x0	RO	Data Select (not supported)
14:13	0x0	RO	Data Scale (not supported)
15	0x0	RW1CS	PME Status Indicates if a previously enabled PME event occurred or not.
21:16	0x0	RsvdP	Reserved
22	0x0	RO	B2/B3 Support, hardwired to 0
23	0x0	RO	Bus Power/Clock Control Enable, hardwired to 0
31:24	0x00	RO	Data register for additional information (not supported)

4.2.6.2 MSI Capability Register Details (DM / RC / SW)

4.2.6.2.1 MSI Capability ID

Offset: `CFG_MSI_CAP + 0x00

Table 4-227 MSI Capability ID Register

Bits	Default	Attr	Description
7:0	0x05	RO	MSI Capability ID

4.2.6.2.2 MSI Next Item Pointer

Offset: `CFG_MSI_CAP + 0x01

Table 4-228 MSI Next Item Pointer

Bits	Default	Attr	Description
7:0	MSI_NEXT_PTR	RO	Next Capability Pointer Points to PCI Express Capabilities by default, writable through the DBI.

4.2.6.2.3 MSI Control Register

Offset: `CFG_MSI_CAP + 0x02

Table 4-229 MSI Control Register

Bits	Default	Attr	Description
0	0	RW	MSI Enabled When set, INTx must be disabled.
3:1	`DEFAULT_MULTI_MSI_CAPABLE	RO	Multiple Message Capable, writable through the DBI
6:4	0x0	RW	Multiple Message Enabled Indicates that multiple Message mode is enabled by system software. The number of Messages enabled must be less than or equal to the Multiple Message Capable value.
7	`MSI_64_EN	RO	64-bit Address Capable, writable through the DBI
15:8	0x00	RO	Reserved

4.2.6.2.4 MSI Lower 32 Bits Address Register

Offset: `CFG_MSI_CAP + 0x04

Table 4-230 MSI Lower 32 Bits Address Register

Bits	Default	Attr	Description
1:0	0x0	RO	Reserved
31:2	0x00000000	RW	Lower 32-bit Address

4.2.6.2.5 MSI Upper 32 bits Address Register

Offset: `CFG_MSI_CAP + 0x08 (if `MSI_64_EN = 1)

Table 4-231 MSI Upper 32 bits Address Register

Bits	Default	Attr	Description
31:0	0x00000000	RW	Upper 32-bit Address Optional, used only if `MSI_64_EN = 1. If `MSI_64_EN = 0, then the Upper 32-bit Address register functions as the MSI Data register and register address `CFG_MSI_CAP + 0x0C is not used.

4.2.6.2.6 MSI Data Register

Offset: `CFG_MSI_CAP + 0x0C if `MSI_64_EN = 1;

`CFG_MSI_CAP + 0x08 if `MSI_64_EN = 0

Table 4-232 MSI Data Register

Bits	Default	Attr	Description
15:0	0x0000	RW	MSI Data Pattern assigned by system software, bits [4:0] are OR-ed with MSI_VECTOR to generate 32 MSI Messages per function.
31:16	0x0000	RO	Reserved

4.2.6.3 PCI Express Capability Register Details (DM / RC / SW)

The core implements the PCI Express Capability Structure as defined in the *PCI Express 2.0 specification*.

4.2.6.3.1 PCI Express Capability List Register

Offset: `CFG_PCIE_CAP

Table 4-233 PCI Express Capability List Register

Bits	Default	Attr	Description
7:0	0x10	RO	PCI Express Capability ID
15:8	PCIE_NEXT_PTR	RO	Next Capability Pointer Points to the MSI-X capability by default, writable through the DBI

4.2.6.3.2 PCI Express Capabilities Register

Offset: `CFG_PCIE_CAP + 0x02

Table 4-234 PCI Express Capabilities Register

Bits	Default	Attr	Description
3:0	0x2	RO	PCI Express Capability Version
7:4	0100b	RO	Device Port Type
8	`SLOT_IMPLEMENTED_N	HwInit	Slot Implemented, writable through the DBI
13:9	`PCIE_CAP_INT_MSG_NUM_N	RO	Interrupt Message Number Updated by hardware, writable through the DBI.
15:14	0x0	RO	RsvdP

4.2.6.3.3 Device Capabilities Register

Offset: `CFG_PCIE_CAP + 0x04

Table 4-235 Device Capabilities Register

Bits	Default	Attr	Description
2:0	Automatically derived from the value you specify for `CX_MAX_MTU	RO	Max_Payload_Size Supported, writable through the DBI
4:3	0x0	RO	Phantom Function Supported This field is writable through the DBI. However, Phantom Function is not supported. Therefore, the application must not write any value other than 0x0 to this field.

Table 4-235 Device Capabilities Register (Continued)

Bits	Default	Attr	Description
5	`DEFAULT_EXT_TAG_FIELD_SUPPORTED	RO	Extended Tag Field Supported This bit is writable through the DBI. However, if the core supports only 5 bits of TAG, then the application must not write a 1 to this field because the hardware to support more than 32 tags are not implemented.
8:6	0x0	RO	Endpoint L0s Acceptable Latency Must be 0x0 for non-Endpoint devices.
11:9	0x0	RO	Endpoint L1 Acceptable Latency Must be 0x0 for non-Endpoint devices.
12	0x0	RO	Undefined since PCI Express 1.1 (Was Attention Button Present for PCI Express 1.0a)
13	0x0	RO	Undefined since PCI Express 1.1 (Was Attention Indicator Present for PCI Express 1.0a)
14	0x0	RO	Undefined since PCI Express 1.1 (Was Power Indicator Present for PCI Express 1.0a)
15	0x1	RO	Role-Based Error Reporting, writable through the DBI. Required to be set for device compliant to 1.1 spec and later.
17:16	0x0	RsvdP	Reserved
25:18	0x00	RO	Captured Slot Power Limit Value Upstream port only.
27:26	0x0	RO	Captured Slot Power Limit Scale Upstream port only.
31:28	0x0	RsvdP	Reserved

4.2.6.3.4 Device Control Register

Offset: `CFG_PCIE_CAP + 0x08

Table 4-236 Device Control Register

Bits	Default	Attr	Description
0	0x0	RW	Correctable Error Reporting Enable
1	0x0	RW	Non-Fatal Error Reporting Enable

**Table 4-236 Device Control Register (Continued)**

Bits	Default	Attr	Description
2	0x0	RW	Fatal Error Reporting Enable
3	0x0	RW	Unsupported Request Reporting Enable
4	0x1	RW	Enable Relaxed Ordering
7:5	0x0	RW	Max_Payload_Size
8	0x0	RW	Extended Tag Field Enable
9	0x0	RW	Phantom Function Enable
10	0x0	RWS	AUX Power PM Enable
11	`DEFAULT_NO_SNOOP_SUPPORTED_N	RW	Enable No Snoop
14:12	0x2	RW	Max_Read_Request_Size
15	0x0	RsvdP	Reserved

4.2.6.3.5 Device Status Register

Offset: `CFG_PCIE_CAP + 0x0A

Table 4-237 Device Status Register

Bits	Default	Attr	Description
0	0	RW1C	Correctable Error Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
1	0	RW1C	Non-Fatal Error detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
2	0	RW1C	Fatal Error Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
3	0	RW1C	Unsupported Request Detected Errors are logged in this register regardless of whether error reporting is enabled in the Device Control register.
4	0	RO	Aux Power Detected From sys_aux_pwr_det input port.
5	0	RO	Transaction Pending Hard-wired to 0.
15:6	0x000	RsvdZ	Reserved



4.2.6.3.6 Link Capabilities Register

Offset: `CFG_PCIE_CAP + 0x0C

Table 4-238 Link Capabilities Register

Bits	Default	Attr	Description
3:0	0x1	RO	<p>Maximum Link Speed Default value is 0x1 for 2.5 Gbps Link, This field is writable through the DBI. However, 0x1 is the only supported value. Therefore, the application must not write any value other than 0x1 to this field</p> <p>Note: For cores supporting Gen2 speed, the following values are accepted: 0001b: 2.5 GHz supported 0010b: 5.0 GHz and 2.5 GHz supported</p>
9:4	`CX_NL	RO	<p>Maximum Link Width Writable through the DBI.</p>
11:10	`AS_LINK_PM_SUPT	RO	<p>Active State Link PM Support The default value is the value you specify during core configuration, writable through the DBI.</p>
14:12	`DEFAULT_L0S_EXIT_ LATENCY	RO	<p>L0s Exit Latency Writable through the DBI.</p>
14:12	`DEFAULT_COMM_L0S_EXIT_ _LATENCY	RO	<p>L0s Exit Latency; when common clock is used Writable through the DBI2 (not readable).</p>
17:15	`DEFAULT_L1_EXIT_ LATENCY	RO	<p>L1 Exit Latency Writable through the DBI.</p>
17:15	`DEFAULT_COMM_L1_EXIT_L ATENCY	RO	<p>L1 Exit Latency; when common clock is used Writable through the DBI2 (not readable).</p>
18	`DEFAULT_CLK_PM_CAP	RO	<p>Clock Power Management Component can tolerate the removal of refclk via CLKREQ# (if supported). Hardwired to 0 for downstream ports. Writable through the DBI.</p>
19	0x0	RO	<p>Surprise Down Error Reporting Capable Not supported, hardwired to 0x0.</p>
20	*	RO	<p>Data Link Layer Active Reporting Capable Hardwired to 1 for Downstream Ports and 0 for Upstream Ports.</p>

Table 4-238 Link Capabilities Register (Continued)

Bits	Default	Attr	Description
21	*	RO	Link Bandwidth Notification Capability Hardwired to 1 for Downstream Ports and 0 for Upstream Ports.
23:22	0x00	RsvdP	Reserved
31:24	`PORT_NUM	HwInit	Port Number

4.2.6.3.7 Link Control Register

Offset: `CFG_PCIE_CAP + 0x10

Table 4-239 Link Control Register

Bits	Default	Attr	Description
1:0	0x0	RW	Active State Link PM Control
2	0x0	RsvdP	Reserved
3	`ROOT_RCB (RC only) 0x0 (SW only)	RO	Read Completion Boundary (RCB) RC: Writable through DBI SW: Hardwired to 0. Not writable through DBI.
4	0x0	RW	Link Disable This bit is reserved for PCI Express-to-PCI/PCI-X bridges and upstream ports of Switches
5	0x0	RW	Retrain Link This bit is reserved for PCI Express-to-PCI/PCI-X bridges and upstream ports of Switches
6	0x0	RW	Common Clock Configuration
7	0x0	RW	Extended Synch
8	0x0	RW	Enable Clock Power Management Hardwired to 0 if Clock Power Management is disabled in the Link Capabilities register.
9	0x0	RO	Hardware Autonomous Width Disable Not supported, hardwired to 0.

Table 4-239 Link Control Register (Continued)

Bits	Default	Attr	Description
10	0x0	RO	<p>Link Bandwidth Management Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Bandwidth Management Status bit has been set.</p> <p>Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.</p>
11	0x0	RO	<p>Link Autonomous Bandwidth Interrupt Enable When set, this bit enables the generation of an interrupt to indicate that the Link Autonomous Bandwidth Status bit has been set.</p> <p>Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.</p>
15:12	0x00	Rsvd	Reserved

4.2.6.3.8 Link Status Register

Offset: `CFG_PCIE_CAP + 0x12

Table 4-240 Link Status Register

Bits	Default	Attr	Description
3:0	0x1	RO	<p>Link Speed Set automatically by hardware after Link initialization. The value is undefined when link is not up.</p>
9:4	0x1	RO	<p>Negotiated Link Width Set automatically by hardware after Link initialization. The value is undefined when link is not up.</p>
10	0x0	RO	Undefined for PCI Express 1.1 (Was Training Error for PCI Express 1.0a)
11	0	RO	<p>Link Training This bit is not applicable and is reserved for Endpoints, PCI Express to PCI/PCI-X bridges, and Upstream Ports of Switches.</p>
12	`SLOT_CLK_CONFIG	HwInit	<p>Slot Clock Configuration Indicates that the component uses the same physical reference clock that the platform provides on the connector. The default value is the value you select during hardware configuration, writable through the DBI.</p>

Table 4-240 Link Status Register (Continued)

Bits	Default	Attr	Description
13	0x0	RO	<p>Data Link Layer Active</p> <p>This bit must be implemented if the corresponding Data Link Layer Link Active Reporting capability bit is implemented. Otherwise, this bit must be hardwired to 0b.</p>
14	0x0	RW1C	<p>Link Bandwidth Management Status</p> <p>This bit is set by hardware to indicate that either of the following has occurred without the Port transitioning through DL_Down status:</p> <ul style="list-style-type: none"> • A Link retraining has completed following a write of 1b to the Retrain Link bit. <p>Note: This bit is set following any write of 1b to the Retrain Link bit, including when the Link is in the process of retraining for some other reason.</p> <ul style="list-style-type: none"> • Hardware has changed Link speed or width to attempt to correct unreliable Link operation, either through an LTSSM timeout or a higher level process. This bit must be set if the Physical Layer reports a speed or width change was initiated by the Downstream component that was not indicated as an autonomous change. <p>Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.</p>
15	0x0	RW1C	<p>Link Autonomous Bandwidth Status</p> <p>This bit is set by hardware to indicate that hardware has autonomously changed Link speed or width, without the Port transitioning through DL_Down status, for reasons other than to attempt to correct unreliable Link operation. This bit must be set if the Physical Layer reports a speed or a width change was initiated by the Downstream component that was indicated as an autonomous change.</p> <p>Note: This bit is not applicable and is reserved for Endpoints, PCI Express-to-PCI/PCI-X bridges, and Upstream Ports of Switches.</p>

4.2.6.3.9 Slot Capabilities Register

This section applies only to Downstream Ports (for example, RC and downstream ports of SW).

Offset: `CFG_PCIE_CAP + 0x14

Table 4-241 Slot Capabilities Register

Bits	Default	Attr	Description
0	`SLOT_ATTEN_BUTTON_PRESENT	HwInit	Attention Indicator Present, writable through the DBI

Table 4-241 Slot Capabilities Register (Continued)

Bits	Default	Attr	Description
1	`SLOT_PWR_CTRL_PRESENT	HwInit	Power Controller Present, writable through the DBI
2	`SLOT_MRL_SENSOR_PRESENT	HwInit	MRL Sensor Present, writable through the DBI
3	`SLOT_ATTN_IND_PRESENT	HwInit	Attention Indicator Present, writable through the DBI
4	`SLOT_PWR_IND_PRESENT	HwInit	Power Indicator Present, writable through the DBI
5	`SLOT_HP_SURPRISE	HwInit	Hot-Plug Surprise, writable through the DBI
6	`SLOT_HP_CAPABLE	HwInit	Hot-Plug Capable, writable through the DBI
14:7	`SET_SLOT_PWR_LIMIT_VAL	HwInit	Slot Power Limit Value, writable through the DBI
16:15	`SET_SLOT_PWR_LIMIT_SCALE	HwInit	Slot Power Limit Scale, writable through the DBI
17	`SLOT_EML_PRESENT	HwInit	Electromechanical Interlock Present, writable through the DBI
18	`SLOT_NO_CC_SUPPORT	HwInit	No Command Complete Support, writable through the DBI
31:19	`SLOT_PHY_SLOT_NUM	HwInit	Physical Slot Number, writable through the DBI

4.2.6.3.10 Slot Control Register

This section applies only to Downstream Ports (for example, RC and downstream ports of SW).

Offset: `CFG_PCIE_CAP + 0x18

Table 4-242 Slot Control Register

Bits	Default	Attr	Description
0	0x0	RW	Attention Button Pressed Enable
1	0x0	RW	Power Fault Detected Enable
2	0x0	RW	MRL Sensor Changed Enable
3	0x0	RW	Presence Detect Changed Enable
4	0x0	RW	Command Completed Interrupt Enable
5	0x0	RW	Hot-Plug Interrupt Enable

**Table 4-242 Slot Control Register (Continued)**

Bits	Default	Attr	Description
7:6	0x3	RW	Attention Indicator Control
9:8	0x3	RW	Power Indicator Control
10	0x0	RW	Power Controller Control
11	0x0	RW	Electromechanical Interlock Control
12	0x0	RW	Data Link Layer State Changed Enable
15:13	0x0	RsvdP	Reserved

4.2.6.3.11 Slot Status Register

This section applies only to Downstream Ports (for example, RC and downstream ports of SW).

Offset: `CFG_PCIE_CAP + 0x1A

Table 4-243 Slot Status Register

Bits	Default	Attr	Description
0	0x0	RW1C	Attention Button Pressed
1	0x0	RW1C	Power Fault Detected
2	0x0	RW1C	MRL Sensor Changed
3	0x0	RW1C	Presence Detect Changed
4	0x0	RW1C	Command Completed
5	0x0	RO	MRL Sensor State
6	0x0	RO	Presence Detect State
7	0x0	RO	Electromechanical Interlock Status
8	0x0	RW1C	Data Link Layer State Changed
15:9	0x0	RsvdZ	Reserved

4.2.6.3.12 Root Control Register

Offset: `CFG_PCIE_CAP + 0x1C

Table 4-244 Root Control Register

Bits	Default	Attr	Description
0	0x0	RW	System Error on Correctable Error Enable
1	0x0	RW	System Error on Non-fatal Error Enable
2	0x0	RW	System Error on Fatal Error Enable



Table 4-244 Root Control Register (Continued)

Bits	Default	Attr	Description
3	0x0	RW	PME Interrupt Enable
4	0x0	RO	CRS Software Visibility Enable Not supported, hardwired to 0x0.
15:5	0x0	RsvdP	Reserved

4.2.6.3.13 Root Capabilities Register

Offset: `CFG_PCIE_CAP + 0x1E

Table 4-245 Root Capabilities Register

Bits	Default	Attr	Description
0	0x0	RO	CRS Software Visibility Not supported, hardwired to 0x0.

4.2.6.3.14 Root Status Register

Offset: `CFG_PCIE_CAP + 0x20

Table 4-246 Root Status Register

Bits	Default	Attr	Description
15:0	0x0	RO	PME Requester ID
16	0x0	RW1C	PME Status
17	0x0	RO	PME Pending
31:18	0x0	RsvdZ	Reserved

4.2.6.3.15 Device Capabilities 2 Register

Offset: `CFG_PCIE_CAP + 0x24

Table 4-247 Device Capabilities 2 Register

Bits	Default	Attr	Description
3:0	0x0	HWInit	Completion Timeout Ranges Supported This field is applicable only to Root Ports, Endpoints that issue Requests on their own behalf, and PCI Express to PCI/PCI-X Bridges that take ownership of Requests issued on PCI Express. If `CX_CPL_TO_RANGES_ENABLE is defined, then the default value is 0xf (A, B, C and D ranges supported) If `CX_CPL_TO_RANGES_ENABLE is not defined, the default value is 0x0.

Table 4-247 Device Capabilities 2 Register (Continued)

Bits	Default	Attr	Description
4	0x1	RO	Completion Timeout Disable Supported

4.2.6.3.16 Device Control 2 Register

Offset: `CFG_PCIE_CAP + 0x28

Table 4-248 Device Control 2 Register

Bits	Default	Attr	Description
3:0	0x0	RsvdP	<p>Completion Timeout Value If `CX_CPL_TO_RANGES_ENABLE is defined, the following encodings apply:</p> <ul style="list-style-type: none"> • 0000b Default range: 50 µs to 50 ms • 0001b 50 µs to 100 µs • 0010b 1 ms to 10 ms • 0101b 16 ms to 55 ms • 0110b 65 ms to 210 ms • 1001b 260 ms to 900 ms • 1010b 1 s to 3.5 s • 1101b 4 s to 13 s • 1110b 17 s to 64 s <p>Values not defined above are reserved. If the default range is chosen, the core will have a timeout in the range of 16ms to 55ms. If `CX_CPL_TO_RANGES_ENABLE is not defined, the core will have a timeout in the range of 16ms to 55ms.</p>
4	0x1	RW	Completion Timeout Disable

4.2.6.3.17 Link Control 2 Register (Gen2)

Offset: `CFG_PCIE_CAP + 30

Table 4-249 Link Control 2 Register

Bits	Default	Attr	Description
3:0	*	RW	<p>Target Link Speed</p> <p>For Downstream ports, this field sets an upper limit on link operational speed by restricting the values advertised by the upstream component in its training sequences:</p> <ul style="list-style-type: none"> • 0001: 2.5Gb/s Target Link Speed • 0010: 5Gb/s Target Link Speed <p>All other encodings are reserved.</p> <p>If a value is written to this field that does not correspond to a speed included in the Supported Link Speeds field, the result is undefined.</p> <p>*The default value of this field is the highest link speed supported by the component (as reported in the Supported Link Speeds field of the Link Capabilities Register) unless the corresponding platform / form factor requires a different default value.</p> <p>For both Upstream and Downstream ports, this field is used to set the target compliance mode speed when software is using the Enter Compliance bit to force a link into compliance mode.</p>
4	0x0	RWS	<p>Enter Compliance</p> <p>Software is permitted to force a link to enter Compliance mode at the speed indicated in the Target Link Speed field by setting this bit to 1b in both components on a link and then initiating a hot reset on the link.</p> <p>The default value of this field following Fundamental Reset is 0b.</p>
5	0x0	RW	<p>Hardware Autonomous Speed Disable</p> <p>When cfg_hw_auto_sp_dis signal is asserted, the application must disable hardware from changing the Link speed for device-specific reasons other than attempting to correct unreliable Link operation by reducing Link speed. Initial transition to the highest supported common link speed is not blocked by this signal.</p>
6	0x0	RWS	<p>Selectable De-emphasis</p> <p>When the Link is operating at 5.0 GT/s speed, selects the level of de-emphasis:</p> <ul style="list-style-type: none"> • 1: -3.5 dB • 0: -6 dB <p>When the Link is operating at 2.5 GT/s speed, the setting of this bit has no effect. Components that support only the 2.5 GT/s speed are permitted to hardwire this bit to 0b.</p> <p>Default value is implementation-specific, unless a specific value is required for a selected form factor or platform.</p>

Table 4-249 Link Control 2 Register (Continued)

Bits	Default	Attr	Description
9:7	0x000	RW	<p>Transmit Margin This field controls the value of the non-de-emphasized voltage level at the Transmitter pins:</p> <ul style="list-style-type: none"> • 000: 800-1200 mV for full swing 400-600 mV for half-swing • 001-010: values must be monotonic with a non-zero slope • 011: 200-400 mV for full-swing and 100-200 mV for halfswing • 100-111: reserved <p>This field is reset to 000b on entry to the LTSSM Polling. Compliance substate. Components that support only the 2.5 GT/s speed are permitted to hard-wire this bit to 0b. When operating in 5.0 GT/s mode with full swing, the de-emphasis ratio must be maintained within +/- 1 dB from the specification-defined operational value (either -3.5 or -6 dB).</p>
10	0x0	RW	<p>Enter Modified Compliance When this bit is set to 1b, the device transmits modified compliance pattern if the LTSSM enters Polling. Compliance state. Components that support only the 2.5 GT/s speed are permitted to hard-wire this bit to 0b.</p>
11	0x0	RWS	<p>Compliance SOS When set to 1b, the LTSSM is required to send SKP Ordered Sets periodically in between the (modified) compliance patterns. Note: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect. Components that support only 2.5 GT/s speed are permitted to hardwire this bit to 0b.</p>
12	0x0	RWS	<p>This bit sets the de-emphasis level in Polling.Compliance state if the entry occurred due to the Tx Compliance Receive bit being 1b. Encodings:</p> <ul style="list-style-type: none"> • 1b: -3.5 dB • 0b: -6 dB <p>Note: When the Link is operating at 2.5 GT/s, the setting of this bit has no effect.</p>
15:13	0x000	Rsvd	Reserve

4.2.6.3.18 Link Status 2 Register (Gen2)

Offset: `CFG_PCIE_CAP + 32

Table 4-250 Link Status 2 Register

Bits	Default	Attr	Description
0	0x0	RO	<p>Current De-emphasis Level</p> <p>When the Link is operating at 5 GT/s speed, this bit reflects the level of de-emphasis. Encodings:</p> <ul style="list-style-type: none">• 1b: -3.5 dB• 0b: -6 dB <p>Note: The value in this bit is undefined when the Link is operating at 2.5 GT/s speed and permitted to hardwire this bit to 0b.</p>

4.2.6.4 MSI-X Capability Register Details (DM / RC / SW)

The MSI-X Capability structure is an optional capability that can be supported either instead of or in addition to the MSI Capability structure. Even though both structures can exist in the same design, only one can be enabled at a time by software.

MSI-X increases the flexibility of MSI by allowing multiple vectors with different destination addresses. This overcomes the 32-vector limit of MSI and provides additional flexibility.

The MSI-X Capability structure points to an MSI-X Table and Pending Bit Array (PBA) structure that resides in memory space under a particular BAR (user configurable).

The MSI-X Capability structure is implemented inside the CDM as part of the capabilities linked list. The application designer is responsible for configuring the core with values for Table and PBA Offset and BIR numbers.

4.2.6.4.1 MSI-X Capability ID

Offset: `CFG_MSIX_CAP + 0x00

Table 4-251 MSI-X Capability ID Register

Bits	Default	Attr	Description
7:0	0x11	RO	MSI-X Capability ID

4.2.6.4.2 MSI-X Next Item Pointer

Offset: `CFG_MSIX_CAP + 0x01

Table 4-252 MSI-X Next Item Pointer

Bits	Default	Attr	Description
7:0	MSIX_NEXT_PTR	RO	Next Capability Pointer Points to the VPD capability by default, writable through the DBI.

4.2.6.4.3 MSI-X Control Register

Offset: `CFG_MSIX_CAP + 0x02

Table 4-253 MSI-X Control Register

Bits	Default	Attr	Description
10:0	`MSIX_TABLE_SIZE_N	RO	MSI-X Table Size Encoded as (Table Size - 1). For example, a value of 0x003 indicates the MSI-X Table Size is 4.
13:11	000b	RsvdZ	Reserved

Table 4-253 MSI-X Control Register (Continued)

Bits	Default	Attr	Description
14	0	RW	<p>Function Mask</p> <ul style="list-style-type: none"> • 1: All vectors associated with the function are masked, regardless of their respective per-vector Mask bits. • 0: Each vector's Mask bit determines whether the vector is masked or not.
15	0	RW	<p>MSI-X Enable</p> <p>If MSI-X is enabled, MSI and INTx must be disabled.</p>

4.2.6.4.4 MSI-X Table Offset and BIR Register

Offset: `CFG_MSIX_CAP + 0x04

Table 4-254 MSI-X Table Offset and BIR Register

Bits	Default	Attr	Description
2:0	`MSIX_TABLE_BIR_N	RO	<p>Table BAR Indicator Register (BIR)</p> <p>Indicates which BAR is used to map the MSI-X Table into memory space:</p> <ul style="list-style-type: none"> • 000: BAR0 • 001: BAR1 • 010: BAR2 • 011: BAR3 • 100: BAR4 • 101: BAR5 • 110: Reserved • 111: Reserved
31:3	`MSIX_TABLE_OFFSET_N	RO	<p>Table Offset</p> <p>Base address of the MSI-X Table, as an offset from the base address of the BAR indicated by the Table BIR bits.</p>

4.2.6.4.5 MSI-X PBA Offset and BIR Register

Offset: `CFG_MSIX_CAP + 0x08

Table 4-255 MSI-X PBA Offset and BIR Register

Bits	Default	Attr	Description
2:0	`MSIX_PBA_BIR_N	RO	<p>Pending Bit Array (PBA) BIR Indicates which BAR is used to map the MSI-X PBA into memory space:</p> <ul style="list-style-type: none"> • 000: BAR0 • 001: BAR1 • 010: BAR2 • 011: BAR3 • 100: BAR4 • 101: BAR5 • 110: Reserved • 111: Reserved
31:3	`MSIX_PBA_OFFSET_N	RO	<p>PBA Offset Base address of the MSI-X PBA, as an offset from the base address of the BAR indicated by the PBA BIR bits.</p>

4.2.6.5 Slot Numbering Capabilities Register Details (SW)

The Slot Numbering Capability contains a single 32-bit register as described below and further defined in the *PCI-to-PCI Bridge Architecture Specification*.

4.2.6.5.1 Slot Numbering Capabilities Register

Offset: `CFG_SLOT_CAP

Table 4-256 Slot Numbering Capabilities Register

Bits	Default	Attr	Description
7:0	0x04	RO	Slot Numbering Capabilities ID Hardwired to 0x04.
15:8	`SLOT_NEXT_PTR	RO	Next Capability Pointer Points to the VPD capability by default, writable through the DBI.
20:16	`SLOT_NUM	RO	Add-in Card Slots Provided
21	`FIRST_IN_CHASSIS	RO	First in Chassis
23:22	0x0	RO	Reserved
31:24	0x00	RW	Chassis Number

4.2.6.6 VPD Capabilities Register Details (DM / RC / SW)

The VPD interface follows the *PCI Base Specification*. The cfg_vpd_int signal pulses when the host has written to this register.

4.2.6.6.1 VPD Control and Capabilities Register

Offset: `CFG_VPD_CAP + 00h

Table 4-257 VPD Control and Capabilities Register

Bit(s)	Type	Reset	Description
31	RW	0	VPD Flag
30:16	RW	0	VPD Address
15:8	HwInit	0	Next Capability Pointer Points to end of capabilities by default, writable through the DBI.
7:0	HwInit	03h	VPD Capability ID

4.2.6.6.2 VPD Data Register

Offset: `CFG_VPD_CAP + 04h

VPD data is updated by way of a host-application handshake, as described in “[Vital Product Data \(VPD\) Support \(optional\)](#)” on page [193](#).

Table 4-258 VPD Control and Capabilities Register

Bit(s)	Type	Reset	Description
31:0	RW	0	VPD Data

4.2.7 PCI Express Extended Capabilities Register Details (DM / RC / SW)

The core implements the following PCI Express Extended Capabilities registers:

- ❖ Advanced Error Reporting Capability register set
- ❖ Virtual Channel Capability register set – only exists in the first function of a multi-function device
- ❖ Device Serial Number Capability register set – only exists in the first function of a multi-function device
- ❖ Power Budgeting Capability register set

The following sections describe the individual registers in each group. For register maps, see “[Register Maps](#)” on page 438.

4.2.7.1 Advanced Error Reporting Capability Registers

4.2.7.1.1 PCI Express Extended Capability Header

Address: 0x100

Table 4-259 PCI Express Extended Capability Header

Bits	Default	Attr	Description
15:0	0x1	RO	PCI Express Extended Capability ID Value is 0x1 for Advanced Error Reporting.
19:16	0x1	RO	Capability Version
31:20	0x140	RO	Next Capability Offset Points to the Virtual Channel Capability structure by default.

4.2.7.1.2 Uncorrectable Error Status Register

Offset: 0x100 + 0x04

Table 4-260 Uncorrectable Error Status Register

Bits	Default	Attr	Description
0	0x0	Undefined	Undefined for PCI Express 1.1 (Was Training Error Status for PCI Express 1.0a)
3:1	0x0	RsvdZ	Reserved
4	0x0	RW1CS	Data Link Protocol Error Status
5	0x0	RO	Surprise Down Error Status (not supported)
11:6	0x00	RsvdZ	Reserved
12	0x0	RW1CS	Poisoned TLP Status
13	0x0	RW1CS	Flow Control Protocol Error Status
14	0x0	RW1CS	Completion Timeout Status
15	0x0	RW1CS	Completer Abort Status
16	0x0	RW1CS	Unexpected Completion Status
17	0x0	RW1CS	Receiver Overflow Status
18	0x0	RW1CS	Malformed TLP Status
19	0x0	RW1CS	ECRC Error Status
20	0x0	RW1CS	Unsupported Request Error Status
31:21	0x000	RsvdZ	Reserved

4.2.7.1.3 Uncorrectable Error Mask Register

Offset: 0x100 + 0x08

Table 4-261 Uncorrectable Error Mask Register

Bits	Default	Attr	Description
0	0	Undefined	Undefined for PCI Express 1.1 (Was Training Error Mask for PCI Express 1.0a)
3:1	0x0	RsvdP	Reserved
4	0	RWS	Data Link Protocol Error Mask
5	0x0	RO	Surprise Down Error Mask (not supported)
11:6	0x00	RsvdP	Reserved
12	0	RWS	Poisoned TLP Mask
13	0	RWS	Flow Control Protocol Error Mask
14	0	RWS	Completion Timeout Mask
15	0	RWS	Completer Abort Mask
16	0	RWS	Unexpected Completion Mask
17	0	RWS	Receiver Overflow Mask
18	0	RWS	Malformed TLP Mask
19	0	RWS	ECRC Error Mask
20	0	RWS	Unsupported Request Error Mask
31:21	0x000	RsvdP	Reserved

4.2.7.1.4 Uncorrectable Error Severity Register

Offset: 0x100 + 0x0C

Table 4-262 Uncorrectable Error Severity Register

Bits	Default	Attr	Description
0	0x1	Undefined	Undefined for PCI Express 1.1 (Was Training Error Severity for PCI Express 1.0a)
3:1	0x0	RsvdP	Reserved
4	0x1	RWS	Data Link Protocol Error Severity
5	0x1	RO	Surprise Down Error Severity (not supported)
11:6	0x00	RsvdP	Reserved
12	0x0	RWS	Poisoned TLP Severity

Table 4-262 Uncorrectable Error Severity Register (Continued)

Bits	Default	Attr	Description
13	0x1	RWS	Flow Control Protocol Error Severity
14	0x0	RWS	Completion Timeout Severity
15	0x0	RWS	Completer Abort Severity
16	0x0	RWS	Unexpected Completion Severity
17	0x1	RWS	Receiver Overflow Severity
18	0x1	RWS	Malformed TLP Severity
19	0x0	RWS	ECRC Error Severity
20	0x0	RWS	Unsupported Request Error Severity
31:21	0x000	RsvdP	Reserved

4.2.7.1.5 Correctable Error Status Register

Offset: 0x100 + 0x10

Table 4-263 Correctable Error Status Register

Bits	Default	Attr	Description
0	0x0	RW1CS	Receiver Error Status
5:1	0x00	RsvdZ	Reserved
6	0x0	RW1CS	Bad TLP Status
7	0x0	RW1CS	Bad DLLP Status
8	0x0	RW1CS	REPLAY_NUM Rollover Status
11:9	0x00	RsvdZ	Reserved
12	0x0	RW1CS	Reply Timer Timeout Status
13	0x0	RW1CS	Advisory Non-Fatal Error Status
31:14	0x00000	RsvdZ	Reserved

4.2.7.1.6 Correctable Error Mask Register

Offset: 0x100 + 0x14

Table 4-264 Correctable Error Mask Register

Bits	Default	Attr	Description
0	0	RWS	Receiver Error Mask

Table 4-264 Correctable Error Mask Register (Continued)

Bits	Default	Attr	Description
5:1	0x00	RsvdP	Reserved
6	0	RWS	Bad TLP Mask
7	0	RWS	Bad DLLP Mask
8	0	RWS	REPLAY_NUM Rollover Mask
11:9	0x0	RsvdP	Reserved
12	0	RWS	Reply Timer Timeout Mask
13	0x1	RWS	Advisory Non-Fatal Error Mask
31:14	0x000000	RsvdP	Reserved

4.2.7.1.7 Advanced Capabilities and Control Register

Offset: 0x100 + 0x18

Table 4-265 Advanced Capabilities and Control Register

Bits	Default	Attr	Description
4:0	0x00	ROS	First Error Pointer
5	`DEFAULT_ECRC_GEN_CAP_N	RO	ECRC Generation Capability
6	0	RWS	ECRC Generation Enable
7	`DEFAULT_ECRC_CHK_CAP_N	RO	ECRC Check Capable
8	0	RWS	ECRC Check Enable
31:9	0x0000000	RsvdP	Reserved

4.2.7.1.8 Header Log Registers

Offset: 0x100 + 0x1C

The Header Log registers collect the header for the TLP corresponding to a detected error. See the *PCI Express 2.0 specification* for details. Each of the Header Log registers is type ROS; the default reset value of each Header Log register is 0x00000000.

Table 4-266 Header Log Register

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Header Log Register (first DWORD)			
0x20	Header Log Register (second DWORD)			
0x24	Header Log Register (third DWORD)			

Table 4-266 Header Log Register (Continued)

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x28	Header Log Register (fourth DWORD)			

4.2.7.1.9 Root Error Command Register

Offset: 0x100 + 0x2C

Table 4-267 Root Error Command Register

Bits	Default	Attr	Description
0	0x0	RW	Correctable Error Reporting Enable
1	0x0	RW	Non-Fatal Error Reporting Enable
2	0x0	RW	Fatal Error Reporting Enable
31:3	0x00000000	RsvdP	Reserved

4.2.7.1.10 Root Error Status Register

Offset: 0x100 + 0x30

Table 4-268 Root Error Status Register

Bits	Default	Attr	Description
0	0x0	RW1CS	ERR_COR Received
1	0x0	RW1CS	Multiple ERR_COR Received
2	0x0	RW1CS	ERR_FATAL/NONFATAL Received
3	0x0	RW1CS	Multiple ERR_FATAL/NONFATAL Received
4	0x0	RW1CS	First Uncorrectable Fatal
5	0x0	RW1CS	Non-Fatal Error Messages Received
6	0x0	RW1CS	Fatal Error Messages Received
26:7	0x00000	RsvdZ	Reserved
31:27	`AER_INT_MSG_NUM_N	RO	Advanced Error Interrupt Message Number, writable through the DBI

4.2.7.1.11 Error Source Identification Register

Offset: 0x100 + 0x34

Table 4-269 Error Source Identification Register

Bits	Default	Attr	Description
15:0	0x0	ROS	ERR_COR Source Identification
31:16	0x0	ROS	ERR_FATAL/NONFATAL Source Identification

4.2.7.2 Virtual Channel Capability Registers

4.2.7.2.1 VC Extended Capability Header

Offset: 0x140

Table 4-270 VC Extended Capability Header

Bits	Default	Attr	Description
15:0	0x2	RO	PCI Express Extended Capability The default value is 0x2 for VC Capability.
19:16	0x1	RO	Capability Version
31:20	0x000	RO	Next Capability Offset Points to the Device Serial Number Capability structure by default, if enabled. Default value should be 'VC_NEXT_PTR'.

4.2.7.2.2 Port VC Capability Register 1

Offset: 0x140 + 0x4

Table 4-271 Port VC Capability Register 1

Bits	Default	Attr	Description
2:0	`DEFAULT_EXT_VC_CNT	RO	Extended VC Count The default value is the one less than the number of VCs that you specify during hardware configuration (`CX_NVC - 1).
3	0x0	RsvdP	Reserved
6:4	`DEFAULT_LOW_PRI_ EXT_VC_CNT	RO	Low Priority Extended VC Count, writable through the DBI
7	0x0	RsvdP	Reserved
9:8	0x0	RO	Reference Clock
11:10	0x0	RO	Port Arbitration Table Entry Size
31:12	0x0	RsvdP	Reserved

4.2.7.2.3 Port VC Capability Register 2

Offset: 0x140 + 0x8

Table 4-272 Port VC Capability Register 2

Bits	Default	Attr	Description
7:0	`DEFAULT_VC_ARB_32	RO	<p>VC Arbitration Capability Indicates which VC arbitration mode(s) the device supports, writable through the DBI:</p> <ul style="list-style-type: none"> Bit 0: Device supports hardware fixed arbitration scheme. For the core, the scheme is 16-phase weighted round robin (WRR). Bit 1: Device supports 32-phase WRR Bit 2: Device supports 64-phase WRR Bit 3: Device supports 128-phase WRR Bits 4–7: Reserved
23:8		RsvdP	Reserved
31:24	0x00	RO	<p>VC Arbitration Table Offset (not supported) The default value is 0x00 (no arbitration table present).</p>

4.2.7.2.4 Port VC Control Register

Offset: 0x140 + 0xC

Bytes: 0–1

Table 4-273 Port VC Control Register

Bits	Default	Attr	Description
0	0x0	RW	Load VC Arbitration Table
3:1	0x0	RW	VC Arbitration Select
15:4	0x0	RsvdP	Reserved

4.2.7.2.5 Port VC Status Register

Offset: 0x140 + 0xC

Bytes: 2–3

Table 4-274 Port VC Status Register

Bits	Default	Attr	Description
0	0x0	RO	Arbitration Table Status
15:1	0x0	RsvdZ	Reserved

4.2.7.2.6 VC Resource Capability Register (0)

Offset: 0x140 + 0x10

Table 4-275 VC Resource Capability Register (0)

Bits	Default	Attr	Description
7:0	0x00	RO	Port Arbitration Capability
13:8	0x00	RsvdP	Reserved
14	0x0	RO	Undefined for PCI Express 1.1 (Was Advanced Packet Switching for PCI Express 1.0a)
15	0x0	HwInit	Reject Snoop Transactions
22:16	0x0	HwInit	Maximum Time Slots
23	0x0	RsvdP	Reserved
31:24	0x00	RO	Port Arbitration Table Offset

4.2.7.2.7 VC Resource Control Register (0)

Offset: 0x140 + 0x14

Table 4-276 VC Resource Control Register (0)

Bits	Default	Attr	Description
7:0	0xFF	RW	TC/VC Map Bit 0 is hardwired to 1; bits 7:1 are RW.
15:8		RsvdP	Reserved
16	0x0	RW	Load Port Arbitration Table
19:17	0x0	RW	Port Arbitration Select
23:20	0x0	RsvdP	Reserved
26:24	0x0	RO	VC ID Hardwired to 0 for VC0.
30:27		RsvdP	Reserved
31	0x1	RO	VC Enable Hardwired to 1 for the first VC.

4.2.7.2.8 VC Resource Status Register (0)

Offset: 0x140 + 0x18

Table 4-277 VC Resource Status Register (0)

Bits	Default	Attr	Description
15:0	0x0	RsvdP	Reserved

Table 4-277 VC Resource Status Register (0) (Continued)

Bits	Default	Attr	Description
16	0x0	RO	Port Arbitration Table Status
17	0x1	RO	VC Negotiation Pending
31:18	0x00	RsvdZ	Reserved

4.2.7.2.9 VC Resource Capability Register (N)

Offset: $0x140 + 0x10 + (N * 0x0C)$

There is a VC Resource Capability Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-278 VC Resource Capability Register (N)

Bits	Default	Attr	Description
7:0	0x00	RO	Port Arbitration Capability
13:8	0x00	RsvdP	Reserved
14	0x0	RO	Undefined for PCI Express 1.1 (Was Advanced Packet Switching for PCI Express 1.0a)
15	0x0	HwInit	Reject Snoop Transactions Not currently supported, must be 0x0.
22:16	0x0	HwInit	Maximum Time Slots
23		RsvdP	Reserved
31:24	0x00	RO	Port Arbitration Table Offset

4.2.7.2.10 VC Resource Control Register (N)

Offset: $0x140 + 0x14 + (N * 0x0C)$

There is a VC Resource Control Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-279 VC Resource Control Register (N)

Bits	Default	Attr	Description
7:0	0x00	RW	TC/VC Map
15:8		RsvdP	Reserved
16	0x0	RW	Load Port Arbitration Table
19:17	0x0	RW	Port Arbitration Select
23:20	0x0	RsvdP	Reserved
26:24	0x0	RW	VC ID

Table 4-279 VC Resource Control Register (N) (Continued)

Bits	Default	Attr	Description
30:27		RsvdP	Reserved
31	0x0	RW	VC Enable

4.2.7.2.11 VC Resource Status Register (N)Offset: $0x140 + 0x18 + (N * 0x0C)$

There is a VC Resource Status Register (N) for each VC in your core configuration, in addition to VC0.

Table 4-280 VC Resource Status Register (N)

Bits	Default	Attr	Description
15:0	0x0	RsvdP	Reserved
16	0x0	RO	Port Arbitration Table Status
17	0x0	RO	VC Negotiation Pending
31:18	0x00	RsvdZ	Reserved

4.2.7.3 Device Serial Number Capability Registers

4.2.7.3.1 Device Serial Number Extended Capability Header

Offset: `SN_PTR (depends on the number of VCs)

Table 4-281 Device Serial Number Extended Capability Header

Bit(s)	Default	Attr	Description
15:0	0x0003	HwInit	Extended Capability ID
19:16	0x1	HwInit	Capability Version
31:20	0x0	HwInit	Points to end of capabilities by default, writable through the DBI

4.2.7.3.2 Serial Number Registers

Offset: `SN_PTR + 0x4/+0x8

Table 4-282 Serial Number Registers

Offset	Default	Attr	Description
0x4	Parameter	RO	Serial Number register (1st DWORD), writable through the DBI
0x8	Parameter	RO	Serial Number register (2nd DWORD), writable through the DBI

4.2.7.4 Power Budgeting Capability

The PCI Power Budgeting Capability allows the system to properly allocate power to devices that are added to the system at runtime. Through this capability, a device can report the power it consumes on a variety of power rails, in a variety of device power management states, and in a variety of operating conditions. The system uses this information to ensure that the system is capable of providing the proper power and cooling levels to the device. Failure to properly indicate device power consumption may risk device or system failure.

Implementation of the Power Budgeting Capability is optional for PCIe devices that are implemented either in a form factor that does not require Hot-Plug support, or that are integrated on the system board. PCIe form factor specifications may require support for power budgeting.

4.2.7.4.1 Power Budgeting Extended Capability Header

Offset: `PB_PTR

Refer to section 7.1.5 in rev. 0.7 of the PCI Express 2.0 specification for a description of the Power Budgeting Capability.

Table 4-283 Power Budgeting Extended Capability Header

Bits	Default	Attr	Description
15:0	0x0004	RO	Extended Capability ID This field is a PCI-SIG defined ID number that indicates the nature and format of the extended capability. Extended Capability ID for Power Budgeting capability is 0x0004.
19:16	0x1	RO	Capability Version This field is a PCI-SIG defined version number that indicates the version of the capability structure present. Must be 0x1 as per the <i>PCI Express 2.0 specification</i> .
31:20	`PB_NEXT_PTR	RO	Next Capability Offset Points to the next capability structure. Default value should be `PB_NEXT_PTR

4.2.7.4.2 Data Select Register

Offset: `PB_PTR + 0x4

This read/write register indexes the Power Budgeting data reported through the Data register and selects the DWORD of Power Budgeting Data that should appear in the Data register. Index values for this register start at 0 to select the first DWORD of Power budgeting Data; subsequent DWORDs of power budgeting data are selected by increasing index values.

4.2.7.4.3 Data Register

Offset: `PB_PTR + 0x8

Table 4-284 Data Select Register

Bits	Default	Attr	Description
7:0	TBD	RO	<p>Base Power</p> <p>Specifies (in watts) the base power value in the given operating condition. This value must be multiplied by the data scale to produce the actual power consumption value.</p>
9:8	TBD	RO	<p>Data Scale</p> <p>Specifies the scale to apply to the Base Power value. The device power consumption is determined by multiplying the contents of the Base Power register field with the value corresponding to the encoding returned by this field:</p> <ul style="list-style-type: none"> 00: 1.0x 01: 0.1x 10: 0.01x 11: 0.001x
12:10	TBD	RO	<p>PM Sub State</p> <p>Specifies the power management sub state of the operating condition being described:</p> <ul style="list-style-type: none"> 000: Default sub state 001 - 111: Device-specific sub state
14:13	TBD	RO	<p>PM State</p> <p>Specifies the power management state of the operating condition being described:</p> <ul style="list-style-type: none"> 00: D0 01: D1 10: D2 11: D3 <p>A device returns 11b in this field and Aux or PME Aux in the Type register to specify the D3-Cold PM state. An encoding of 11b along with any other Type register value specifies the D3-Hot state.</p>
17:15	TBD	RO	<p>Type</p> <p>Specifies the power rail of the operating condition being described:</p> <ul style="list-style-type: none"> 000: Power (12V) 001: Power (3.3V) 010: Power (1.8V) 111: Thermal <p>All other encodings are reserved.</p>

**Table 4-284 Data Select Register (Continued)**

Bits	Default	Attr	Description
20:18	TBD	RO	<p>Power Rail</p> <p>Specifies the type of the operating condition being described:</p> <ul style="list-style-type: none"> 000: PME Aux 001: Auxiliary 010: Idle 011: Sustained 111: Maximum <p>All other encodings are reserved.</p>
31:21	TBD	RsvdP	Reserved

4.2.7.4.4 Power Budget Capability Register

Offset: `PB_PTR + 0xC

This register indicates the power budgeting capability of a device.

Table 4-285 Power Budget Capability Register

Bits	Default	Attr	Description
0	DEFAULT_ PWR_BUDGET_ SYS_ALLOC	HwInit	<p>System Allocated</p> <p>This bit when set, indicates that the power budget for the device is included within the system power budget. Reported power budgeting data for this device should be ignored by software for power budgeting decisions if this bit is set.</p>
7:1	TBD	RsvdP	Reserved



4.3 PCIe Registers: Port Logic

The Port Logic registers are vendor-specific registers and are used mainly for status reporting and testing. The usage of the Port Logic registers is the same in both EP and RC modes and in the Switch. The Port Logic registers reside in the application register section of the configuration space starting at address 0x700. Port logic registers can be configured to map into a memory BAR0 of function0, which is enabled by ENABLE_MEM_MAP_PL_REG.

The Port Logic registers are implemented by default, but they can be optionally be removed from the core hardware configuration. You can exclude the Port Logic registers from your core by selecting the Remove Port Logic Registers configuration option ('CX_PL_REG_DISABLE = 1').

This section describes the Port Logic register map and the usage of each Port Logic register. The topics are:

- ❖ [Port Logic Register Map](#)
- ❖ [Port Logic Registers: General](#)
- ❖ [Port Logic Registers: Flow Control Credit Status](#)
- ❖ [Port Logic Registers: Transmit Arbitration](#)
- ❖ [Port Logic Registers: Receive Queue Control](#)
- ❖ [Port Logic Registers: Segmented Buffer Depth](#)
- ❖ [Port Logic Registers: Gen2](#)
- ❖ [Port Logic Registers: PHY Status and Control Registers](#)

4.3.1 Port Logic Register Map

Table 4-286 shows the register map for the Port Logic registers. Refer to the indicated pages for detailed register descriptions.

Table 4-286 Port Logic Register Map

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
0x700	Ack Latency Timer and Replay Timer Register				508
+0x4	Other Message Register				509
+0x8	Port Force Link Register				510
+0xC	Ack Frequency Register				511
+0x10	Port Link Control Register				513
+0x14	Lane Skew Register				515
+0x18	Symbol Number Register				516
+0x1C	Symbol Timer Register and Filter Mask Register 1				517
+0x20	Filter Mask Register 2				520
+0x24	PL_reg36. Reserved for internal feature.				
+0x28	Debug Register 0				521
+0x2C	Debug Register 1				521
+0x30	Transmit Posted FC Credit Status Register				522
+0x34	Transmit Non-Posted FC Credit Status Register				522
+0x38	Transmit Completion FC Credit Status Register				523
+0x3C	Queue Status Register				523
+0x40	VC Transmit Arbitration Register 1				524
+0x44	VC Transmit Arbitration Register 2				524
+0x48	VC0 Posted Receive Queue Control				525
+0x4C	VC0 Non-Posted Receive Queue Control				527
+0x50	VC0 Completion Receive Queue Control				528
+0x54	VC1 Posted Receive Queue Control				529
+0x58	VC1 Non-Posted Receive Queue Control				530
+0x5C	VC1 Completion Receive Queue Control				531
+0x60	VC2 Posted Receive Queue Control				532
+0x64	VC2 Non-Posted Receive Queue Control				533

Table 4-286 Port Logic Register Map (Continued)

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0x68	VC2 Completion Receive Queue Control				534
+0x6C	VC3 Posted Receive Queue Control				535
+0x70	VC3 Non-Posted Receive Queue Control				536
+0x74	VC3 Completion Receive Queue Control				537
+0x78	VC4 Posted Receive Queue Control				538
+0x7C	VC4 Non-Posted Receive Queue Control				539
+0x80	VC4 Completion Receive Queue Control				540
+0x84	VC5 Posted Receive Queue Control				541
+0x88	VC5 Non-Posted Receive Queue Control				542
+0x8C	VC5 Completion Receive Queue Control				543
+0x90	VC6 Posted Receive Queue Control				544
+0x94	VC6 Non-Posted Receive Queue Control				545
+0x98	VC6 Completion Receive Queue Control				546
+0x9C	VC7 Posted Receive Queue Control				547
+0xA0	VC7 Non-Posted Receive Queue Control				548
+0xA4	VC7 Completion Receive Queue Control				549
+0xA8	VC0 Posted Buffer Depth				550
+0xAC	VC0 Non-Posted Buffer Depth				551
+0xB0	VC0 Completion Buffer Depth				552
+0xB4	VC1 Posted Buffer Depth				553
+0xB8	VC1 Non-Posted Buffer Depth				554
+0xBC	VC1 Completion Buffer Depth				555
+0xC0	VC2 Posted Buffer Depth				556
+0xC4	VC2 Non-Posted Buffer Depth				557
+0xC8	VC2 Completion Buffer Depth				558
+0xCC	VC3 Posted Buffer Depth				559
+0xD0	VC3 Non-Posted Buffer Depth				560
+0xD4	VC3 Completion Buffer Depth				561
+0xD8	VC4 Posted Buffer Depth				562

**Table 4-286 Port Logic Register Map (Continued)**

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0	Page
+0xDC	VC4 Non-Posted Buffer Depth				563
+0xE0	VC4 Completion Buffer Depth				564
+0xE4	VC5 Posted Buffer Depth				565
+0xE8	VC5 Non-Posted Buffer Depth				566
+0xEC	VC5 Completion Buffer Depth				567
+0xF0	VC6 Posted Buffer Depth				568
+0xF4	VC6 Non-Posted Buffer Depth				569
+0xF8	VC6 Completion Buffer Depth				570
+0xFC	VC7 Posted Buffer Depth				571
+0x100	VC7 Non-Posted Buffer Depth				572
+0x104	VC7 Completion Buffer Depth				573
+0x10C	Gen2				574
+0x110	PHY Status				575
+0x114	PHY Control				576
+0x118	Remote_rd_req_size Control. Reserved for internal feature.				
+0x11C	Sbc_init. Reserved for internal feature.				



4.3.2 Port Logic Registers: General

4.3.2.1 Ack Latency Timer and Replay Timer Register

Offset: 0x700

Table 4-287 Ack Latency Timer and Replay Timer Register

Bits	Default	Attr	Description
15:0	4143 / `CX_NB	RWS	<p>Round Trip Latency Time Limit</p> <p>The Ack/Nak latency timer expires when it reaches this limit.</p> <p>The default value depends on number of bytes (NB) per cycle, which depends on which version of the core you are using (1 = 250 MHz, 2 = 125 MHz).</p> <p>The default is then updated based on the Negotiated Link Width and Max_Payload_Size.</p> <p>Note: If operating at 5 Gb/s, then an additional 51/CX_NB is added. This is for additional internal processing for received TLPs and transmitted DLLPs.</p>
31:16	12429 / `CX_NB	RWS	<p>Replay Time Limit</p> <p>The replay timer expires when it reaches this limit. The core initiates a replay upon reception of a Nak or when the replay timer expires.</p> <p>The default value depends on number of bytes (NB) per cycle, which depends on which version of the core you are using (1 = 250 MHz, 2 = 125 MHz).</p> <p>The default is then updated based on the Negotiated Link Width and Max_Payload_Size.</p> <p>Note: If operating at 5 Gb/s, then an additional 153/CX_NB is added. This is for additional internal processing for received TLPs and transmitted DLLPs.</p>

4.3.2.2 Other Message Register

Offset: 0x700 + 0x4

Table 4-288 Other Message Register

Bits	Default	Attr	Description
31:0	`DEFAULT_REQ_OTHER_MSG	RW	<p>Other Message Register</p> <p>This register can be used for either of the following purposes:</p> <ul style="list-style-type: none"> • To send a specific PCI Express Message, the application writes the payload of the Message into this register, then sets bit 0 of the Port Link Control Register to send the Message. • To store a corruption pattern for corrupting the LCRC on all TLPs, the application places a 32-bit corruption pattern into this register and enables this function by setting bit 25 of the Port Link Control Register. When enabled, the transmit LCRC result is XOR'd with this pattern before inserting it into the packet.

4.3.2.3 Port Force Link Register

Offset: 0x700 + 0x8

Table 4-289 Port Force Link Register

Bits	Default	Attr	Description
7:0	`DEFAULT_LINK_NUM	RWS	Link Number Not used for Endpoint
14:8	0x00	RsvdP	Reserved
15	0	RW*	Force Link Forces the Link to the state specified by the Link State field. The Force Link pulse will trigger Link re-negotiation. * As the Force Link is a pulse, writing a 1 to it does trigger the forced link state event, even though reading it always returns a 0.
21:16	0x00	RWS	Link State The Link state that the core will be forced to when bit 15 (Force Link) is set. State encoding is defined in xmlh_ltssm.v.
23:22	0x0	RsvdP	Reserved
31:24	0x7	RWS	Low Power Entrance Count The Power Management state will wait for this many clock cycles for the associated completion of a CfgWr to D-state register to go low-power. This register is intended for applications that do not let the core handle a completion for configuration request to the PMCSR register. Note: Only used in the DM core (in EP mode), EP core, and the upstream port of a Switch.

4.3.2.4 Ack Frequency Register

Offset: 0x700 + 0xC

Table 4-290 Ack Frequency Register

Bits	Default	Attr	Description
7:0	`DEFAULT_ACK_FREQUENCY	RWS	<p>Ack Frequency</p> <p>The core accumulates the number of pending Ack's specified here (up to 255) before sending an Ack.</p>
15:8	`CX_NFTS	RWS	<p>N_FTS</p> <p>The number of Fast Training Sequence ordered sets to be transmitted when transitioning from L0s to L0. The maximum number of FTS ordered-sets that a component can request is 255.</p> <p>Note: The core does not support a value of zero; a value of zero can cause the LTSSM to go into the recovery state when exiting from L0s.</p>
23:16	`CX_COMM_NFTS	RWS	<p>N_FTS when common clock is used.</p> <p>The number of Fast Training Sequence ordered sets to be transmitted when transitioning from L0s to L0. The maximum number of FTS ordered-sets that a component can request is 255.</p> <p>This field is writable only if the parameters are selected as follows during configuration of the core; otherwise, it will be hard coded to CX_COMM_NFTS.</p> <ul style="list-style-type: none"> • CX_NFTS != CX_COMM_NFTS • DEFAULT_L0S_EXIT_LATENCY != DEFAULT_COMM_L0S_EXIT_LATENCY • DEFAULT_L1_EXIT_LATENCY != DEFAULT_COMM_L1_EXIT_LATENCY <p>Note: The core does not support a value of zero; a value of zero can cause the LTSSM to go into the recovery state when exiting from L0s.</p>

Table 4-290 Ack Frequency Register (Continued)

Bits	Default	Attr	Description
26:24	`DEFAULT_L0S_ENTR_LATENCY	RWS	L0s Entrance Latency Values correspond to: <ul style="list-style-type: none">• 000: 1 µs• 001: 2 µs• 010: 3 µs• 011: 4 µs• 100: 5 µs• 101: 6 µs• 110 or 111: 7 µs
29:27	`DEFAULT_L1_ENTR_LATENCY	RWS	L1 Entrance Latency Values correspond to: <ul style="list-style-type: none">• 000: 1 µs• 001: 2 µs• 010: 4 µs• 011: 8 µs• 100: 16 µs• 101: 32 µs• 110 or 111: 64 µs
30	0	RWS	Enter ASPM L1 without receive in L0s. Allow core to enter ASPM L1 even when link partner did not go to L0s (receive is not in L0s). When not set, core goes to ASPM L1 only after idle period during which both receive and transmit are in L0s.
31	0x0	RsvdP	Reserved



4.3.2.5 Port Link Control Register

Offset: 0x700 + 0x10

Table 4-291 Port Link Control Register

Bits	Default	Attr	Description
0	0	RWS	Other Message Request When software writes a ‘1’ to this bit, the core transmits the Message contained in the Other Message register.
1	0	RWS	Scramble Disable Turns off data scrambling.
2	0	RWS	Loopback Enable Turns on loopback.
3	0	RWS	Reset Assert Triggers a recovery and forces the LTSSM to the Hot Reset state (downstream port only).
4	0	RsvdP	Reserved
5	1	RWS	DLL Link Enable Enables Link initialization. If DLL Link Enable = 0, the core does not transmit InitFC DLLPs and does not establish a Link.
6	0	RsvdP	Reserved
7	`DEFAULT_FAST_LINK_ENABLE	RWS	Fast Link Mode Sets all internal timers to fast mode for simulation purposes.
11:8	0x1	RsvdP	Reserved
15:12	0x0	RsvdP	Reserved
21:16	Derived from `CX_NL	RWS	Link Mode Enable <ul style="list-style-type: none"> • 000001: x1 • 000011: x2 • 000111: x4 • 001111: x8 • 011111: x16 • 111111: x32 (not supported) The default value is the number of Lanes supported in the version of the core you are using.
25:22	0x0	RWS	Reserved



Table 4-291 Port Link Control Register (Continued)

Bits	Default	Attr	Description
31:26	0x0	RsvdP	Reserved

4.3.2.6 Lane Skew Register

Offset: 0x700 + 0x14

Table 4-292 Lane Skew Register

Bits	Default	Attr	Description
23:0	0x000000	RWS	Insert Lane Skew for Transmit (not supported for x16) Optional feature that causes the core to insert skew between Lanes for test purposes. There are three bits per Lane. The value is in units of one symbol time. For example, the value 010b for a Lane forces a skew of two symbol times for that Lane. The maximum skew value for any Lane is 5 symbol times.
24	0	RWS	Flow Control Disable Prevents the core from sending FC DLLPs.
25	0	RWS	Ack/Nak Disable Prevents the core from sending Ack and Nak DLLPs.
30:26	0x0	RsvdP	Reserved
31	0	RWS	Disable Lane-to-Lane Deskew Causes the core to disable the internal Lane-to-Lane deskew logic.

4.3.2.7 Symbol Number Register

Offset: 0x700 + 0x18

Table 4-293 Symbol Number Register

Bits	Default	Attr	Description
3:0	0xA	RWS	Number of TS Symbols Sets the number of TS identifier symbols that are sent in TS1 and TS2 ordered sets.
7:4	0x0	RsvdP	Reserved
10:8	0x3	RWS	Number of SKP Symbols
13:11	0x0	RsvdP	Reserved
18:14	`DEFAULT_ REPLAY_ADJ	RWS	Timer Modifier for Replay Timer Increases the timer value for the replay timer, in increments of 64 clock cycles.
23:19	0x0	RWS	Timer Modifier for Ack/Nak Latency Timer Increases the timer value for the Ack/Nak latency timer, in increments of 64 clock cycles.
28:24	0x0	RWS	Timer Modifier for Flow Control Watchdog Timer Increases the timer value for the Flow Control watchdog timer, in increments of 16 clock cycles.
31:29	CX_NFUNC	RWS	Configuration Requests targeted at function numbers above this value will be returned with UR (unsupported request).

4.3.2.8 Symbol Timer Register and Filter Mask Register 1

Offset: 0x700 + 0x1C

Table 4-294 Symbol Timer Register and Filter Mask 1

Bits	Default	Attr	Description
10:0	1280 / `CX_NB	RWS	<p>SKP Interval Value</p> <p>The number of symbol times to wait between transmitting SKP ordered sets. Note that the core actually waits the number of symbol times in this register plus 1 between transmitting SKP ordered sets. The application must program this register accordingly. For example, if 1536 we're programmed into this register (in a 250MHz core), then the core will actually transmit Skp ordered sets once every 1537 symbol times.</p> <p>Also, the value programmed to this register is actually clock ticks and not symbol times. In a 125MHz core, programming the value programmed to this register should be scaled down by a factor of 2 (since 1 clock tick=2 symbol times in this case).</p>
14:11	0x0	RsvdP	Reserved
15	0x0	RWS	Disable FC Watchdog Timer

Table 4-294 Symbol Timer Register and Filter Mask 1 (Continued)

Bits	Default	Attr	Description
31:16	`DEFAULT_FILTER_MASK_1 = 0x0	RWS	<p>Mask RADM Filtering and Error Handling Rules</p> <p>There are several mask bits to turn off the filtering and error handling rules (Refer to “Receive Filtering” on page 60).</p> <p>In each case, 0 applies the associated filtering rule and 1 masks the associated filtering rule. A more detailed description for these bits is provided in Table 4-295.</p> <ul style="list-style-type: none"> • [31]: Mask filtering of received Configuration Requests (RC mode only) • [30]: Mask filtering of received I/O Requests (RC mode only) • [29]: Send Message TLPs to the application on RTRGT1 and send decoded Message on the SII (1) or send decoded Message on the SII, then drop the Message TLPs (0). The default value for this bit is the inverse of `FLT_DROP_MSG. That is, if `FLT_DROP_MSG = 1, then the default value of this bit is 0 (drop Message TLPs). Note that this bit only controls message TLPs <i>other than</i> Vendor MSGs. Vendor MSGs are controlled by Filter Mask Register 2, bits [1:0]. • [28]: Mask ECRC error filtering for Completions • [27]: Mask ECRC error filtering • [26]: Mask Length mismatch error for received Completions • [25]: Mask Attributes mismatch error for received Completions • [24]: Mask Traffic Class mismatch error for received Completions • [23]: Mask function mismatch error for received Completions • [23]: Mask Requester ID mismatch error for received Completions • [21]: Mask Tag error rules for received Completions • [20]: Mask Locked Request filtering • [19]: Mask Type 1 Configuration Request filtering • [18]: Mask BAR match filtering • [17]: Mask poisoned TLP filtering • [16]: Mask function mismatch filtering for incoming Requests

Table 4-295 Filter Mask 1: Mask RADM Filtering and Error Handling Rules

Bits	Name	Description
31	`CX_FLT_MASK_RC_CFG_DISCARD	<ul style="list-style-type: none"> • 0: For RADM RC filter to not allow CFG transaction being received • 1: For RADM RC filter to allow CFG transaction being received

Table 4-295 Filter Mask 1: Mask RADM Filtering and Error Handling Rules (Continued)

Bits	Name	Description
30	`CX_FLT_MASK_RC_IO_DISCARD	<ul style="list-style-type: none"> • 0: For RADM RC filter to not allow IO transaction being received • 1: For RADM RC filter to allow IO transaction being received
29	`CX_FLT_MASK_MSG_DROP	<ul style="list-style-type: none"> • 0: Drop MSG TLP (except for Vendor MSG) • 1 - Do not Drop MSG (except for Vendor MSG)
28	`CX_FLT_MASK_CPL_ECRC_DISCARD	<ul style="list-style-type: none"> • 0: Discard TLPs with ECRC errors for CPL type • 1: Allow TLPs with ECRC errors to be passed up for CPL type
27	`CX_FLT_MASK_ECRC_DISCARD	<ul style="list-style-type: none"> • 0: Discard TLPs with ECRC errors • 1: Allow TLPs with ECRC errors to be passed up
26	`CX_FLT_MASK_CPL_LEN_MATCH	<ul style="list-style-type: none"> • 0: Enforce length match for rcvd CPL TLPs; infraction results in cpl_abort, and possibly AER of unexp_cpl_err • 1: MASK length match for rcvd CPL TLPs
25	`CX_FLT_MASK_CPL_ATTR_MATCH	<ul style="list-style-type: none"> • 0: Enforce attribute match for rcvd CPL TLPs; infraction results in cpl_abort, and possibly AER of unexp_cpl_err, cpl_rcvd_ur, cpl_rcvd_ca • 1: Mask attribute match for rcvd CPL TLPs
24	`CX_FLT_MASK_CPL_TC_MATCH	<ul style="list-style-type: none"> • 0: Enforce Traffic Class match for rcvd CPL TLPs; infraction results in cpl_abort, and possibly AER of unexp_cpl_err, cpl_rcvd_ur, cpl_rcvd_ca • 1: Mask Traffic Class match for rcvd CPL TLPs
23	`CX_FLT_MASK_CPL_FUNC_MATCH	<ul style="list-style-type: none"> • 0: Enforce function match for rcvd CPL TLPs; infraction results in cpl_abort, and possibly AER of unexp_cpl_err, cpl_rcvd_ur, cpl_rcvd_ca • 1: Mask function match for rcvd CPL TLPs
22	`CX_FLT_MASK_CPL_REQID_MATCH	<ul style="list-style-type: none"> • 0: Enforce Req. Id match for rcvd CPL TLPs; infraction result in cpl_abort, and possibly AER of unexp_cpl_err, cpl_rcvd_ur, cpl_rcvd_ca • 1: Mask Req. Id match for rcvd CPL TLPs
21	`CX_FLT_MASK_CPL_TAGERR_MATCH	<ul style="list-style-type: none"> • 0: Enforce Tag Error Rules for rcvd CPL TLPs; infraction result in cpl_abort, and possibly AER of unexp_cpl_err, cpl_rcvd_ur, cpl_rcvd_ca • 1: Mask Tag Error Rules for rcvd CPL TLPs
20	`CX_FLT_MASK_LOCKED_RD_AS_U R	<ul style="list-style-type: none"> • 0: Treat locked Read TLPs as UR for EP; Supported for RC • 1: Treat locked Read TLPs as Supported for EP; UR for RC

Table 4-295 Filter Mask 1: Mask RADM Filtering and Error Handling Rules (Continued)

Bits	Name	Description
19	`CX_FLT_MASK_CFG_TYPE1_REQUEST_AS_UR	<ul style="list-style-type: none"> • 0: Treat CFG type1 TLPs as UR for EP; Supported for RC • 1: Treat CFG type1 TLPs as Supported for EP; UR for RC
18	`CX_FLT_MASK_UR_OUTSIDE_BAR	<ul style="list-style-type: none"> • 0: Treat out-of-bar TLPs as UR • 1: Treat out-of-bar TLPs as Supported Requests
17	`CX_FLT_MASK_UR_POIS	<ul style="list-style-type: none"> • 0: Treat poisoned TLPs as UR • 1: Treat poisoned TLPs as Supported Requests
16	`CX_FLT_MASK_UR_FUNC_MISMATCH	<ul style="list-style-type: none"> • 0: Treat Function MisMatched TLPs as UR • 1: Treat Function MisMatched TLPs as Supported

4.3.2.9 Filter Mask Register 2

Offset: 0x700 + 0x20

Table 4-296 Filter Mask 2

Bits	Default	Attr	Description
31:0	`DEFAULT_FILTERER_MASK_2 = 0x0	RWS	<p>Mask RADM Filtering and Error Handling Rules</p> <p>There are several mask bits used to turn off the filtering and error handling rules (refer to “Receive Filtering” on page 60).</p> <ul style="list-style-type: none"> • [31:4]: Reserved • [3]: `define CX_FLT_MASK_HANDLE_FLUSH <ul style="list-style-type: none"> - 0: Disable Core Filter to handle flush request - 1: Enable Core Filter to handle flush request • [2]: `define CX_FLT_MASK_DABORT_4UCPL <ul style="list-style-type: none"> - 0: Enable DLLP abort for unexpected CPL - 1: Do not enable DLLP abort for unexpected CPL • [1]: Mask Vendor MSG Type 1 dropped silently • [0]: Mask Vendor MSG Type 0 dropped with UR error reporting. <p>In each case for Mask Vendor MSG Type, 0 applies the associated filtering rule and 1 masks the associated filtering rule. 0 is the default.</p> <p>For example, if bit 0 (controlling Vendor MSG type 0), is set to 0 (default), Vendor MSG0 will be dropped and treated as UR; if set to 1, it will be passed to the TRGT1 interface.</p> <p>If bit 1 (controlling Vendor MSG Type 1) is set to 0 (default), Vendor MSG1 will be silently dropped; if set to 1, it will be passed to the TRGT1 interface.</p>



4.3.2.10 Debug Register 0

Offset: 0x700 + 0x28

Table 4-297 Debug Register 0

Bits	Default	Attr	Description
31:0	0x0	RO	The value on cxpl_debug_info[31:0].

4.3.2.11 Debug Register 1

Offset: 0x700 + 0x2C

Table 4-298 Debug Register 1

Bits	Default	Attr	Description
31:0	0x0	RO	The value on cxpl_debug_info[63:32].



4.3.3 Port Logic Registers: Flow Control Credit Status

4.3.3.1 Transmit Posted FC Credit Status

Offset: 0x700 + 0x30

Table 4-299 Transmit Posted FC Credit Status

Bits	Default	Attr	Description
11:0	*	ROS	Transmit Posted Data FC Credits The Posted Data credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
19:12	*	ROS	Transmit Posted Header FC Credits The Posted Header credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
31:20	0x000	RsvdP	Reserved

*Default value depends on the number of advertised credits for header and data {12'b0, xtlh_xadm_ph_cdts, xtlh_xadm_pd_cdts}; If the number of advertised completion credits (both header and data) are infinite, then the default would be {12'b0, 8'hFF, 12'hFFF}.

4.3.3.2 Transmit Non-Posted FC Credit Status

Offset: 0x700 + 0x34

Table 4-300 Transmit Non-Posted FC Credit Status

Bits	Default	Attr	Description
11:0	*	ROS	Transmit Non-Posted Data FC Credits The Non-Posted Data credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
19:12	*	ROS	Transmit Non-Posted Header FC Credits The Non-Posted Header credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
31:20	0x000	RsvdP	Reserved

*Default value depends on the number of advertised credits for header and data {12'b0, xtlh_xadm_nph_cdts, xtlh_xadm_npd_cdts}; If the number of advertised completion credits (both header and data) are infinite, then the default would be {12'b0, 8'hFF, 12'hFFF}.

4.3.3.3 Transmit Completion FC Credit Status

Offset: 0x700 + 0x38

Table 4-301 Transmit Completion FC Credit Status

Bits	Default	Attr	Description
11:0	*	ROS	Transmit Completion Data FC Credits The Completion Data credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
19:12	*	ROS	Transmit Completion Header FC Credits The Completion Header credits advertised by the receiver at the other end of the Link, updated with each UpdateFC DLLP.
31:20	0x000	RsvdP	Reserved

*Default value depends on the number of advertised credits for header and data {12'b0, xtlh_xadm_cplh_cdts, xtlh_xadm_cpld_cdts}; If the number of advertised completion credits (both header and data) are infinite, then the default would be {12'b0, 8'hFF, 12'hFFF}.

4.3.3.4 Queue Status

Offset: 0x700 + 0x3C

Table 4-302 Queue Status

Bits	Default	Attr	Description
0	0	ROS	Received TLP FC Credits Not Returned Indicates that the core has sent a TLP but has not yet received an UpdateFC DLLP indicating that the credits for that TLP have been restored by the receiver at the other end of the Link. Note: This bit is for simulation only and will always be synthesized as 0.
1	0	ROS	Transmit Retry Buffer Not Empty Indicates that there is data in the transmit retry buffer.
2	0	ROS	Received Queue Not Empty Indicates there is data in one or more of the receive buffers.
31:3	0x0000000	RsvdP	Reserved

4.3.4 Port Logic Registers: Transmit Arbitration

4.3.4.1 VC Transmit Arbitration Register 1

Offset: 0x700 + 0x40

VC Transmit Arbitration Registers 1 and 2 specify the weights assigned to VC0–VC7 to be used for WRR transmit arbitration for VCs in the LPVC group. The following rules and restrictions apply regarding the values programmed in VC Transmit Arbitration Registers 1 and 2:

- ❖ There are 8 bits allocated for each weight value.
- ❖ No weight value for a VC in the LPVC group can be less than 1.
- ❖ No weight value can be greater than the number of phases in the selected arbitration scheme.
- ❖ The sum of the weights assigned to all VCs in the LPVC group must equal the number of phases in the selected arbitration scheme. For example, if 64-phase WRR arbitration is selected, the total of all WRR Weight values for all VCs in the LPVC group must equal 64.

Each of the VC numbers listed in [Table 4-303](#) and [Table 4-304](#) is a VC ID, not the VC structure number.

Read/write access to VC Transmit Arbitration Registers 1 and 2 depends on the value of `CX_LPVC_WRR_WEIGHT_WRITABLE:

- ❖ If `CX_LPVC_WRR_WEIGHT_WRITABLE is defined, VC Transmit Arbitration Registers 1 and 2 are RWS.
- ❖ If `CX_LPVC_WRR_WEIGHT_WRITABLE is not defined, VC Transmit Arbitration Registers 1 and 2 are hardwired to the default values set by the configuration parameters listed in the Default columns of [Table 4-303](#) and [Table 4-304](#).

Table 4-303 VC Transmit Arbitration Register 1

Bits	Default	Attr	Description
7:0	`LPVC_WRR_WEIGHT_VC0	See register description above.	WRR Weight for VC0
15:8	`LPVC_WRR_WEIGHT_VC1		WRR Weight for VC1
23:16	`LPVC_WRR_WEIGHT_VC2		WRR Weight for VC2
31:24	`LPVC_WRR_WEIGHT_VC3		WRR Weight for VC3

4.3.4.2 VC Transmit Arbitration Register 2

Offset: 0x700 + 0x44

Table 4-304 VC Transmit Arbitration Register 2

Bits	Default	Attr	Description
7:0	`LPVC_WRR_WEIGHT_VC4	See register description above.	WRR Weight for VC4
15:8	`LPVC_WRR_WEIGHT_VC5		WRR Weight for VC5
23:16	`LPVC_WRR_WEIGHT_VC6		WRR Weight for VC6
31:24	`LPVC_WRR_WEIGHT_VC7		WRR Weight for VC7

4.3.5 Port Logic Registers: Receive Queue Control

4.3.5.1 VC0 Posted Receive Queue Control



Note The data and header credits fields of the Receive Queue Control registers are used in all receive buffer configurations.

All other fields of the Receive Queue Control and Depth registers are used only in the segmented-buffer configuration.

Offset: 0x700 + 0x48

Table 4-305 VC0 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC0	ROS	<p>VC0 Posted Data Credits The number of initial Posted data credits for VC0, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC0	ROS	<p>VC0 Posted Header Credits The number of initial Posted header credits for VC0, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC0	ROS	<p>VC0 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC0, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time: <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward </p>
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC0	ROS	<p>TLP Type Ordering for VC0 Determines the TLP type ordering rule for VC0 receive queues, used only in the segmented-buffer configuration, writable through the DBI: <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express 2.0 specification</i>. • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted </p>

Table 4-305 VC0 Posted Receive Queue Control (Continued)

Bits	Default	Attr	Description
31	`CX_RADM_STRICT_VC _PRIORITY	ROS	<p>VC Ordering for Receive Queues</p> <p>Determines the VC ordering rule for the receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Strict ordering, higher numbered VCs have higher priority • 0: Round robin

4.3.5.2 VC0 Non-Posted Receive Queue Control

Offset: 0x700 + 0x4C

Table 4-306 VC0 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC0	ROS	<p>VC0 Non-Posted Data Credits The number of initial Non-Posted data credits for VC0, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC0	ROS	<p>VC0 Non-Posted Header Credits The number of initial Non-Posted header credits for VC0, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC0	ROS	<p>VC0 Non-Posted TLP Queue Mode The operating mode of the Non-Posted receive queue for VC0, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.3 VC0 Completion Receive Queue Control

Offset: 0x700 + 0x50

Table 4-307 VC0 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC0	ROS	<p>VC0 Completion Data Credits</p> <p>The number of initial Completion data credits for VC0, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC0	ROS	<p>VC0 Completion Header Credits</p> <p>The number of initial Completion header credits for VC0, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC0	ROS	<p>VC0 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC0, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.4 VC1 Posted Receive Queue Control

Offset: 0x700 + 0x54

Table 4-308 VC1 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC1	ROS	<p>VC1 Posted Data Credits The number of initial Posted data credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC1	ROS	<p>VC1 Posted Header Credits The number of initial Posted header credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC1	ROS	<p>VC1 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC1, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC1	ROS	<p>TLP Type Ordering for VC1 Determines the TLP type ordering rule for VC1 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.5 VC1 Non-Posted Receive Queue Control

Offset: 0x700 + 0x58

Table 4-309 VC1 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC1	ROS	<p>VC1 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC1	ROS	<p>VC1 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC1	ROS	<p>VC1 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC1, used only in the segmented-buffer configuration, writable through the DBI.</p> <p>Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.6 VC1 Completion Receive Queue Control

Offset: 0x700 + 0x5C

Table 4-310 VC1 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC1	ROS	<p>VC1 Completion Data Credits</p> <p>The number of initial Completion data credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC1	ROS	<p>VC1 Completion Header Credits</p> <p>The number of initial Completion header credits for VC1, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC1	ROS	<p>VC1 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC1, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.7 VC2 Posted Receive Queue Control

Offset: 0x700 + 0x60

Table 4-311 VC2 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC2	ROS	<p>VC2 Posted Data Credits The number of initial Posted data credits for VC2, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC2	ROS	<p>VC2 Posted Header Credits The number of initial Posted header credits for VC2, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC2	ROS	<p>VC2 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC2, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING _RULES_VC2	ROS	<p>TLP Type Ordering for VC2 Determines the TLP type ordering rule for VC2 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.8 VC2 Non-Posted Receive Queue Control

Offset: 0x700 + 0x64

Table 4-312 VC2 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC2	ROS	<p>VC2 Non-Posted Data Credits The number of initial Non-Posted data credits for VC2, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC2	ROS	<p>VC2 Non-Posted Header Credits The number of initial Non-Posted header credits for VC2, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC2	ROS	<p>VC2 Non-Posted TLP Queue Mode The operating mode of the Non-Posted receive queue for VC2, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time: <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward </p>
31:24	0x00	RsvdP	Reserved

4.3.5.9 VC2 Completion Receive Queue Control

Offset: 0x700 + 0x68

Table 4-313 VC2 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC2	ROS	<p>VC2 Completion Data Credits</p> <p>The number of initial Completion data credits for VC2, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC2	ROS	<p>VC2 Completion Header Credits</p> <p>The number of initial Completion header credits for VC2, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC2	ROS	<p>VC2 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC2, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.10 VC3 Posted Receive Queue Control

Offset: 0x700 + 0x6C

Table 4-314 VC3 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC3	ROS	<p>VC3 Posted Data Credits The number of initial Posted data credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC3	ROS	<p>VC3 Posted Header Credits The number of initial Posted header credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC3	ROS	<p>VC3 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC3, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC3	ROS	<p>TLP Type Ordering for VC3 Determines the TLP type ordering rule for VC3 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.11 VC3 Non-Posted Receive Queue Control

Offset: 0x700 + 0x70

Table 4-315 VC3 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC3	ROS	<p>VC3 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC3	ROS	<p>VC3 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC3	ROS	<p>VC3 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC3, used only in the segmented-buffer configuration, writable through the DBI.</p> <p>Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.12 VC3 Completion Receive Queue Control

Offset: 0x700 + 0x74

Table 4-316 VC3 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC3	ROS	<p>VC3 Completion Data Credits</p> <p>The number of initial Completion data credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC3	ROS	<p>VC3 Completion Header Credits</p> <p>The number of initial Completion header credits for VC3, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC3	ROS	<p>VC3 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC3, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.13 VC4 Posted Receive Queue Control

Offset: 0x700 + 0x78

Table 4-317 VC4 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC4	ROS	<p>VC4 Posted Data Credits The number of initial Posted data credits for VC4, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC4	ROS	<p>VC4 Posted Header Credits The number of initial Posted header credits for VC4, used for all receive queue buffer configurations. This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC4	ROS	<p>VC4 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC4, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time: <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward </p>
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC4	ROS	<p>TLP Type Ordering for VC4 Determines the TLP type ordering rule for VC4 receive queues, used only in the segmented-buffer configuration, writable through the DBI: <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted </p>
31	0x0	RsvdP	Reserved

4.3.5.14 VC4 Non-Posted Receive Queue Control

Offset: 0x700 + 0x7C

Table 4-318 VC4 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC4	ROS	<p>VC4 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC4, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC4	ROS	<p>VC4 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC4, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC4	ROS	<p>VC4 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC4, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.15 VC4 Completion Receive Queue Control

Offset: 0x700 + 0x80

Table 4-319 VC4 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC4	ROS	<p>VC4 Completion Data Credits The number of initial Completion data credits for VC4, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC4	ROS	<p>VC4 Completion Header Credits The number of initial Completion header credits for VC4, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC4	ROS	<p>VC4 Completion TLP Queue Mode The operating mode of the Completion receive queue for VC4, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.16 VC5 Posted Receive Queue Control

Offset: 0x700 + 0x84

Table 4-320 VC5 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC5	ROS	<p>VC5 Posted Data Credits The number of initial Posted data credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC5	ROS	<p>VC5 Posted Header Credits The number of initial Posted header credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC5	ROS	<p>VC5 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC5, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC5	ROS	<p>TLP Type Ordering for VC5 Determines the TLP type ordering rule for VC5 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.17 VC5 Non-Posted Receive Queue Control

Offset: 0x700 + 0x88

Table 4-321 VC5 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC5	ROS	<p>VC5 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC5	ROS	<p>VC5 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC5	ROS	<p>VC5 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC5, used only in the segmented-buffer configuration, writable through the DBI.</p> <p>Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.18 VC5 Completion Receive Queue Control

Offset: 0x700 + 0x8C

Table 4-322 VC5 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC5	ROS	<p>VC5 Completion Data Credits</p> <p>The number of initial Completion data credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC5	ROS	<p>VC5 Completion Header Credits</p> <p>The number of initial Completion header credits for VC5, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC5	ROS	<p>VC5 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC5, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.19 VC6 Posted Receive Queue Control

Offset: 0x700 + 0x90

Table 4-323 VC6 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC6	ROS	<p>VC6 Posted Data Credits The number of initial Posted data credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC6	ROS	<p>VC6 Posted Header Credits The number of initial Posted header credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC6	ROS	<p>VC6 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC6, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING_RULES_VC6	ROS	<p>TLP Type Ordering for VC6 Determines the TLP type ordering rule for VC6 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.20 VC6 Non-Posted Receive Queue Control

Offset: 0x700 + 0x94

Table 4-324 VC6 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC6	ROS	<p>VC6 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC6	ROS	<p>VC6 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC6	ROS	<p>VC6 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC6, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.21 VC6 Completion Receive Queue Control

Offset: 0x700 + 0x98

Table 4-325 VC6 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC6	ROS	<p>VC6 Completion Data Credits</p> <p>The number of initial Completion data credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC6	ROS	<p>VC6 Completion Header Credits</p> <p>The number of initial Completion header credits for VC6, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC6	ROS	<p>VC6 Completion TLP Queue Mode</p> <p>The operating mode of the Completion receive queue for VC6, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.22 VC7 Posted Receive Queue Control

Offset: 0x700 + 0x9C

Table 4-326 VC7 Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_PQ_DCRD_VC7	ROS	<p>VC7 Posted Data Credits The number of initial Posted data credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_PQ_HCRD_VC7	ROS	<p>VC7 Posted Header Credits The number of initial Posted header credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_P_QMODE_VC7	ROS	<p>VC7 Posted TLP Queue Mode The operating mode of the Posted receive queue for VC7, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
29:24	0x00	RsvdP	Reserved
30	`CX_RADM_ORDERING _RULES_VC7	ROS	<p>TLP Type Ordering for VC7 Determines the TLP type ordering rule for VC7 receive queues, used only in the segmented-buffer configuration, writable through the DBI:</p> <ul style="list-style-type: none"> • 1: Ordering of received TLPs follows the rules in <i>PCI Express Base 2.0 specification</i> • 0: Strict ordering for received TLPs: Posted, then Completion, then Non-Posted
31	0x0	RsvdP	Reserved

4.3.5.23 VC7 Non-Posted Receive Queue Control

Offset: 0x700 + 0xA0

Table 4-327 VC7 Non-Posted Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_NPQ_DCRD_VC7	ROS	<p>VC7 Non-Posted Data Credits</p> <p>The number of initial Non-Posted data credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_NPQ_HCRD_VC7	ROS	<p>VC7 Non-Posted Header Credits</p> <p>The number of initial Non-Posted header credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_NP_QMODE_VC7	ROS	<p>VC7 Non-Posted TLP Queue Mode</p> <p>The operating mode of the Non-Posted receive queue for VC7, used only in the segmented-buffer configuration, writable through the DBI.</p> <p>Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.5.24 VC7 Completion Receive Queue Control

Offset: 0x700 + 0xA4

Table 4-328 VC7 Completion Receive Queue Control

Bits	Default	Attr	Description
11:0	`RADM_CPLQ_DCRD_VC7	ROS	<p>VC7 Completion Data Credits The number of initial Completion data credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
19:12	`RADM_CPLQ_HCRD_VC7	ROS	<p>VC7 Completion Header Credits The number of initial Completion header credits for VC7, used for all receive queue buffer configurations.</p> <p>This field is writable through the DBI if `CX_DYNAMIC_FC_CREDIT is defined.</p>
20	0x0	RsvdP	Reserved
23:21	`RADM_CPL_QMODE_VC7	ROS	<p>VC7 Completion TLP Queue Mode The operating mode of the Completion receive queue for VC7, used only in the segmented-buffer configuration, writable through the DBI. Only one bit can be set at a time:</p> <ul style="list-style-type: none"> • Bit 23: Bypass • Bit 22: Cut-through • Bit 21: Store-and-forward
31:24	0x00	RsvdP	Reserved

4.3.6 Port Logic Registers: Segmented Buffer Depth

4.3.6.1 VC0 Posted Buffer Depth



- The Buffer Depth registers are used only in the segmented-buffer configuration.
- When writing to the Buffer Depth registers through the DBI, the new value must be less than or equal to the default value set during hardware configuration. (Writing through the DBI is possible only when `CX_DYNAMIC_SEG_SIZE is defined.

Offset: 0x700 + 0xA8

Table 4-329 VC0 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC0	ROS	VC0 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC0	ROS	VC0 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.2 VC0 Non-Posted Buffer Depth

Offset: 0x700 + 0xAC

Table 4-330 VC0 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC0	ROS	VC0 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC0	ROS	VC0 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.3 VC0 Completion Buffer Depth

Offset: 0x700 + 0xB0

Table 4-331 VC0 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC0	ROS	VC0 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC0	ROS	VC0 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC0 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.4 VC1 Posted Buffer Depth

Offset: 0x700 + 0xB4

Table 4-332 VC1 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC1	ROS	VC1 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC1	ROS	VC1 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.5 VC1 Non-Posted Buffer Depth

Offset: 0x700 + 0xB8

Table 4-333 VC1 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC1	ROS	VC1 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC1	ROS	VC1 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.6 VC1 Completion Buffer Depth

Offset: 0x700 + 0xBC

Table 4-334 VC1 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC1	ROS	VC1 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC1	ROS	VC1 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC1 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.7 VC2 Posted Buffer Depth

Offset: 0x700 + 0xC0

Table 4-335 VC2 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC2	ROS	VC2 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC2	ROS	VC2 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.8 VC2 Non-Posted Buffer Depth

Offset: 0x700 + 0xC4

Table 4-336 VC2 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC2	ROS	VC2 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC2	ROS	VC2 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.9 VC2 Completion Buffer Depth

Offset: 0x700 + 0xC8

Table 4-337 VC2 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC2	ROS	VC2 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC2	ROS	VC2 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC2 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.10 VC3 Posted Buffer Depth

Offset: 0x700 + 0xCC

Table 4-338 VC3 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC3	ROS	VC3 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC3	ROS	VC3 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.11 VC3 Non-Posted Buffer Depth

Offset: 0x700 + 0xD0

Table 4-339 VC3 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC3	ROS	VC3 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC3	ROS	VC3 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.12 VC3 Completion Buffer Depth

Offset: 0x700 + 0xD4

Table 4-340 VC3 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC3	ROS	VC3 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC3	ROS	VC3 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC3 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.13 VC4 Posted Buffer Depth

Offset: 0x700 + 0xD8

Table 4-341 VC4 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC4	ROS	VC4 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC4	ROS	VC4 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.14 VC4 Non-Posted Buffer Depth

Offset: 0x700 + 0xDC

Table 4-342 VC4 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC4	ROS	VC4 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC4	ROS	VC4 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.15 VC4 Completion Buffer Depth

Offset: 0x700 + 0xE0

Table 4-343 VC4 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC4	ROS	VC4 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC4	ROS	VC4 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC4 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.16 VC5 Posted Buffer Depth

Offset: 0x700 + 0xE4

Table 4-344 VC5 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC5	ROS	VC5 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC5	ROS	VC5 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.17 VC5 Non-Posted Buffer Depth

Offset: 0x700 + 0xE8

Table 4-345 VC5 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC5	ROS	VC5 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC5	ROS	VC5 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.18 VC5 Completion Buffer Depth

Offset: 0x700 + 0xEC

Table 4-346 VC5 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC5	ROS	VC5 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC5	ROS	VC5 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC5 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.19 VC6 Posted Buffer Depth

Offset: 0x700 + 0xF0

Table 4-347 VC6 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC6	ROS	VC6 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC6	ROS	VC6 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.20 VC6 Non-Posted Buffer Depth

Offset: 0x700 + 0xF4

Table 4-348 VC6 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC6	ROS	VC6 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC6	ROS	VC6 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.21 VC6 Completion Buffer Depth

Offset: 0x700 + 0xF8

Table 4-349 VC6 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC6	ROS	VC6 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC6	ROS	VC6 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC6 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.22 VC7 Posted Buffer Depth

Offset: 0x700 + 0xFC

Table 4-350 VC7 Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_PQ_DDP_VC7	ROS	VC7 Posted Data Queue Depth Sets the number of entries in the Posted data queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_PQ_HDP_VC7	ROS	VC7 Posted Header Queue Depth Sets the number of entries in the Posted header queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.23 VC7 Non-Posted Buffer Depth

Offset: 0x700 + 0x100

Table 4-351 VC7 Non-Posted Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_NPQ_DDP_VC7	ROS	VC7 Non-Posted Data Queue Depth Sets the number of entries in the Non-Posted data queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_NPQ_HDP_VC7	ROS	VC7 Non-Posted Header Queue Depth Sets the number of entries in the Non-Posted header queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.6.24 VC7 Completion Buffer Depth

Offset: 0x700 + 0x104

Table 4-352 VC7 Completion Buffer Depth Control

Bits	Default	Attr	Description
13:0	`RADM_CPLQ_DDP_VC7	ROS	VC7 Completion Data Queue Depth Sets the number of entries in the Completion data queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
15:14	0x0	RsvdP	Reserved
25:16	`RADM_CPLQ_HDP_VC7	ROS	VC7 Posted Header Queue Depth Sets the number of entries in the Completion header queue for VC7 when using the segmented-buffer configuration, writable through the DBI if `CX_DYNAMIC_SEG_SIZE is defined.
31:26	0x00	RsvdP	Reserved

4.3.7 Port Logic Registers: Gen2

A new register for gen2 has been added at offset 0x10C.

4.3.7.1 Port Logic Register (Gen2)

Offset: 0x10C

Bits	Default	Attribute	Description
31:21	0x0	Rsvd	Reserved
20	`SEL_DE_EM PHASIS	RW	Used to set the de-emphasis level for upstream ports.
19	0x0	RW	Config Tx Compliance Receive Bit When set to 1, signals LTSSM to transmit TS ordered sets with the compliance receive bit assert (equal to 1).
18	0x0	RW	Config PHY Tx Swing Indicates the voltage level the PHY should drive. When set to 1, indicates Full Swing. When set to 0, indicates Low Swing
17	0x0	RW	Directed Speed Change Indicates to the LTSSM whether or not to initiate a speed change.
16:8	0x0	RW	Lane Enable Indicates the number of lanes to check for exit from electrical idle in Polling.Active and Polling.Compliance. 1 = x1, 2 = x2, etc. Used to limit the maximum link width to ignore “broken” lanes that detect a receiver, but will not exit electrical idle and would otherwise prevent a valid link from being configured.
7:0	`DEFAULT_G EN2_N_FTS	RW	Sets the Number of Fast Training Sequences (N_FTS) that the core advertises as its N_FTS during GEN2 Link training. This value is used to inform the Link partner about the PHY's ability to recover synchronization after a low power state. The number should be provided by the PHY vendor. Note: Do not set N_FTS to zero; doing so can cause the LTSSM to go into the recovery state when exiting from L0s.

4.3.8 Port Logic Registers: PHY Status and Control Registers

4.3.8.1 PHY Status Register

Offset: 0x700 + 0x110

Table 4-353 PHY Status Register

Bits	Default	Attr	Description
31:0	0	ROS	PHY Status Data received directly from the phy_cfg_status bus.

4.3.8.2 PHY Control Register

Offset: 0x700 + 0x114

Table 4-354 PHY Control Register

Bits	Default	Attr	Description
31:0	0	RWS	PHY Control Data sent directly to the cfg_phy_control bus.

5

Parameters

The configuration parameters for the core are in two groups:

- ❖ Parameters that are available through the coreConsultant GUI
- ❖ Advanced parameters which, if necessary, you can modify by editing the source files directly

5.1 Parameters Available in the coreConsultant GUI

The configuration parameters in the coreConsultant GUI are grouped as follows:

- ❖ [General Configuration Parameters](#)
- ❖ [Transmit Configuration Parameters](#)
- ❖ [Queuing and Buffer Configuration Parameters](#)
- ❖ [VC Configuration Parameters](#)
- ❖ [Filter Configuration Parameters](#)
- ❖ [Common Register Configuration Parameters](#)
- ❖ [Function Configuration Parameters](#)
- ❖ [RAM Configuration Parameters](#)
- ❖ [PHY Configuration Parameters](#)
- ❖ [Design Pipelining Configuration Parameters](#)

The following sections describe the user-modifiable parameters in the above groups.

5.1.1 General Configuration Parameters

[TABLE 5-1](#) defines the general configuration parameters.

TABLE 5-1 General Configuration Parameters

coreConsultant Option Name	Description
PCIe Device Type	<p>Function: Controls the device type advertised in the PCIe Capabilities Register.</p> <p>Value Range: PCIE_EP, PCIE_EP_LEGACY</p> <p>Default Value: PCIE_EP</p> <p>Parameter Name: CX_DEVICE_TYPE</p> <p>Device Type: EP only</p>
Base Core Operating Frequency	<p>Function: The basic operating frequency of the core. If the core is configured for Gen2 Dynamic Frequency, the frequency will be doubled when operating at Gen2 speed. The speed of 62.5 MHz is applicable only to a Gen1 core.</p> <p>Value Range: 62.5 MHz, 125 MHz, 250 MHz</p> <p>Default Value: 250 MHz</p> <p>Parameter Name: CX_FREQ</p> <p>Device Type: All</p>
Maximum Number of Lanes Supported	<p>Function: Maximum number of Lanes the core will support.</p> <p>Value Range: 1 (x1), 2 (x2), 4 (x4), 8 (x8), or 16 (x16)</p> <p>Default Value: 4 (x4)</p> <p>Parameter Name: CX_NL</p> <p>Device Type: All</p>
Number of Virtual Channels	<p>Function: The number of Virtual Channels (VCs) to be supported. The core supports up to 8 VCs. Since VCs consume resources and are often not necessary, you can create a more efficient core by including only the required number of VCs.</p> <p>Value Range: 1–8</p> <p>Default Value: 1</p> <p>Parameter Name: CX_NVC</p> <p>Device Type: All</p>
Number of Functions	<p>Function: Number of functions to support. Only 1 function is supported if AXI/AHB is selected.</p> <p>Value Range: 1–8</p> <p>Default Value: 1</p> <p>Parameter Name: CX_NFUNC</p> <p>Device Type: DM / EP</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Maximum Payload Size Supported	<p>Function: The largest packet payload (Maximum Transfer Unit) that the device will support.</p> <p>This is distinct from the maximum operating payload (Max_Payload_Size) which may be set by software. This value is used to set memory sizes and places other restrictions on the size of structures. The device will support any valid payload size equal to or smaller than the value specified here.</p> <p>Value Range: 128, 256, 512, 1024, 2048, or 4096</p> <p>Default Value: 256</p> <p>Parameter Name: CX_MAX_MTU</p> <p>Device Type: All</p>
Maximum Tags Supported	<p>Function: Specifies the maximum Tag supported to manage received completions (shared by all functions in a device). This is used to size the Receive Completion look-up table and timeout RAM. The value that you choose in the GUI is the maximum number of tags to store received completions. The actual CX_MAX_TAG value that is propagated in the configured source code is the maximum receive Tag value. For example, specifying a value of 8 means `CX_MAX_TAG = 7; the application can specify receive completion Tag values 0–7.</p> <p>Value Range: 2, 4, 8, 16, 32, 64, 128, or 256</p> <p>Default Value: 32</p> <p>Parameter Name: CX_MAX_TAG</p> <p>Device Type: All</p>
Enable ECRC Support	<p>Function: Include or exclude the ECRC generation and checking hardware.</p> <p>This is distinct from the software control that may be used to enable or disable ECRC. When disabled, ECRC checking and insertion logic will not be included in the core in order to reduce gates. This is especially important for large architectures, in which these blocks can consume timing margin as well as area.</p> <p>When enabled, the core will include logic to check and insert ECRC.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_ECRC_ENABLE</p> <p>Device Type: All</p>
Enable ECRC Stripping	<p>Function: Include or exclude ECRC stripping hardware. If enabled, the core will strip ECRC from all incoming packets.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_ECRC_STRIP_ENABLE</p> <p>Device Type: All</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
FPGA	<p>Function: Select this option if you are implementing the core in an FPGA.</p> <p>Value Range: True (1) or False (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: FPGA</p> <p>Device Type: All</p>
Enable Lane Reversal Support	<p>Function: Include support for Lane reversal.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • True (1): Lane reversal is enabled • False (0): Lane reversal is disabled <p>Default Value: True (1)</p> <p>Parameter Name: CX_LANE_REVERSE</p> <p>Device Type: All</p>
Remove Port Logic Registers	<p>Function: Determines whether to exclude the Port Logic register block from your core configuration. Refer to Section 4.3, “PCIe Registers: Port Logic” on page 504 more information.</p> <ul style="list-style-type: none"> • True (1): Port Logic register block is removed. • False (0): Port Logic register block is not removed. <p>Default Value: False (0)</p> <p>Parameter Name: CX_PL_REG_DISABLE</p> <p>Device Type: All</p>
DBI Read Only Write Enable	<p>Function: Determines whether selected HwInit and RO register bits in the core configuration space are writable through the DBI.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • True (1): HwInit and RO register bits are writable through the DBI. • False (0): HwInit and RO register bits are not writable through the DBI. <p>Default Value: False (0)</p> <p>Parameter Name: CX_DBI_RO_WR_EN</p> <p>Device Type: All</p>
DBI Full Access with BAR and Function Number Supported	<p>Function: Provide support to allow DBI full access with BAR number and Function number.</p> <p>Value Range: Disabled (0) or Enabled (1)</p> <p>Default Value: Disabled (0)</p> <p>Parameter Name: DBI_MULTI_FUNC_BAR_EN</p> <p>Device Type: All</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Application Error Reporting	<p>Function: Determines whether to include input ports for application-detected error reporting.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: APP_RETURN_ERR_EN</p> <p>Device Type: All</p>
Mask Completion Timeout Errors	<p>Function: Mask detection of Completion timeout errors. If selected, the core will not automatically report Completion timeout errors. The application must check for Completion timeouts and report Completion timeout errors using the app_err_bus input signal.</p> <p>The Mask Completion Timeout Errors option is available when Application Error Reporting is selected.</p> <p>Value Range: True (1) or False (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CPL_TIMEOUT_ERR_MASK</p> <p>Device Type: All</p>
Provide control to flip physical RX/TX lanes	<p>Function: Include or exclude the tx_lane_flip_en and rx_lane_flip_en inputs, which control Lane reversal when operating in RC mode.</p> <p>Select this option to allow the application to initiate Lane reversal when automatic Lane reversal does not occur because Lane 0 is not detected.</p> <p>For example, in a situation where an x4 core is connected to an x8 device that has its Lanes reversed, the core does not detect Lane 0 of the x8 device (only Lanes 7 down to 4) and therefore, does not perform Lane reversal during Link initialization. In this case, the application can assert tx_lane_flip_en and rx_lane_flip_en to force Lane reversal.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CX_LANE_FLIP_CTRL_EN</p> <p>Device Type: DM / RC / SW (not EP)</p>
Disable Lane Deskew	<p>Function: Disable the deskewing mechanism.</p> <p>Disable the deskewing mechanism only if connecting to a multi-Lane PHY that implements Lane-to-Lane deskewing.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • True (1): Lane deskew is disabled • False (0): Lane deskew is enabled <p>Default Value: False (0)</p> <p>Parameter Name: CX_DESKEW_DISABLE</p> <p>Device Type: All</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Maximum skew in symbols	<p>Function: Specifies the maximum skew between lanes in symbols Values: 5 or 8 Default Value: 5 Parameter Name: CX_MAX_SKew_NUM Device Type: All</p>
Gen 2 Mode	<p>Function: Gen 2 operating mode. Value Range: Dynamic Frequency, Dynamic Width, Disabled Default Value: Disabled Parameter Name: CX_GEN2_MODE Device Type: All</p>
Cfg Directed Speed Change	<p>Function: If turned on, the core will initiate a gen2 speed change after the link is initialized. Value Range: 0 or 1 <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) Default Value: 0 Parameter Name: DEFAULT_GEN2_SPEED_CHANGE Device Type: All</p>
Function Level Reset Support	<p>Function: Enables core support for Function Level Reset (FLR). Value Range: 0 or 1 <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) Default Value: 'CX_SRIOV_ENABLE Parameter Name: CX_FLR_ENABLE Device Type: EP</p>
AMBA Enable	<p>Function: Enable AXI or AHB interface. Values: <ul style="list-style-type: none"> • None (0) • AHB (1) (not available in SW) • AXI (2) (not available in SW) Default Value: None (0) Parameter Name: AMBA_INTERFACE Device Type: DM/EP/RC (not SW)</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Multiple Devices/Buses per Function	<p>Function: Determines the size of the cfg_pbus_num and cfg_pbus_dev_num buses: one set of bits for all functions or separate bit-fields for each configured function.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Separate bit-fields for each function False (0): One bit-field for all functions <p>Default Value: False (0)</p> <p>Parameter Name: MULTI_DEVICE_AND_BUS_PER_FUNC_EN</p> <p>Device Type: DM / EP (not RC / SW)</p>
Enable Address Alignment	<p>Function: Enable address alignment for transmitted TLPs. If selected, the address alignment feature is available and is controlled by the client0_addr_align_en input for XALI0, client1_addr_align_en for XALI1, and client2_addr_align_en for XALI2.</p> <p>Value Range: True (1) or False (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: GLOB_ADDR_ALIGN_EN</p> <p>Device Type: All</p>
Peer to Peer Support	<p>Function: Support peer-to-peer transactions. This optional parameter may either be selected from the General Configuration page in the GUI or added directly to a configuration batch file with the following command:</p> <pre>set_configuration_parameter CX_P2P_ENABLE 1</pre> <p>Value Range: True (1) or false (0)</p> <p>Default Value: false (0)</p> <p>Parameter Name: CX_P2P_ENABLE</p> <p>Device Type: All</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Address Translation Supported	<p>Function: When enabled, the core supports address translation services. This address translation is for a proprietary local address translation implementation and is not related to the PCI-SIG ATS specification support. Address translation is used for mapping different address ranges to different memory spaces comprehended by the application. A typical example will map the AMBA memory space to PCI memory space when the application has the PCIe-to-AMBA bridge in place. Other examples could include switch applications or IOV applications. The core provides only a mechanism to allow the application to take control over address translation and the core will perform the actual swap of the address and other fields of a request. Refer to “Address Translation” for more details.</p> <p>Value Range: On, Off</p> <p>Default Value: Off</p> <p>Parameter Name: ADDR_TRANSLATION_SUPPORT_EN</p> <p>Device Type: All</p>
Enable Diagnostic Bus	<p>Function: Enables routing of diag_status_bus and diag_ctrl_bus signals to the SII interface.</p> <p>Values:</p> <ul style="list-style-type: none"> True (1): Routing is enabled False (0): Routing is disabled <p>Default Value: 0</p> <p>Parameter Name: DIAGNOSTIC_ENABLE</p> <p>Device Type: All</p>
Include Target Interface 1	<p>Function: Include or exclude the receive target 1 interface (RTRGT1) from the design.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): RTRGT1 is included False (0): RTRGT1 is not included <p>Default Value: True (1)</p> <p>Parameter Name: TRGT1_POPULATE</p> <p>Device Type: All</p> <p>In RC mode, this parameter is read only because RTRGT1 is required for RC mode.</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Support Vendor Messages	<p>Function: If vendor message support is not needed, a narrower address bus can be used. This saves gates and makes some rams significantly narrower.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Vendor Messages are Supported False (0): Vendor Messages are Not Supported <p>Default Value: False (0)</p> <p>Parameter Name: VENDOR_MESSAGE_SUPPORT</p> <p>Device Type: EP (not DM/RC/SW)</p> <p>NOTE: The control for the actual dropping of the Vendor Message is in Filter Mask Register 2.</p>
Optional Check Enable	<p>Function: When on, violations of the byte enable check (Section 2.2.5 of PCIe spec) and the address/length check (Section 2.2.7) will be treated as Malformed TLPs. Also, violations of the various Flow Control check will result in a Flow Control Protocol Error (FCPE). Flow control checks are described in Section 2.6.1 of the PCIe spec.</p> <p>Default Value: False (0)</p> <p>Parameter Name: ENABLE_OPTIONAL_CHECKS</p> <p>Device Type: All</p>
Enable ASPM L1 Timeout	<p>Function: Enable the ASPM L1 timer so that the core will automatically go to L1 when the timer expires and the conditions in the <i>PCI Express 2.0 specification</i> are met. If you disable the ASPM L1 timer, the application can request L1 entry by asserting app_req_entr_l1. The ASPM L1 timer should be larger than the L0s timer. Otherwise, ASPM L0s and L1 timeout may not function properly.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): L1 timer is enabled False (0): L1 timer is disabled <p>Default Value: True (1)</p> <p>Parameter Name: CX_ASPM_TIMEOUT_ENTR_L1_EN</p> <p>Device Type: All</p>
LBC Address Bus Width	<p>Function: Defines the width of the ELBI address bus for the Local Bus Controller (LBC).</p> <p>Value Range: 1–32</p> <p>Default Value: 32 (bits)</p> <p>Parameter Name: CX_LBC_EXT_AW</p> <p>Device Type: All</p>

TABLE 5-1 General Configuration Parameters (Continued)

coreConsultant Option Name	Description
Completion Timeout Ranges Enable	<p>Function: Provides support for Completion Timeout Ranges.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_CPL_TO_RANGES_ENABLE</p> <p>Device Type: EP</p>
Address Translation Services Support	<p>Function: When enabled, core supports address translation services. Provides partial support of PCI-SIG ATS specification and allows the application to use/set the Address Type (AT) field of the TLP header. Refer to "PCI-SIG Address Translation Services (ATS)".</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_ATS_ENABLE</p> <p>Device Type: EP</p>
RX Address Translation Services Support	<p>Function: When enabled, core supports RX address translation services. Provides partial support of PCI-SIG ATS specification and allows the application to use the Address Type (AT) field of the TLP header.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 'CX_ATS_ENABLE'</p> <p>Parameter Name: ATS_RX_ENABLE</p> <p>Device Type: EP</p>
TX Address Translation Services Support	<p>Function: When enabled, core supports TX address translation services. Provides partial support of PCI-SIG ATS specification and allows the application to set the Address Type (AT) field of the TLP header.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 'CX_ATS_ENABLE'</p> <p>Parameter Name: ATS_TX_ENABLE</p> <p>Device Type: EP</p>

5.1.2 Transmit Configuration Parameters

TABLE 5-1 describes the parameters related to TLPs transmitted from the transmit client interfaces.

TABLE 5-1 Transmit Configuration Parameters

coreConsultant Option Name	Description
Include 3rd Client Interface	<p>Function: Include the optional third transmit client interface (XALI2) in the design.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): XALI2 is included False (0): XALI2 is not included <p>Default Value: False (0)</p> <p>Parameter Name: CLIENT2_POPULATED</p> <p>Device Type: All</p>
Block Client 0 Interface Block Client 1 Interface Block Client 2 Interface	<p>Function: Determines whether or not to let the core block outbound transactions in low power modes. According to the PCIe spec, all requests except completions should be blocked in certain low power states. Because the application decides whether or not completions and other types of requests are coexisted at the client0/1/2 interface, this parameter must be configured by the application. If the client0/1/2 interface never has completions, then this parameter should be set to true. If the client0/1/2 interface has completions and other types of requests, then this parameter should be set to false, and the application is responsible for blocking all requests other than completions based on output signal pm_xtlh_block_tlp.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Client 0/1/2 interface blocking is enabled False (0): Client 0/1/2 interface blocking is disabled <p>Default Value: True (1)</p> <p>Parameter Name: CX_CLIENT0_BLOCK_NEW_TLP CX_CLIENT1_BLOCK_NEW_TLP CX_CLIENT2_BLOCK_NEW_TLP</p> <p>Device Type: All</p>
Populate ports for available credit buses	<p>Function: Include top-level ports to provide available credit information to the application.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Include top-level ports for available credits False (0): Do not include top-level ports for available credits <p>Default Value: False (0)</p> <p>Parameter Name: XADM_CRD_EN</p> <p>Device Type: All</p>

TABLE 5-1 Transmit Configuration Parameters (Continued)

coreConsultant Option Name	Description
Transmit Arbitration Method	<p>Function: Selects the arbitration method for transmitted TLPs., as described in Section 2.3.5, “Transmit TLP Arbitration” on page 56. This parameter can be changed as follows only if the number of VCs is larger than 1.</p> <ul style="list-style-type: none"> • Client based: Provides round robin arbitration between the three transmit clients. • Strict Priority: Provides strict priority between the three transmit clients. Client0 is lowest, Client1 is higher, Client2 (if implemented) is highest. • VC based: Provides VC-based arbitration priority across two classes of VCs: <ul style="list-style-type: none"> - Low priority VC (LPVC) groups can be programmed to render weighted round robin (WRR) or round robin priority. - High priority VC (HPVC) groups provide strict priority arbitration, with priority toward the highest-numbered VC IDs. <p>Value Range: 0 (VC based) or 1 (Client based) or 2 (Strict Priority) Default Value: 1 (Client based) Parameter Name: CX_XADM_ARB_MODE Device Type: All</p>
Client Interface TLP pullback feature	<p>Function: Enable the transmit clients to cancel a TLP that is currently submitted for transmission.</p> <p>Value Range: True (1) or false (0) Default Value: False (0) Parameter Name: CLIENT_PULLBACK Device Type: All</p>
Enable LPVC WRR Weights Writable	<p>Function: Determines whether the WRR arbitration VC weight values in VC Transmit Arbitration Register 1 and VC Transmit Arbitration Register 2 are writable through the DBI.</p> <p>Value Range: <ul style="list-style-type: none"> • True (1): WRR weight values are writable through the DBI • False (0): WRR weight values not are writable through the DBI </p> <p>Default Value: False (0) Parameter Name: CX_LPVC_WRR_WEIGHT_WRITABLE Device Type: All</p>

**TABLE 5-1** Transmit Configuration Parameters (Continued)

coreConsultant Option Name	Description
VC ID #0 Weight	Function: The WRR (Weighted Round Robin) weight value for VC ID 0 and VC ID 1-7. If LPVC WRR Weights Writable is true, the application can change the value by writing to VC Transmit Arbitration Register 1 through the DBI.
VC ID #1 Weight	Value Range: 0x0–0x7F
VC ID #2 Weight	Default Value:
VC ID #3 Weight	<ul style="list-style-type: none"> • 0xF (VC ID 0) • 0x0 (VC ID 1-7)
VC ID #4 Weight	Parameter Name: LPVC_WRR_WEIGHT_VC0
VC ID #5 Weight	LPVC_WRR_WEIGHT_VC1
VC ID #6 Weight	LPVC_WRR_WEIGHT_VC2
VC ID #7 Weight	LPVC_WRR_WEIGHT_VC3 LPVC_WRR_WEIGHT_VC4 LPVC_WRR_WEIGHT_VC5 LPVC_WRR_WEIGHT_VC6 LPVC_WRR_WEIGHT_VC7
	Device Type: All
Special Completion Handling	Function: Check for a specific number of Completion credits (Completion Credit Threshold) to be available before transmitting a Completion. Value Range: True (1) or false (0) Default Value: False (0) Parameter Name: SPECIAL_MAX_CPL_CRD_ENABLE Device Type: DM / RC (not EP / SW)
Completion Credit Threshold	Function: The number of Completion credits required to be available before the core will transmit a Completion. Value Range: 1–`CX_MAX_MTU/16 Default Value: `CX_MAX_MTU/16 Parameter Name: SPECIAL_MAX_CPL_CRD Device Type: DM / RC (not EP / SW)
Compare Completion Credit	Function: Enable the core to compare the actual size of a requested Completion (as opposed to a maximum size Completion) against available Completion credits before transmitting the Completion. This parameter saves gates and improves timing when set to false. Value Range: True (1) or false (0) Default Value: True (1) Parameter Name: CPL_LEN_CMP_ENABLE Device Type: All

TABLE 5-1 Transmit Configuration Parameters (Continued)

coreConsultant Option Name	Description
Special Posted Credit Handling	<p>Function: Check for a specific number of Posted credits (Posted Credit Threshold) to be available before transmitting a Posted TLP.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: SPECIAL_MAX_P_CRD_ENABLE</p> <p>Device Type: DM / RC (not EP / SW)</p>
Posted Credit Threshold	<p>Function: The number of Posted credits required to be available before the core will transmit a Posted TLP.</p> <p>Value Range: 1–`CX_MAX_MTU/16</p> <p>Default Value: `CX_MAX_MTU/16</p> <p>Parameter Name: SPECIAL_MAX_P_CRD</p> <p>Device Type: DM / RC (not EP / SW)</p>
Compare Posted Credit	<p>Function: Enable the core to compare the size of a requested Posted payload length against available Posted credits before transmitting the Posted TLP.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: P_LEN_CMP_ENABLE</p> <p>Device Type: All</p>

5.1.3 Queuing and Buffer Configuration Parameters

TABLE 5-1 lists the configuration parameters that affect the retry buffer and receive queue buffers.

TABLE 5-1 Queuing and Buffer Configuration

coreConsultant Option Name	Description
Enable Auto Size of Retry Buffer	<p>Function: Enable or disable automatic size calculation for the retry buffer.: </p> <ul style="list-style-type: none"> True (1): Enable Auto Size: The sizes of the retry buffer and SOT buffer are automatically calculated based on the Maximum Payload Size Supported value, the Link width, and device latencies. False (0): Disable Auto Size: Automatic buffer sizing is disabled. The value that you enter for Retry Buffer Depth directly sets the depth of the retry buffer. <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_RBUF_AUTOSIZE</p> <p>Device Type: All</p>
Internal Delay / Link Partner Delay	<p>Function: Read-only parameter that indicates the sum of the MAC/PHY delays, used for retry buffer auto-sizing.</p> <p>Value Range: 0–500</p> <p>Default Value: 19</p> <p>Parameter Name: CX_INTERNAL_DELAY</p> <p>Device Type: All</p>
Retry Buffer Depth	<p>Function: The depth of the retry buffer. When retry buffer auto-sizing is enabled, Retry Buffer Depth is read-only and indicates the automatically calculated depth of the retry buffer.</p> <p>When retry buffer auto-sizing is disabled, Retry Buffer Depth is the user-specified depth of the retry buffer. Note that if the autosize feature is disabled, then the minimum value specified must be large enough to allow normal operation. The default value is the automatically calculated value.</p> <p>Parameter Name: CX_RBUF_DEPTH</p> <p>Device Type: All</p>
Retry Buffer Width	<p>Function: Read-only parameter that indicates the width of the retry buffer.</p> <p>Parameter Name: RBUF_WIDTH</p> <p>Device Type: All</p>

TABLE 5-1 Queuing and Buffer Configuration (Continued)

coreConsultant Option Name	Description
Minimum SOT Depth	<p>Function: When retry buffer auto-sizing is enabled, the value displayed here corresponds to the number of minimum-sized TLPs that can reside in the retry buffer. The SOT buffer requires one entry for each TLP that can be stored in the retry buffer. The actual SOT buffer depth is a combination of the value displayed here and the additional requirements that the actual SOT buffer depth must be at least 32 and must be a power of 2.</p> <p>When retry buffer auto-sizing is disabled, Minimum SOT Depth is the user-specified depth of the SOT buffer.</p> <p>Parameter Name: CX_SOTBUF_DEPTH</p> <p>Device Type: All</p>
SOT Buffer Depth	<p>Function: Read-only parameter that indicates the depth of the SOT buffer.</p> <p>Parameter Name: SOTBUF_DEPTH</p> <p>Device Type: All</p>
SOT Buffer Width	<p>Function: Read-only parameter that indicates the width of the SOT buffer.</p> <p>Parameter Name: SOTBUF_WIDTH</p> <p>Device Type: All</p>
Specify Queue Mode	<p>Function: Specifies the memory configuration for the receive queues.:.</p> <ul style="list-style-type: none"> • Single: Use the single-buffer configuration. (DM/RC/EP only) • Multiple: Use the multiple-buffer configuration. (DM/RC/EP only) • Segment Buffer: Use the segmented-buffer configuration. <p>Value Range: 0 (Single), 1 (Multiple), or 2 (Segment Buffer)</p> <p>Default Value: 0 (Single) (DM / RC / EP) 2 (Segmented Buffer) (SW)</p> <p>Parameter Name: CX_RADMQ_MODE</p> <p>Device Type: All</p> <p>Only the segmented buffer is supported in the SW core.</p> <p>For more information, refer to Section 2.3.9, “Receive Queuing (DM / RC / EP)” on page 74.</p>

**TABLE 5-1 Queuing and Buffer Configuration (Continued)**

coreConsultant Option Name	Description
Posted Q Use Ordering FIFO	<p>Function: If Posted TLP queues are not bypassed, this parameter provides a switch to control whether the order FIFO affects Posted queue operations.:.</p> <ul style="list-style-type: none"> • If Posted Q Use Ordering FIFO is true, the Order FIFO controls the presentation of received Posted TLPs to the application. • If Posted Q Use Ordering FIFO is false, the Order FIFO does not influence the presentation of received Posted TLPs to the application. <p>Note: This parameter is for multiple buffer queue architecture only</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_RADMQ_P_NB_ORDER_LIST</p> <p>Device Type: All</p>
Non-Posted Q Use Ordering FIFO	<p>Function: If Non-Posted TLP queues are not bypassed, this parameter provides a switch to control whether the order FIFO affects Non-Posted queue operations.:.</p> <ul style="list-style-type: none"> • If Non-Posted Q Use Ordering FIFO is true, the Order FIFO controls the presentation of received Non-Posted TLPs to the application. • If Non-Posted Q Use Ordering FIFO is false, the Order FIFO does not influence the presentation of received Non-Posted TLPs to the application. <p>Note: This parameter is for multiple buffer queue architecture only</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_RADMQ_NP_NB_ORDER_LIST</p> <p>Device Type: All</p>
Completion Q Use Ordering FIFO	<p>Function: If Completion queues are not bypassed, this parameter provides a switch to control whether the order FIFO affects Completion queue operations.:.</p> <ul style="list-style-type: none"> • If Completion Q Use Ordering FIFO is true, the Order FIFO controls the presentation of received Completion TLPs to the application. • If Completion Q Use Ordering FIFO is false, the Order FIFO does not influence the presentation of received Completion TLPs to the application. <p>Note: This parameter is for multiple buffer queue architecture only</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CX_RADMQ_CPL_NB_ORDER_LIST</p> <p>Device Type: All</p>

TABLE 5-1 Queuing and Buffer Configuration (Continued)

coreConsultant Option Name	Description
Receive VC Arbitration	<p>Function: The VC arbitration scheme for sending received TLPs to the application.:.</p> <ul style="list-style-type: none"> • Strict Priority: Higher number VC IDs have higher priority • Round Robin: Round robin arbitration between VCs <p>Note: This parameter is for segment buffer queue architecture only.</p> <p>Value Range: 0 (Strict Priority) or 1 (Round Robin)</p> <p>Default Value: 1 (Round Robin)</p> <p>Parameter Name: CX_RADM_STRICT_VC_PRIORITY</p> <p>Device Type: All</p>
Support Relaxed Ordering	<p>Function: Allows Completions to be sent to the application out of order. This is valid only in segmented buffer queue mode.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: RELAXED_ORDER_SUPPORT</p> <p>Device Type: All</p>
Enable Support for Cut-Through Mode	<p>Function: Select this option if you are going to select cut-through mode for any receive queue.</p> <p>Note: This parameter is for segment buffer queue architecture only.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CUT_THROUGH_INVOLVED</p> <p>Device Type: All</p>
Enable Passing of ECRC Error Notification to the Application	<p>Function: Select this option if you want the ECRC error notification to be passed through to the application.</p> <p>Note: This parameter should be set to True if you plan on masking ECRC error filtering via the Filter Mask register.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: ECRC_ERR_PASS_THROUGH</p> <p>Device Type: All</p>
Enable Dynamic FC Credit Adjustment	<p>Function: Enable the application to update the advertised FC credit values by writing to the Port Logic registers (for example, the VCO Posted Receive Queue Control register).</p> <p>This option is only applicable in the segmented-buffer configuration.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CX_DYNAMIC_FC_CREDIT</p> <p>Device Type: All</p>

TABLE 5-1 Queuing and Buffer Configuration (Continued)

coreConsultant Option Name	Description
Enable Dynamic Q Depth Adjustment	<p>Function: Enable the application to update the receive queue buffer segment depths for individual TLP types and VCs by writing to the Port Logic registers (for example, the VC0 Posted Buffer Depth register). When using dynamic buffer sizing, the new buffer size cannot be larger than the default buffer sizes set by the Buffer Depth: Hdr and Buffer Depth: Data configuration options.</p> <p>This option is only applicable in the segmented-buffer configuration.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: CX_DYNAMIC_SEG_SIZE</p> <p>Device Type: All</p>
PCIe Ordering Rules Support	<p>Function: Enable support for the PCI Express Ordering Rule arbitration mode (only in Segmented Buffer configuration). Select this option if you are using Ordering Rule for Receive Arbitration Between Types for one or more VCs.</p> <p>Note: This parameter is for segment buffer queue architecture only</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CLUMP_SUPPORT</p> <p>Device Type: All</p>

5.1.4 VC Configuration Parameters

TABLE 5-1 lists the VC-related configuration parameters.

TABLE 5-1 VC Configuration

coreConsultant Option Name	Description
VC0: Posted Advertised Credits: Mode	<p>Function: The operating mode for the Posted TLP receive queue for VC0 only (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: 1 (Store/Fwd)</p> <p>Parameter Name: RADM_P_QMODE_VC0</p> <p>Device Type: All</p>
VC0: Posted Advertised Credits: Hdr	<p>Function: The number of Posted TLP header credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x23 (automatically calculated)</p> <p>Parameter Name: RADM_PQ_HCRD_VC0</p> <p>Device Type: All</p>
VC0: Posted Advertised Credits: Data	<p>Function: The number Posted TLP data credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x48 (automatically calculated)</p> <p>Parameter Name: RADM_PQ_DCRD_VC0</p> <p>Device Type: All</p>
VC0: Non-Posted Advertised Credits: Mode	<p>Function: The operating mode for the Non-Posted TLP receive queue for VC0 only (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: 1 (Store/Fwd)</p> <p>Parameter Name: RADM_NP_QMODE_VC0</p> <p>Device Type: All</p>
VC0: Non-Posted Advertised Credits: Hdr	<p>Function: The number Non-Posted TLP header credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x23 (automatically calculated)</p> <p>Parameter Name: RADM_NPQ_HCRD_VC0</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC0: Non-Posted Advertised Credits: Data	<p>Function: The number Non-Posted TLP data credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x5 (automatically calculated)</p> <p>Parameter Name: RADM_NPQ_DCRD_VC0</p> <p>Device Type: All</p>
VC0: Completion Advertised Credits: Mode	<p>Function: The operating mode for the Completion TLP receive queue for VC0 only (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration): store-and-forward, cut-through, or bypass.</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: 4 (Bypass)</p> <p>Parameter Name: RADM_CPL_QMODE_VC0</p> <p>Device Type: All</p>
VC0: Completion Advertised Credits: Hdr	<p>Function: The number Completion TLP header credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: RADM_CPLQ_HCRD_VC0</p> <p>Device Type: All</p>
VC0: Completion Advertised Credits: Data	<p>Function: The number of Completion TLP data credits to advertise for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: RADM_CPLQ_DCRD_VC0</p> <p>Device Type: All</p>
VC0: Receive Arbitration Between Types	<p>Function: Specifies the arbitration scheme for sending received TLPs to the application for VC0 (applicable only in the segmented-buffer configuration).:</p> <ul style="list-style-type: none"> • Strict Priority: The priority order is Posted, then Completion, then Non-Posted • Ordering Rule: The arbitration is according to the ordering rules the <i>PCI Express 2.0 specification</i> <p>Value Range: 0 (Strict Priority) or 1 (Ordering Rule)</p> <p>Default Value: 0 (Strict Priority)</p> <p>Parameter Name: CX_RADM_ORDERING_RULES_VC0</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC0: Decouple Depth from Credit	<p>Function: Allow the number of advertised credits and receive queue depths to be specified independently for VC0. Otherwise, the queue depths are automatically determined from the number of advertised credits.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: RADM_DEPTH_DECUPLE_VC0</p> <p>Device Type: All</p>
VC0: Posted Buffer Depth: Hdr	<p>Function: The number of entries in the Posted header buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: 46</p> <p>Parameter Name: RADM_PQ_HDP_VC0</p> <p>Device Type: All</p>
VC0: Posted Buffer Depth: Data	<p>Function: The number of entries in the Posted data buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: 212</p> <p>Parameter Name: RADM_PQ_DDP_VC0</p> <p>Device Type: All</p>
VC0: Non-Posted Buffer Depth: Hdr	<p>Function: The number of entries in the Non-Posted header buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: 0</p> <p>Parameter Name: RADM_NPQ_HDP_VC0</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC0: Non-Posted Buffer Depth: Data	<p>Function: The number of entries in the Non-Posted data buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: 0</p> <p>Parameter Name: RADM_NPQ_DDP_VC0</p> <p>Device Type: All</p>
VC0: Completion Buffer Depth: Hdr	<p>Function: The number of entries in the Completion header buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: 0</p> <p>Parameter Name: RADM_CPLQ_HDP_VC0</p> <p>Device Type: All</p>
VC0: Completion Buffer Depth: Data	<p>Function: The number of entries in the Completion data buffer for VC0 (in the segmented-buffer configuration) or all VCs (in the multiple or single-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: 0</p> <p>Parameter Name: RADM_CPLQ_DDP_VC0</p> <p>Device Type: All</p>
VC1–VC7: Posted Advertised Credits: Mode	<p>Function: The operating mode for the Posted TLP receive queue for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_P_QMODE_VC1 through RADM_P_QMODE_VC7</p> <p>Device Type: All</p>
VC1–VC7: Posted Advertised Credits: Hdr	<p>Function: The number of Posted TLP header credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_PQ_HCRD_VC1 through RADM_PQ_HCRD_VC7</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC1–VC7: Posted Advertised Credits: Data	<p>Function: The number of Posted TLP data credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_PQ_DCRD_VC1 through RADM_PQ_DCRD_VC7</p> <p>Device Type: All</p>
VC1–VC7: Non-Posted Advertised Credits: Mode	<p>Function: The operating mode for the Non-Posted TLP receive queue for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_NP_QMODE_VC1 through RADM_NP_QMODE_VC7</p> <p>Device Type: All</p>
VC1–VC7: Non-Posted Advertised Credits: Hdr	<p>Function: The number of Non-Posted TLP header credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_NPQ_HCRD_VC1 through RADM_NPQ_HCRD_VC7</p> <p>Device Type: All</p>
VC1–VC7: Non-Posted Advertised Credits: Data	<p>Function: The number of Non-Posted TLP data credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_NPQ_DCRD_VC1 through RADM_NPQ_DCRD_VC7</p> <p>Device Type: All</p>
VC1–VC7: Completion Advertised Credits: Mode	<p>Function: The operating mode for the Completion TLP receive queue for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 1 (Store/Fwd), 2 (Cut Thru), or 4 (Bypass)</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_CPL_QMODE_VC1 through RADM_CPL_QMODE_VC7</p> <p>Device Type: All</p>

**TABLE 5-1 VC Configuration (Continued)**

coreConsultant Option Name	Description
VC1–VC7: Completion Advertised Credits: Hdr	<p>Function: The number of Completion TLP header credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_CPLQ_HCRD_VC1 through RADM_CPLQ_HCRD_VC7</p> <p>Device Type: All</p>
VC1–VC7: Completion Advertised Credits: Data	<p>Function: The number of Completion TLP data credits to advertise for VC1–VC7 (editable only in the segmented-buffer configuration).</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: Same as the current value for VC0</p> <p>Parameter Names: RADM_CPLQ_DCRD_VC1 through RADM_CPLQ_DCRD_VC7</p> <p>Device Type: All</p>
VC1–VC7: Receive Arbitration Between Types	<p>Function: Specifies the arbitration scheme for sending received TLPs to the application (applicable only in the segmented-buffer configuration):</p> <ul style="list-style-type: none"> • Strict Priority: The priority order is Posted, then Completion, then Non-Posted • Ordering Rule: The arbitration is according to the ordering rules in <i>PCI Express 2.0 specification</i>. <p>Value Range: 0 (Strict Priority) or 1 (Ordering Rule)</p> <p>Default Value: 0 (Strict Priority)</p> <p>Parameter Name: CX_RADM_ORDERING_RULES_VC1 through CX_RADM_ORDERING_RULES_VC7</p> <p>Device Type: All</p>
VC1–VC7: Decouple Depth from Credit	<p>Function: Allow the number of advertised credits and receive queue depths to be specified independently for VC1–VC7 (editable only in the segmented-buffer configuration). Otherwise, the queue depths are automatically determined from the number of advertised credits.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Names: RADM_DEPTH_DECOPPLE_VC1 through RADM_DEPTH_DECOPPLE_VC7</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC1–VC7: Posted Buffer Depth: Hdr	<p>Function: The number of entries in the Posted header buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_PQ_HDP_VC1 through RADM_PQ_HDP_VC7</p> <p>Device Type: All</p>
VC1–VC7: Posted Buffer Depth: Data	<p>Function: The number of entries in the Posted data buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_PQ_DDP_VC1 through RADM_PQ_DDP_VC7</p> <p>Device Type: All</p>
VC1–VC7: Non-Posted Buffer Depth: Hdr	<p>Function: The number of entries in the Non-Posted header buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_NPQ_HDP_VC1 through RADM_NPQ_HDP_VC7</p> <p>Device Type: All</p>
VC1–VC7: Non-Posted Buffer Depth: Data	<p>Function: The number of entries in the Non-Posted data buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_NPQ_DDP_VC1 through RADM_NPQ_DDP_VC7</p> <p>Device Type: All</p>

TABLE 5-1 VC Configuration (Continued)

coreConsultant Option Name	Description
VC1–VC7: Completion Buffer Depth: Hdr	<p>Function: The number of entries in the Completion header buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–1023</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_CPLQ_HDP_VC1 through RADM_CPLQ_HDP_VC7</p> <p>Device Type: All</p>
VC1–VC7: Completion Buffer Depth: Data	<p>Function: The number of entries in the Completion data buffer for VC1–VC7 (editable only in the segmented-buffer configuration). This option is read-only if the queue is bypassed or Decouple Depth from Credit is not selected.</p> <p>Value Range: 0–16383</p> <p>Default Value: Automatically calculated from credit value</p> <p>Parameter Names: RADM_CPLQ_DDP_VC1 through RADM_CPLQ_DDP_VC7</p> <p>Device Type: All</p>

5.1.5 Filter Configuration Parameters

TABLE 5-1 defines the filter configuration parameters.

TABLE 5-1 Filter Configuration Parameters

coreConsultant Option Name	Description
Number of Address Bits Passed to the Application	<p>Function: Indicates the number of address bits that are passed through the receive queues and are sent to the application on the RTRGT1 interface (radm_trgt1_addr). If vendor message support (via VENDOR_MESSAGE_SUPPORT parameter) is not needed, a narrower address bus can be used. This saves gates and makes some RAMs significantly more narrow.</p> <p>Value Range: 26-64</p> <p>Minimum number of bits allowed is 32 if Target 1 interface is populated.</p> <p>Minimum number of bits allowed is 26 if Target 1 interface is not populated.</p> <p>Default Value: If VENDOR_MESSAGE_SUPPORT=1, then default is 64. Otherwise, default is 32.</p> <p>Parameter Name: FLT_Q_ADDR_WIDTH</p> <p>Device Type: All</p> <p>Read only for the Switch.</p>
Allow AER (UR/CA Error) for TLPs Destined for Target 1	<p>Function: Determines whether to enable Advanced Error Reporting (AER) for Unsupported Request (UR) and Completer Abort (CA) errors for TLPs that are targeted to RTRGT1.</p> <p>Value Range: 0 (Supress_Trgt1_AER) or 1 (Allow_Trgt1_AER)</p> <p>Default Value: 0 (Supress_Trgt1_AER)</p> <p>Parameter Name: CX_MASK_UR_CA_4_TRGT1</p> <p>Device Type: All</p>
FLT Message Drop	<p>Function: Determines whether to drop Message TLPs or pass them to the application on RTRGT1. In either case, the core passes the content of the decoded Message to the application on the SII:</p> <ul style="list-style-type: none"> True (1): Discard Message after decoding False (0): Pass Message TLP to application on RTRGT1 <p>The application can override the value of this option at runtime by writing to the Symbol Timer Register and Filter Mask Register 1.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: FLT_DROP_MSG</p> <p>Device Type: All</p>

TABLE 5-1 Filter Configuration Parameters (Continued)

coreConsultant Option Name	Description
Filter Mask Register 1	<p>Function: Default value for Filter Mask Register 1 (Symbol Timer Register and Filter Mask Register 1).</p> <p>Value Range: 0 – 0xFFFF</p> <p>Default Value: 0 or 0x2000 if FLT_DROP_MSG is not set</p> <p>Parameter Name: DEFAULT_FILTER_MASK_1</p> <p>Device Type: All</p>
**Filter Mask Register 2 (not visible to coreConsultant)	<p>Function: Default value for Filter Mask Register 2.</p> <p>Value Range: 0 – 0xFFFFFFFF</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_FILTER_MASK_2</p> <p>Device Type: All</p>
**ENABLE_MEM_MAP_PL_REG (not visible in coreConsultant)	<p>Function: Enables routing of memory transactions to the Port Logic configuration registers.</p> <p>Value Range: Disabled (0) or Enabled (1)</p> <p>Default Value: Disabled (0)</p> <p>Parameter Name: ENABLE_MEM_MAP_PL_REG</p> <p>Device Type: DM/EP/SW (not RC)</p>

5.1.6 Common Register Configuration Parameters

TABLE 5-1 defines the configuration parameters that set default values for the non-function-specific configuration registers.

TABLE 5-1 Common Register Configuration Parameters

coreConsultant Option Name	Description
Upper Configuration Limit	<p>Function: Specifies a limit above which incoming configuration requests will be routed to the destination interface defined by TARGET_ABOVE_CONFIG_LIMIT. The TARGET_ABOVE_CONFIG_LIMIT defines either target1 or target0 as a destination. The CONFIG_LIMIT parameter is normally set to a limit that divides the core's configure space registers from the application's configure space registers. The core LBC module uses this limit to direct a configuration request to the CDM or ELBI. We recommend that the application sets a proper value based on its extended configuration registers.</p> <p>Value Range: 0–0x3FF (4K range)</p> <p>Default Value: 0x3FF</p> <p>Parameter Name: CONFIG_LIMIT</p> <p>Device Type: All</p>
Target of Configuration Above Upper Limit	<p>Function: Target Interface Destination for configuration requests that the configuration register address is above CONFIG_LIMIT.</p> <p>Value Range: Target_0 (1) or Target_1 (2)</p> <p>Default Value: TRGT1_POPULATE==1 ? Target_1 : Target_0</p> <p>Parameter Name: TARGET_ABOVE_CONFIG_LIMIT</p> <p>Device Type: All</p>
Default Target Interface	<p>Function: Specifies the default target interface for incoming Requests that are not mapped to either RTRGT0 or RTRGT1 through the map-by-BAR parameters.</p> <p>Value Range: Target_0 (0) or Target_1 (1)</p> <p>Default Value: Target_0 (0)</p> <p>Parameter Name: DEFAULT_TARGET</p> <p>Device Type: All</p> <p>This option applies only when the DM core is operating in EP mode.</p>
Target CPL LUT Enable	<p>Function: Enable the Completion lookup table, as described in Section 3.2.1, “Transmit Client Interface Protocol Rules” on page 114.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: TRGT_CPL_LUT_EN</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Maximum Remote Tags Supported	<p>Function: Specifies the maximum number of tags track in the Target Completion LUT Used to size the target completion look-up-table and timeout RAM.</p> <p>Value Range: 2, 4, 8, 16, 32, 64, 128, or 256</p> <p>Default Value: `CX_MAX_TAG</p> <p>Parameter Name: CX_REMOTE_MAX_TAG</p> <p>Device Type: All</p>
RADM CPL LUT Store Byte Count	<p>Function: Determines whether or not to store the byte count in the RADM Completion lookup table.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Device Type: All</p> <p>Parameter Name: RADM_CPL_LUT_STORE_BYTE_CNT</p>
Client Data/Address Bus Parity Protection	<p>Function: Determines whether to generate and check parity on the transmit client interface address and data buses, and internal datapaths, and the bus widths on which to calculate parity.</p> <p>If you select 8, 16, or 32-bit parity checking, the core implements parity checking (even) parity on the transmit and receive datapath segments that are not covered by ECRC. If a parity error occurs, the core asserts a pulse on the associated bit of app_parity_errs[2:0], but does not drop the associated packet.</p> <p>Value Range: 0 (None), 32 (32-bit), 16 (16-bit), or 8 (8-bit)</p> <p>Default Value: 0 (None)</p> <p>Parameter Name: CX_CLIENT_PAR_MODE</p> <p>Device Type: All</p>
Application Par Error Out Enable	<p>Function: Include top-level parity error signals.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: APP_PAR_ERR_OUT_EN</p> <p>Device Type: All</p>
Application Return CRD Enable	<p>Function: Include optional top-level ports for the application to update the Flow Control credits counters. The optional ports are the app_*_ca ports on the SII (see Section 3.13, “System Information Interface (SII)” on page 200).</p> <p>Note: The application is not expected to return completion credits since completion credits are infinite per the <i>PCI Express 2.0 specification</i>.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: APP_RETURN_CRD_EN</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Default Link Number	<p>Function: Default value for the Link Number field in the Port Force Link Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x4</p> <p>Parameter Name: DEFAULT_LINK_NUM</p> <p>Device Type: All</p>
Default ACK Frequency	<p>Function: Default value for the Ack Frequency field in the Ack Frequency Register. A value of 0 in the Ack Frequency register indicates that this Ack frequency control feature is turned off. A value of 1 indicates that the core will ack every TLP received.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: DEFAULT_ACK_FREQUENCY</p> <p>Device Type: All</p>
Default Replay Timer Adjust	<p>Function: Default value for the Timer Modifier for Replay Timer field in the Symbol Number Register. Each value increases the replay timer by 64.</p> <p>Value Range: 0x0–0x1F</p> <p>Default Value: (NFTS*4) / (NB*64)</p> <p>Parameter Name: DEFAULT_REPLAY_ADJ</p> <p>Device Type: All</p>
Default L1 Entry Latency	<p>Function: Default value for the L1 Entrance Latency field.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 000: 1 µs • 001: 2 µs • 010: 4 µs • 011: 8 µs • 100: 16 µs <p>Default Value: Value is set automatically unless Custom or Generic PHY is used.</p> <p>Parameter Name: DEFAULT_L1_ENTR_LATENCY</p> <p>Device Type: All</p>

**TABLE 5-1 Common Register Configuration Parameters (Continued)**

coreConsultant Option Name	Description
Default L0S Entry Latency	<p>Function: Default value for the L0s Entrance Latency field.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 000: 1 μs • 001: 2 μs • 010: 3 μs • 011: 4 μs • 100: 5 μs • 101: 6 μs • 110 or 111: 7 μs <p>Default Value: Value is set automatically unless Custom or Generic PHY is used.</p> <p>Parameter Name: DEFAULT_L0S_ENTR_LATENCY</p> <p>Device Type: All</p>
MSI Capability	<p>Function: Include the MSI Capability structure.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: MSI_CAP_ENABLE</p> <p>Device Type: All</p>
MSI IO Enable	<p>Function: Include optional top-level ports for MSI, as listed in Section 3.8, “Message Signaled Interrupt (MSI) Interface (DM / EP / SW)” on page 183.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: MSI_IO</p> <p>Device Type: All</p>
Enable 64-bit MSI Support	<p>Function: Default value for the 64-bit Address Capable bit in the MSI Control Register.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: MSI_64_EN</p> <p>Device Type: All</p> <p>This parameter is adjustable only when the Device Type is set to RC.</p>
Default Multiple MSI Capability	<p>Function: Default value for the Multiple Message Capable field in the MSI Control Register.</p> <p>Value Range: 0x0–0x7</p> <p>Default Value: 0x0</p> <p>Parameter Name: DEFAULT_MULTI_MSI_CAPABLE</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
MSI PVM Support	<p>Function: Support MSI Capability Per Vector Masking Value Range: 0 or 1 <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) Default Value: 'MSI_CAP_ENABLE and NOT('CX_SRIOV_ENABLE) Parameter Name: MSI_PVM_EN Device Type: EP</p>
MSI-X Capability	<p>Function: Include the MSI-X Capability structure. Value Range: True (1) or false (0) Default Value: False (0) Parameter Name: MSIX_CAP_ENABLE Device Type: All</p>
MSI-X IO Enable	<p>Function: Include optional top-level ports for MSI-X, as listed in Section 3.9, "MSI-X Interface (optional) (DM / EP / SW)" on page 188. If included, the MSI-X ports are used only when the core is operating in EP mode. Value Range: True (1) or false (0) Default Value: False (0) Parameter Name: MSIX_IO Device Type: DM / EP / SW (not RC)</p>
L1 Exit Latency	<p>Function: Default value for the L1 Exit Latency field in the Link Capabilities Register. This parameter represents a characteristic of the PHY being used. It measures the total time from when the core initiates a transition from P1 to P0 until the PHY begins providing valid receive data to the core.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 1 μs) • 1 (1 μs to less than 2 μs) • 2 (2 μs to less than 4 μs) • 3 (4 μs to less than 8 μs) • 4 (8 μs to less than 16 μs) • 5 (16 μs to less than 32 μs) • 6 (32 μs to 64 μs) • 7 (More than 64 μs) <p>Default Value: 6 (32 μs to 64 μs) Parameter Name: DEFAULT_L1_EXIT_LATENCY Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
L1 Exit Latency (Common Clock)	<p>Function: Default value for the L1 Exit Latency field in the Link Capabilities Register, when common clock is used. It measures the total time from when the core initiates a transition from P1 to P0 until the PHY begins providing valid receive data to the core.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 1 µs) • 1 (1 µs to less than 2 µs) • 2 (2 µs to less than 4 µs) • 3 (4 µs to less than 8 µs) • 4 (8 µs to less than 16 µs) • 5 (16 µs to less than 32 µs) • 6 (32 µs to 64 µs) • 7 (More than 64 µs) <p>Default Value: 6 (32 µs to 64 µs)</p> <p>Parameter Name: DEFAULT_COMM_L1_EXIT_LATENCY</p> <p>Device Type: All</p>
**L0s Exit Latency (not visible to coreConsultant)	<p>Function: Default value for the L0s Exit Latency field in the Link Capabilities Register.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 64 ns) • 1 (64 ns to less than 128 ns) • 2 (128 ns to less than 256 ns) • 3 (256 ns to less than 512 ns) • 4 (512 ns to less than 1 µs) • 5 (1 µs to less than 2 µs) • 6 (2 µs to 4 µs) • 7 (More than 4 µs) <p>Default Value: 3 (256 ns to less than 512 ns)</p> <p>Parameter Name: DEFAULT_LOS_EXIT_LATENCY</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
**L0s Exit Latency (Common Clock) (not visible to coreConsultant)	<p>Function: Default value for the L0s Exit Latency field in the Link Capabilities Register, when common clock is used.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 64 ns) • 1 (64 ns to less than 128 ns) • 2 (128 ns to less than 256 ns) • 3 (256 ns to less than 512 ns) • 4 (512 ns to less than 1 µs) • 5 (1 µs to less than 2 µs) • 6 (2 µs to 4 µs) • 7 (More than 4 µs) <p>Default Value: 3 (256 ns to less than 512 ns)</p> <p>Parameter Name: DEFAULT_COMM_L0S_EXIT_LATENCY</p> <p>Device Type: All</p>
Use Platform Reference Clock	<p>Function: Default value for the Slot Clock Configuration bit in the Link Status Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 1</p> <p>Parameter Name: SLOT_CLK_CONFIG</p> <p>Device Type: All</p>
Active State Link PM Support	<p>Function: Default value for the Active State Link PM Support field in the Link Capabilities Register.</p> <p>Value Range: 0x1 (L0s Entry Supported) or 0x3 (L0s and L1 Entry Supported)</p> <p>Default Value: 0x3 (L0s and L1 Entry Supported)</p> <p>Parameter Name: AS_LINK_PM_SUPT</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Default L0S Accept Latency	<p>Function: Default value for the Endpoint L0s Acceptable Latency field in the Device Capabilities Register.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 64 ns) • 1 (64 ns to less than 128 ns) • 2 (128 ns to less than 256 ns) • 3 (256 ns to less than 512 ns) • 4 (512 ns to less than 1 µs) • 5 (1 µs to less than 2 µs) • 6 (2 µs to 4 µs) • 7 (More than 4 µs) <p>Default Value: 4 (512 ns to less than 1 µs)</p> <p>Parameter Name: DEFAULT_EP_L0S_ACCPT_LATENCY</p> <p>Device Type: DM / EP (not RC / SW)</p>
Default L1 Accept Latency	<p>Function: Default value for the Endpoint L1 Acceptable Latency field in the Device Capabilities Register.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (Less than 1 µs) • 1 (1 µs to less than 2 µs) • 2 (2 µs to less than 4 µs) • 3 (4 µs to less than 8 µs) • 4 (8 µs to less than 16 µs) • 5 (16 µs to less than 32 µs) • 6 (32 µs to 64 µs) • 7 (More than 64 µs) <p>Default Value: 3 (4 µs to less than 8 µs)</p> <p>Parameter Name: DEFAULT_EP_L1_ACCPT_LATENCY</p> <p>Device Type: DM / EP (not RC / SW)</p>
Port Number	<p>Function: Default value for the Port Number field in the Link Capabilities Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: PORT_NUM</p> <p>Device Type: All</p>
RCB Support	<p>Function: Support the ability of configuration software to indicate the Read Completion Boundary value.</p> <p>Value Range: True (1) or False (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: CX_RCB_SUPPORT</p> <p>Device Type: DM / EP (not RC / SW)</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Physical Slot Number	<p>Function: Default value for the Physical Slot Number field in the Slot Capabilities Register.</p> <p>Value Range: 0x0–0x1FFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: SLOT_PHY_SLOT_NUM</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Power Limit Scale	<p>Function: Default value for the Slot Power Limit Scale field in the Slot Capabilities Register.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (1.0x) • 1 (0.1x) • 2 (0.01x) • 3 (0.001x) <p>Default Value: 0x0</p> <p>Parameter Name: SET_SLOT_PWR_LIMIT_SCALE</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Power Limit Value	<p>Function: Default value for the Slot Power Limit Value field in the Slot Capabilities Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: SET_SLOT_PWR_LIMIT_VAL</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot is Hot-Plug Capable	<p>Function: Default value for the Hot-Plug Capable bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_HP_CAPABLE</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Support Hot-Plug Surprise	<p>Function: Default value for the Hot-Plug Surprise bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_HP_SURPRISE</p> <p>Device Type: DM / RC / SW (not EP)</p>

**TABLE 5-1 Common Register Configuration Parameters (Continued)**

coreConsultant Option Name	Description
Disable Hot-Plug Software Notification	<p>Function: Default value for the No Command Complete Support bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_NO_CC_SUPPORT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Electromechanical Interlock Implemented	<p>Function: Default value for the Electromechanical Interlock Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_EML_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Power Indicator Present	<p>Function: Default value for the Power Indicator Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_PWR_IND_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Attention Indicator Present	<p>Function: Default value for the Attention Indicator Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_ATTEN_IND_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot MRL Sensor Present	<p>Function: Default value for the MRL Sensor Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_MRL_SENSOR_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Slot Power Controller Present	<p>Function: Default value for the Power Controller Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_PWR_CTRL_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Slot Attention Button Present	<p>Function: Default value for the Attention Button Present bit in the Slot Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_ATTEN_BUTTON_PRESENT</p> <p>Device Type: DM / RC / SW (not EP)</p>
Clock PM Support	<p>Function: Default value for the Clock Power Management bit in the Link Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_CLK_PM_CAP</p> <p>Device Type: All</p>
Support Advanced Error Reporting	<p>Function: Support PCI Express Advanced Error Reporting (required if ECRC is enabled).</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: AER_ENABLE</p> <p>Device Type: All</p>
Serial Number Capability	<p>Function: Support Device Serial Number Capability</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: SERIAL_CAP_ENABLE</p> <p>Device Type: All</p>
Device Serial Number (1 st DW)	<p>Function: Specifies the first 32-bits of the device serial number</p> <p>Value Range: 0x0 – 0xFFFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: DEFAULT_SN_DW1</p> <p>Device Type: All</p>
Device Serial Number (2 nd DW)	<p>Function: Specifies the second 32-bits of the device serial number</p> <p>Value Range: 0x0 – 0xFFFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: DEFAULT_SN_DW2</p> <p>Device Type: All</p>
Power Budget Capability	<p>Function: Support Power Budgeting Capability</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: PWR_BUDGET_CAP_ENABLE</p> <p>Device Type: All</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Power Budget System Allocation	<p>Function: Default value for the System Allocated bit in the Power Budget Capability Register, writable through DBI.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: DEFAULT_PWR_BUDGET_SYS_ALLOC</p> <p>Device Type: All</p>
Virtual Channel Capability	
Vital Product Data (VPD)	<p>Function: Support Vital Product Data Capability</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: VPD_CAP_ENABLE</p> <p>Device Type: All</p>
Virtual Channel Support	<p>Function: Support PCI Express Virtual Channel Capability (required if there are multiple VCs).</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: True (1)</p> <p>Parameter Name: VC_ENABLE</p> <p>Device Type: All</p>
VC Arbitration Capability	<p>Function: Default value for the VC Arbitration Capability field in Port VC Capability Register 2.</p> <p>Value Range: 0x0–0xF</p> <p>Default Value: 0x0</p> <p>Parameter Name: DEFAULT_VC_ARB_32</p> <p>Device Type: All</p>
Low Priority Extended VC Count	<p>Function: Default value for the Low Priority Extended VC Count field in Port VC Capability Register 1.</p> <p>Value Range: 0–`CX_NVC</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_LOW_PRI_EXT_VC_CNT</p> <p>Device Type: All</p>
Slot ID Capability	<p>Function: Enable Slot ID Capability Structure</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <p>Parameter Name: SLOT_CAP_ENABLE</p> <p>Device Type: SW only (not DM / RC / EP)</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
Slot Number	<p>Function: Contains the binary value of the number of PCI expansion slots located directly on the secondary interface of this bridge.</p> <p>Value Range:</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_NUM</p> <p>Device Type: SW only (not DM / RC / EP)</p>
Slot First In Chassis	<p>Function: When set, it indicates that this bridge is the first in an expansion chassis.</p> <p>Value Range:</p> <p>Default Value: 0</p> <p>Parameter Name: FIRST_IN_CHASSIS</p> <p>Device Type: SW only (not DM / RC / EP)</p>
Selectable De-emphasis	<p>Function: Select the de-emphasis level.</p> <p>Value Range:</p> <ul style="list-style-type: none"> 0: -6 dB 1: -3.5 dB <p>Default Value: 0</p> <p>Parameter Name: SEL_DE_EMPHASIS</p> <p>Device Type: DM / RC / SW (not EP)</p>
	<p style="text-align: center;">SR-IOV Capability</p>
Support SR-IOV	<p>Function: Enable SR-IOV Capability</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_SRIOV_ENABLE</p> <p>Device Type: EP</p>
Virtual Function Dependency Link Support	<p>Function: Enables support for VF dependency link. Valid only if CX_NFUNC>1.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_VF_DEPENDENCY_LINK_SUPP</p> <p>Device Type: EP / DM (EP mode)</p>

TABLE 5-1 Common Register Configuration Parameters (Continued)

coreConsultant Option Name	Description
VF MSI-X Capability	<p>Function: Enable VF MSI-X Capability</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 'CX_SRIOV_ENABLE</p> <p>Parameter Name: VF_MSIX_CAP_ENABLE</p> <p>Device Type: EP / DM (EP mode)</p>
VF MSI-X Table Size	<p>Function: Default value for the MSI-X Table Size field in the VF MSI-X Control Register.</p> <p>Value Range: 0x0–0x7FF</p> <p>Default Value: 0x0</p> <p>Parameter Name: VF_MSIX_TABLE_SIZE</p> <p>Device Type: EP / DM (EP mode)</p>
VF MSI-X Table BIR	<p>Function: Default value for the Table BAR Indicator Register (BIR) field in the VF MSI-X Table Offset and BIR Register. Indicates which BAR is used to map the MSI-X Table into memory space.</p> <p>Value Range: 000 (BAR0) through 101 (BAR5)</p> <p>Default Value: 000 (BAR0)</p> <p>Parameter Name: VF_MSIX_TABLE_BIR</p> <p>Device Type: EP / DM (EP mode)</p>
VF MSI-X Table Offset	<p>Function: Default value for the Table Offset field in the VF MSI-X Table Offset and BIR Register.</p> <p>Value Range: 0x0–0x1FFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: VF_MSIX_TABLE_OFFSET</p> <p>Device Type: EP / DM (EP mode)</p>
VF MSI-X PBA BIR	<p>Function: Default value for the Pending Bit Array (PBA) BIR field in the VF MSI-X PBA Offset and BIR Register. Indicates which BAR is used to map the MSI-X PBA into memory space.</p> <p>Value Range: 000 (BAR0) through 101 (BAR5)</p> <p>Default Value: 000 (BAR0)</p> <p>Parameter Name: VF_MSIX_PBA_BIR</p> <p>Device Type: EP / DM (EP mode)</p>
VF MSI-X PBA Offset	<p>Function: Default value for the PBA Offset field in the VF MSI-X PBA Offset and BIR Register.</p> <p>Value Range: 0x0–0x1FFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: VF_MSIX_PBA_OFFSET</p> <p>Device Type: EP / DM (EP mode)</p>

5.1.7 Function Configuration Parameters

TABLE 5-1 lists the parameters that are unique to each function in multi-function configuration. The N at the end of each parameter name is the function number (0-7).

TABLE 5-1 Function Configuration Parameters

coreConsultant Option Name	Description
PCI Express Capability	
PCIe Capabilities Interrupt Message Number	<p>Function: Default value for the Interrupt Message Number field in the PCI Express Capabilities Register.</p> <p>Value Range: 0x0–0x1F</p> <p>Default Value: 0x0</p> <p>Parameter Name: PCIE_CAP_INT_MSG_NUM_N (PCIE_CAP_INT_MSG_NUM_0 through PCIE_CAP_INT_MSG_NUM_7)</p> <p>Device Type: All for PCIE_CAP_INT_MSG_NUM_0 DM / EP (not RC / SW) for PCIE_CAP_INT_MSG_NUM_1-7</p>
Is Port Connected to Slot	
Extended Tag Support	<p>Function: Indicates that the PCI Express Link associated with this Port is connected to a slot.</p> <p>Value Range: 0 (not set) or 1 (set)</p> <p>Default Value: 0</p> <p>Parameter Name: SLOT_IMPLEMENTED_N (SLOT_IMPLEMENTED_0 through SLOT_IMPLEMENTED_7)</p> <p>Device Type: All for SLOT_IMPLEMENTED_0 DM / EP (not RC / SW) for SLOT_IMPLEMENTED_1-7</p>
Add Support For Attention Button	<p>Function: Default value for the Attention Button Present bit in the Device Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_ATT_BUTT_PRE_N (DEFAULT_ATT_BUTT_PRE_0 through DEFAULT_ATT_BUTT_PRE_7)</p> <p>Device Type: All for DEFAULT_ATT_BUTT_PRE_0 DM / EP (not RC / SW) for DEFAULT_ATT_BUTT_PRE_1-7</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Add Support For Attention Indicator	<p>Function: Default value for the Attention Indicator Present bit in the Device Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_ATT_IND_PRE_N (DEFAULT_ATT_IND_PRE_0 through DEFAULT_ATT_IND_PRE_7)</p> <p>Device Type: All for DEFAULT_ATT_IND_PRE_0 DM / EP (not RC / SW) for DEFAULT_ATT_IND_PRE_1-7</p>
Add Support For Power Indicator	<p>Function: Default value for the Power Indicator Present bit in the Device Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_PWR_IND_PRE_N (DEFAULT_PWR_IND_PRE_0 through DEFAULT_PWR_IND_PRE_7)</p> <p>Device Type: All for DEFAULT_PWR_IND_PRE_0 DM / EP (not RC / SW) for DEFAULT_PWR_IND_PRE_1-7</p>
Support No-Snoop	<p>Function: Default value for the Enable No Snoop bit in the Device Control Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_NO_SNOOP_SUPPORTED_N (DEFAULT_NO_SNOOP_SUPPORTED_0 through DEFAULT_NO_SNOOP_SUPPORTED_7)</p> <p>Device Type: All for DEFAULT_NO_SNOOP_SUPPORTED_0 DM / EP (not RC / SW) for DEFAULT_NO_SNOOP_SUPPORTED_1-7</p>
Enable Root RCB	<p>Function: Default value for the Read Completion Boundary (RCB) bit in the Link Control Register.</p> <p>Value Range: 0 (64-bit) or 1 (128-bit)</p> <p>Default Value: 0 (64-bit)</p> <p>Parameter Name: ROOT_RCB_N</p> <p>Device Type: DM / RC (not EP / SW)</p>

Per-PF MSI-X Register Configuration

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
MSI-X Table Size	<p>Function: Default value for the MSI-X Table Size field in the MSI-X Control Register.</p> <p>Value Range: 0x0–0x7FF</p> <p>Default Value: 0x0</p> <p>Parameter Name: MSIX_TABLE_SIZE_N (MSIX_TABLE_SIZE_0 through MSIX_TABLE_SIZE_7)</p> <p>Device Type: All for MSIX_TABLE_SIZE_0 DM / EP (not RC / SW) for MSIX_TABLE_SIZE_1-7</p>
MSI-X Table BIR	<p>Function: Default value for the Table BAR Indicator Register (BIR) field in the MSI-X Table Offset and BIR Register.</p> <p>Value Range: 000 (BAR0) through 101 (BAR5)</p> <p>Default Value: 000 (BAR0)</p> <p>Parameter Name: MSIX_TABLE_BIR_N (MSIX_TABLE_BIR_0 through MSIX_TABLE_BIR_7)</p> <p>Device Type: All for MSIX_TABLE_BIR_0 DM / EP (not RC / SW) for MSIX_TABLE_BIR_1-7</p>
MSI-X Table Offset	<p>Function: Default value for the Table Offset field in the MSI-X Table Offset and BIR Register.</p> <p>Value Range: 0x0–0x1FFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: MSIX_TABLE_OFFSET_N (MSIX_TABLE_OFFSET_0 through MSIX_TABLE_OFFSET_7)</p> <p>Device Type: All for MSIX_TABLE_OFFSET_0 DM / EP (not RC / SW) for MSIX_TABLE_OFFSET_1-7</p>
MSI-X PBA BIR	<p>Function: Default value for the Pending Bit Array (PBA) BIR field in the MSI-X PBA Offset and BIR Register.</p> <p>Value Range: 000 (BAR0) through 101 (BAR5)</p> <p>Default Value: 000 (BAR0)</p> <p>Parameter Name: MSIX_PBA_BIR_N (MSIX_PBA_BIR_0 through MSIX_PBA_BIR_7)</p> <p>Device Type: All for MSIX_PBA_BIR_0 DM / EP (not RC / SW) for MSIX_PBA_BIR_1-7</p>
MSI-X PBA Offset	<p>Function: Default value for the PBA Offset field in the MSI-X PBA Offset and BIR Register.</p> <p>Value Range: 0x0–0x1FFFFFFF</p> <p>Default Value: 0x0</p> <p>Parameter Name: MSIX_PBA_OFFSET_N (MSIX_PBA_OFFSET_0 through MSIX_PBA_OFFSET_7)</p> <p>Device Type: DM / EP (not RC / SW)</p>

Per-PF Advanced Error Register Configuration

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Default ECRC Check Capability	<p>Function: Default value for the ECRC Check Capable bit in the Advanced Capabilities and Control Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 1</p> <p>Parameter Name: DEFAULT_ECRC_CHK_CAP_N (DEFAULT_ECRC_CHK_CAP_0 through DEFAULT_ECRC_CHK_CAP_7)</p> <p>Device Type: All for DEFAULT_ECRC_CHK_CAP_0 DM / EP (not RC / SW) for DEFAULT_ECRC_CHK_CAP_1-7</p>
Default ECRC Generation Capability	<p>Function: Default value for the ECRC Generation Capability bit in the Advanced Capabilities and Control Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 1</p> <p>Parameter Name: DEFAULT_ECRC_GEN_CAP_N (DEFAULT_ECRC_GEN_CAP_0 through DEFAULT_ECRC_GEN_CAP_7)</p> <p>Device Type: All for DEFAULT_ECRC_GEN_CAP_0 DM / EP (not RC / SW) for DEFAULT_ECRC_GEN_CAP_1-7</p>
Advanced Error Interrupt Message Number	<p>Function: Default value for the Advanced Error Interrupt Message Number field of the Root Error Status Register.</p> <p>Value Range: 0x0–0x1F</p> <p>Default Value: 0x0</p> <p>Parameter Name: AER_INT_MSG_NUM_N</p> <p>Device Type: DM / RC (not EP / SW)</p>

Per PF Power Management Register Configuration

PME Support	<p>Function: Default value for the PME_Support field in the Power Management Capabilities Register.</p> <p>Value Range: 0x0–0x1F</p> <p>Default Value: 0x1B</p> <p>Parameter Name: PME_SUPPORT_N (PME_SUPPORT_0 through PME_SUPPORT_7)</p> <p>Device Type: All for PME_SUPPORT_0 DM / EP (not RC / SW) for PME_SUPPORT_1-7</p>
D1 Support	<p>Function: Default value for the D1 Support bit in the Power Management Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 1</p> <p>Parameter Name: D1_SUPPORT_N (D1_SUPPORT_0 through D1_SUPPORT_7)</p> <p>Device Type: All for D1_SUPPORT_0 DM / EP (not RC / SW) for D1_SUPPORT_1-7</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
D2 Support	<p>Function: Default value for the D2 Support bit in the Power Management Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: D2_SUPPORT_N (D2_SUPPORT_0 through D2_SUPPORT_7)</p> <p>Device Type: All for D2_SUPPORT_0 DM / EP (not RC / SW) for D2_SUPPORT_1-7</p>
Device Specific Initialization	<p>Function: Default value for the Device Specific Initialization (DSI) bit in the Power Management Capabilities Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEV_SPEC_INIT_N (DEV_SPEC_INIT_0 through DEV_SPEC_INIT_7)</p> <p>Device Type: All for DEV_SPEC_INIT_0 DM / EP (not RC / SW) for DEV_SPEC_INIT_7</p>
Auxiliary Current	<p>Function: Default value for the Aux Current field in the Power Management Capabilities Register.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 0 (0 mA) • 1 (55 mA) • 2 (100 mA) • 3 (160 mA) • 4 (220 mA) • 5 (270 mA) • 6 (320 mA) • 7 (375 mA) <p>Default Value: 7 (375 mA)</p> <p>Parameter Name: AUX_CURRENT_N (AUX_CURRENT_0 through AUX_CURRENT_7)</p> <p>Device Type: All for AUX_CURRENT_0 DM / EP (not RC / SW) for AUX_CURRENT_1-7</p>
No Reset on D3hot->D0 Transition	<p>Function: Default value for the No Soft Reset bit in the Power Management Control and Status Register.</p> <p>Value Range: 0 or 1</p> <p>Default Value: 0</p> <p>Parameter Name: DEFAULT_NO_SOFT_RESET_N (DEFAULT_NO_SOFT_RESET_0 through DEFAULT_NO_SOFT_RESET_7)</p> <p>Device Type: All for DEFAULT_NO_SOFT_RESET_0 DM / EP (not RC / SW) for DEFAULT_NO_SOFT_RESET_1-7</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Per PF PCI Register Configuration	
Device Identification Number	<p>Function: Default value for the Device ID field in the Device ID and Vendor ID Register.</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0xABCD</p> <p>Parameter Name: CX_DEVICE_ID_N (CX_DEVICE_ID_0 through CX_DEVICE_ID_7)</p> <p>Device Type: All for CX_DEVICE_ID_0 DM / EP (not RC / SW) for CX_DEVICE_ID_1-7</p>
Vendor Identification Number	<p>Function: Default value for the Vendor ID field in the Device ID and Vendor ID Register.</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x16C3</p> <p>Parameter Name: CX_VENDOR_ID_N (CX_VENDOR_ID_0 through CX_VENDOR_ID_7)</p> <p>Device Type: All for CX_VENDOR_ID_0 DM / EP (not RC / SW) for CX_VENDOR_ID_1-7</p>
Device Revision Number	<p>Function: Default value for the Revision ID field in the Revision ID Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x01</p> <p>Parameter Name: CX_REVISION_ID_N (CX_REVISION_ID_0 through CX_REVISION_ID_7)</p> <p>Device Type: All for CX_REVISION_ID_0 DM / EP (not RC / SW) for CX_REVISION_ID_1-7</p>
Subsystem Device ID	<p>Function: Default value for the Subsystem ID field in the Subsystem ID and Subsystem Vendor ID Register.</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x0000</p> <p>Parameter Name: SUBSYS_DEV_ID_N (SUBSYS_DEV_ID_0 through SUBSYS_DEV_ID_7)</p> <p>Device Type: DM / EP (not RC / SW)</p>
Subsystem Vendor ID	<p>Function: Default value for the Subsystem Vendor ID field in the Subsystem ID and Subsystem Vendor ID Register.</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x0000</p> <p>Parameter Name: SUBSYS_VENDOR_ID_N (SUBSYS_VENDOR_ID_0 through SUBSYS_VENDOR_ID_7)</p> <p>Device Type: DM / EP (not RC / SW)</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Base Class Code	<p>Function: Default value for the Base Class Code field in the Class Code Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x00</p> <p>Parameter Name: BASE_CLASS_CODE_N (BASE_CLASS_CODE_0 through BASE_CLASS_CODE_7)</p> <p>Device Type: All for BASE_CLASS_CODE_0 DM / EP (not RC / SW) for BASE_CLASS_CODE_1-7</p>
Sub Class Code	<p>Function: Default value for the Subclass Code field in the Class Code Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x00</p> <p>Parameter Name: SUB_CLASS_CODE_N (SUB_CLASS_CODE_0 through SUB_CLASS_CODE_7)</p> <p>Device Type: All for SUB_CLASS_CODE_0 DM / EP (not RC / SW) for SUB_CLASS_CODE_1-7</p>
Programming Interface Code	<p>Function: Default value for the Programming Interface field in the Class Code Register.</p> <p>Value Range: 0x0–0xFF</p> <p>Default Value: 0x00</p> <p>Parameter Name: IF_CODE_N (IF_CODE_0 through IF_CODE_7)</p> <p>Device Type: All for IF_CODE_0 DM / EP (not RC / SW) for IF_CODE_1-7</p>
CardBus CIS Address Pointer	<p>Function: Default value for the CardBus CIS Pointer Register.</p> <p>Value Range: 0x0–0xFFFF</p> <p>Default Value: 0x0000</p> <p>Parameter Name: CARDBUS_CIS_PTR_N (CARDBUS_CIS_PTR_0 through CARDBUS_CIS_PTR_7)</p> <p>Device Type: DM / EP (not RC / SW)</p>
Interrupt	<p>Function: Default value for the Interrupt Pin Register.</p> <p>Value Range: 0 (None), 1 (INTA), 2 (INTB), 3 (INTC), or 4 (INTD)</p> <p>Default Value: 1 (INTA)</p> <p>Parameter Name: INT_PIN_MAPPING_N (INT_PIN_MAPPING_0 through INT_PIN_MAPPING_7)</p> <p>Device Type: DM / EP (not RC / SW)</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
IO Address Decode	<p>Function: Default value for the 32-Bit I/O Space bit in the I/O Base and I/O Limit Register.</p> <p>Value Range: 0 (16-bit) or 1 (32-bit)</p> <p>Default Value: 0 (16-bit)</p> <p>Parameter Name: IO_DECODE_32_N (IO_DECODE_32_0 through IO_DECODE_32_7)</p> <p>Device Type: DM / RC (not EP and SW)</p>
Memory Address Decode	<p>Function: Default value for the 64-Bit Memory Addressing bit in the Prefetchable Memory Base and Limit Register.</p> <p>Value Range: 0 (32-bit) or 1 (64-bit)</p> <p>Default Value: 0 (32-bit)</p> <p>Parameter Name: MEM_DECODE_64_N (MEM_DECODE_64_0 through MEM_DECODE_64_7)</p> <p>Device Type: DM / RC (not EP and SW)</p>
Enable ROM BAR	<p>Function: Determines whether the Expansion ROM BAR is enabled by default. The Expansion ROM BAR can also be enabled or disabled at runtime by writing to bit 0 of the ROM BAR Mask register (if `ROM_MASK_WRITABLE_N = 1).</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): ROM BAR is enabled False (0): ROM BAR is disabled <p>Parameter Name: ROM_BAR_ENABLED_N (ROM_BAR_ENABLED_0 through ROM_BAR_ENABLED_7)</p> <p>Device Type: All for ROM_BAR_ENABLED_0 DM / EP (not RC / SW) for ROM_BAR_ENABLED_1-7</p>
ROM BAR Mask	<p>Function: Determines the default value of the Expansion ROM BAR Mask register. The BAR Mask register specifies which bits of the Expansion ROM BAR are non-writable by host software, which determines the size of the BAR.</p> <p>The maximum value for `ROM_MASK_N is 0xFFFF because the maximum space that can be claimed by an Expansion ROM BAR is 16 MB.</p> <p>See Section 4.1.6.12, “BAR Mask Registers” on page 350 for more information in EP mode.</p> <p>See Section 4.2.5.11, “BAR Mask Registers” on page 455 for more information in RC mode.</p> <p>Value Range: 0x0–0xFFFFFFFF</p> <p>Parameter Name: ROM_MASK_N (ROM_MASK_0 through ROM_MASK_7)</p> <p>Device Type: All for ROM_MASK_0 DM / EP (not RC / SW) for ROM_MASK_1-7</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Allow Reprogramming of ROM BAR Mask	<p>Function: Determines whether the Expansion ROM BAR Mask register is writable by application software. If writing to a Expansion ROM BAR Mask register is enabled, the application can write to the Expansion ROM BAR Mask register through the DBI by asserting dbi_cs2 in addition to dbi_cs.</p> <p>If `ROM_BAR_ENABLED_N = 0 and `ROM_MASK_WRITABLE_N = 0, then the Expansion ROM BAR is excluded from the core hardware configuration.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): ROM BAR mask is writable False (0): ROM BAR mask is not writable <p>Parameter Name: ROM_MASK_WRITABLE_N (ROM_MASK_WRITABLE_0 through ROM_MASK_WRITABLE_7)</p> <p>Device Type: All for ROM_MASK_WRITABLE_0 DM / EP (not RC / SW) for ROM_MASK_WRITABLE_1-7</p>
Specify ROM BAR Target Interface	<p>Function: Direct incoming Requests that pass filtering and match the Expansion ROM BAR to either RTRGT0 or RTRGT1.</p> <p>For example, setting `ROM_FUNC0_BAR0_TARGET_MAP to 1 maps all incoming Requests for the Expansion ROM BAR of function 0 to RTRGT1.</p> <p>If `TRGT1_POPULATE = 0 (no RTRGT1 interface), then the map-by-BAR parameters have no effect; all Requests that pass filtering are routed to RTRGT0.</p> <p>Value Range: 0 (Target_0) or 1 (Target_1)</p> <p>Parameter Name: ROM_FUNCN_TARGET_MAP (ROM_FUNC0_TARGET_MAP through ROM_FUNC7_TARGET_MAP)</p> <p>Device Type: All for ROM_FUNC0_TARGET_MAP DM / EP (not RC / SW) for ROM_FUNC1-7_TARGET_MAP</p>

Physical Function BAR_0-7

BAR Setup Parameters

Determine the configuration of the 6 available BARs (EP mode) or 2 available BARs (RC mode).

During initialization, application software can write to the BARs to modify the BAR setup. There is a set of BAR-related parameters for each function.

For each of the parameters listed below, the lower case n is the BAR number and the upper case N at the end of the parameter name is the function number (0-7). For example BAR0_TYPE_1 defines the BAR type for BAR 0 of function 1.

The default values are different for each BAR. You can view the default BAR setup in the Specify Configuration dialog of the coreConsultant GUI.

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Enable BAR_n	<p>Function: Determines whether the BAR is enabled by default. The BAR can also be enabled or disabled at runtime by writing to bit 0 of the BAR Mask register (if `BARn_MASK_WRITABLE_N = 1).</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): BAR is enabled False (0): BAR is disabled <p>Parameter Name: BARn_ENABLED_N (BAR0-5_ENABLED_0 through BAR0-5_ENABLED_7)</p> <p>Device Type: All for BAR0_ENABLED_0 and BAR1_ENABLED_0 DM / EP (not RC / SW) for BAR0_ENABLED_1-7, BAR1_ENABLED_1-7, and BAR2-5_ENABLED_N</p>
BAR_n is Memory or I/O	<p>Function: Determines whether the BAR is for memory or I/O (bit 0 of the BAR).</p> <p>Value Range: 0 (Memory) or 1 (I/O)</p> <p>Parameter Name: MEMn_SPACE_DECODER_N (MEM0-5_SPACE_DECODER_0 through MEM0-5_SPACE_DECODER_7)</p> <p>Device Type: All for MEM0_SPACE_DECODER_0 and MEM1_SPACE_DECODER_0 DM / EP (not RC / SW) for MEM0_SPACE_DECODER_1-7, MEM1_SPACE_DECODER_1-7, and MEM2-5_SPACE_DECODER_N</p>
BAR_n is Prefetchable	<p>Function: Determines whether a memory BAR is for prefetchable memory (bit 3 of a memory BAR).</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Memory is prefetchable False (0): Memory is not prefetchable <p>Parameter Name: PREFETCHABLEn_N (PREFETCHABLE0-5_0 through PREFETCHABLE0-5_7)</p> <p>Device Type: All for PREFETCHABLE0_0 and PREFETCHABLE1_0 DM / EP (not RC / SW) for PREFETCHABLE0_1-7, PREFETCHABLE1_1-7, and PREFETCHABLE2-5_N</p>
BAR_n Bit Size	<p>Function: Determines the type (bits [2:1] for a memory BAR).</p> <p>Value Range: 00 (32-bit) or 10 (64-bit)</p> <p>Parameter Name: BARn_TYPE_N (BAR0-5_TYPE_0 through BAR0-5_TYPE_7)</p> <p>Device Type: All for BAR0_TYPE_0 and BAR1_TYPE_0 DM / EP (not RC / SW) for BAR0_TYPE_1-7, BAR1_TYPE_1-7, and BAR2-5_TYPE_N</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Allow Reprogramming of BAR_n Mask	<p>Function: Determines whether the BAR Mask register is writable by application software. This allows the application to write the BAR Mask register through the DBI using the BAR's address and asserting dbi_cs2 in addition to dbi_cs.</p> <p>If `BARn_ENABLED_N = 0 and `BARn_MASK_WRITABLE_N = 0, then BAR n is excluded from the core hardware configuration.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Mask is writable False (0): Mask is not writable <p>Parameter Name: BARn_MASK_WRITABLE_N (BAR0-5_MASK_WRITABLE_N through BAR0-5_MASK_WRITABLE_N)</p> <p>Device Type: All for BAR0_MASK_WRITABLE_0 and BAR1_MASK_WRITABLE_0 DM / EP (not RC / SW) for BAR0_MASK_WRITABLE_1-7, BAR1_MASK_WRITABLE_1-7, and BAR2-5_MASK_WRITABLE_N</p>
BAR_n Mask	<p>Function: Determines the default value of the BAR Mask register. The BAR Mask register specifies which bits of the BAR are non-writable by host software, which determines the size of the BAR. See Section 4.1.6.12, “BAR Mask Registers” on page 350 for more information in EP mode.</p> <p>See Section 4.2.5.11, “BAR Mask Registers” on page 455 for more information in RC mode.</p> <p>Value Range: 0x0–0xFFFFFFFFFFFFFF</p> <p>Parameter Name: BARn_MASK_N (BAR0-5_MASK_0 through BAR0-5_MASK_7)</p> <p>Device Type: All for BAR0_MASK_0 and BAR1_MASK_0 DM / EP (not RC / SW) for BAR0_MASK_1-7, BAR1_MASK_1-7, and BAR2-5_MASK_N</p>

**TABLE 5-1 Function Configuration Parameters (Continued)**

coreConsultant Option Name	Description
Specify Target Interface for BAR_n	<p>Function: Direct incoming Requests that pass filtering and match BAR_n to either RTRGT0 or RTRGT1.</p> <p>For example, setting `MEM_FUNC0_BAR0_TARGET_MAP to 1 maps all incoming Requests for BAR0 of function 0 to RTRGT1.</p> <p>If `TRGT1_POPULATE = 0 (no RTRGT1 interface), then the map-by-BAR parameters have no effect; all Requests that pass filtering are routed to RTRGT0.</p> <p>Value Range: 0 (Target_0) or 1 (Target_1)</p> <p>Parameter Name: MEM_FUNCN_BARn_TARGET_MAP (MEM_FUNC0_BAR0-5_TARGET_MAP through MEM_FUNC7_BAR0-5_TARGET_MAP)</p> <p>Device Type: All for MEM_FUNC0_BAR0_TARGET_MAP and MEM_FUNC0_BAR1_TARGET_MAP DM / EP (not RC / SW) for MEM_FUNC0_BAR2-5_TARGET_MAP and MEM_FUNC1-7_BARn_TARGET_MAP</p>
Per PF SR-IOV Register Configuration	
Number of Virtual Functions	<p>Function: 16-bit number of virtual functions supported for this physical function (PF function number 0-7). Valid only if CX_SRIOV_ENABLE=1.</p> <p>Value Range: 0 - 4096</p> <p>Default Value: 1</p> <p>Parameter Name: CX_MAX_VF_N (CX_MAX_VF_0 through CX_MAX_VF_7)</p> <p>Device Type: EP / DM (EP mode)</p>
Virtual Function Dependency Link	<p>Function: Specifies the 8-bit VF dependency link for the PF. Valid only if CX_NFUNC>1 and CX_VF_DEPENDENCY_LINK_SUPP=1.</p> <p>Value Range: 0 through 0xFF</p> <p>Default Value: 0</p> <p>Parameter Name: CX_VF_DEPENDENCY_LINK_N (CX_VF_DEPENDENCY_LINK_0 through CX_VF_DEPENDENCY_LINK_7)</p> <p>Device Type: EP / DM (EP mode)</p>
Virtual Function Device Identification Number	<p>Function: Specifies the 16-bit VF device ID for this PF. The VF device ID may be different from the PF's device ID.</p> <p>Value Range: 0 through 0xFFFF</p> <p>Default Value: CX_DEVICE_ID_N</p> <p>Parameter Name: CX_VF_DEVICE_ID_N (CX_VF_DEVICE_ID_0 through CX_VF_DEVICE_ID_7)</p> <p>Device Type: EP / DM (EP mode)</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Virtual Function Supported Page Sizes	<p>Function: Specifies the 32-bit supported page sizes for VFs for the PFs.</p> <p>Value Range: 0x553 through 0xFFFF</p> <p>Default Value:</p> <p>Parameter Name: CX_VF_SUPP_PAGE_SIZE_0</p> <p>Device Type: EP / DM (EP mode)</p>
First VF Offset with ARI	<p>Function: Specifies the Routing ID offset of the first VF in this PF for an ARI Capable hierarchy.</p> <p>Value Range: 0 through 0xFFFF</p> <p>Default Value: CX_NFUNC</p> <p>Parameter Name: FIRST_VF_OFFSET_ARI_CAP_HIER1_N (FIRST_VF_OFFSET_ARI_CAP_HIER1_0 through FIRST_VF_OFFSET_ARI_CAP_HIER1_7)</p> <p>Device Type: EP / DM (EP mode)</p>
First VF Offset without ARI	<p>Function: Specifies the Routing ID offset of the first VF in this PF for a non-ARI Capable hierarchy.</p> <p>Value Range: 0 through 0xFFFF</p> <p>Default Value: 0x0100</p> <p>Parameter Name: FIRST_VF_OFFSET_ARI_CAP_HIER0_N (FIRST_VF_OFFSET_ARI_CAP_HIER0_0 through FIRST_VF_OFFSET_ARI_CAP_HIER0_7)</p> <p>Device Type: EP / DM (EP mode)</p>
VF Stride with ARI	<p>Function: Specifies the Routing ID stride from one VF to the next one in this PF for an ARI Capable hierarchy.</p> <p>Value Range: 0 through 0xFFFF</p> <p>Default Value: 'CX_NFUNC</p> <p>Parameter Name: VF_STRIDE_ARI_CAP_HIER1_N (VF_STRIDE_ARI_CAP_HIER1_0 through VF_STRIDE_ARI_CAP_HIER1_7)</p> <p>Device Type: EP / DM (EP mode)</p>
VF Stride without ARI	<p>Function: Specifies the Routing ID stride from one VF to the next one in this PF for a non-ARI Capable hierarchy.</p> <p>Value Range: 0 through 0xFFFF</p> <p>Default Value: 0x0100</p> <p>Parameter Name: VF_STRIDE_ARI_CAP_HIER0_N (VF_STRIDE_ARI_CAP_HIER0_0 through VF_STRIDE_ARI_CAP_HIER0_7)</p> <p>Device Type: EP / DM (EP mode)</p>

Per-PF Virtual Function BAR_0-7

**TABLE 5-1 Function Configuration Parameters (Continued)**

coreConsultant Option Name	Description
VF BAR Setup Parameters	
Determine the configuration of the 6 available BARs. These parameters are valid only if VF_BARn_ENABLED_N=1.	
During initialization, application software can write to the BARs to modify the BAR setup. There is a set of BAR-related parameters for each function.	
For each of the parameters listed below, the lower case n is the BAR number and the upper case N at the end of the parameter name is the function number (0–7). For example BAR0_TYPE_1 defines the BAR type for BAR 0 of function 1.	
The default values are different for each BAR. You can view the default BAR setup in the Specify Configuration dialog of the coreConsultant GUI.	
Enable VF BAR_n	<p>Function: Determines whether the VF BAR is enabled by default. The VF BAR can also be enabled or disabled at runtime by writing to bit 0 of the BAR Mask register (if 'VF_BARn_MASK_WRITABLE_N = 1).</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): BAR is enabled False (0): BAR is disabled <p>Default Value:</p> <p>Parameter Name: VF_BARn_ENABLED_N (VF_BAR0-5_ENABLED_0 through VF_BAR0-5_ENABLED_7)</p> <p>Device Type: EP / DM (EP mode)</p>
VF BAR_n is Memory	<p>Function: Determines whether the VF BAR is for memory or I/O (bit 0 of the BAR).</p> <p>Value Range: 0 (Memory) or 1 (I/O)</p> <p>Default Value: 0</p> <p>Parameter Name: VF_MEMn_SPACE_DECODER_N (VF_MEM0-5_SPACE_DECODER_0 through VF_MEM0-5_SPACE_DECODER_7)</p> <p>Device Type: EP / DM (EP mode)</p>
VF BAR_n is Prefetchable	<p>Function: Determines whether a memory VF BAR is for prefetchable memory (bit 3 of a memory BAR). Valid only if VF_MEMn_SPACE_DECODER_N = 0.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Memory is prefetchable False (0): Memory is not prefetchable <p>Default Value:</p> <p>Parameter Name: VF_PREFETCHABLEn_N (VF_PREFETCHABLE0-5_0 through VF_PREFETCHABLE0-5_7)</p> <p>Device Type: EP / DM (EP mode)</p>



TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
VF BAR_n Bit Size	<p>Function: Determines the type (bits [2:1] for a memory VF BAR).</p> <p>Value Range: 00 (32-bit) or 10 (64-bit)</p> <p>Default Value:</p> <p>Parameter Name: VF_BARn_TYPE_N (VF_BAR0-5_TYPE_0 through VF_BAR0-5_TYPE_7)</p> <p>Device Type: EP / DM (EP mode)</p>
Allow Reprogramming of VF BAR_n Mask	<p>Function: Determines whether the VF BAR Mask register is writable by application software. This allows the application to write the BAR Mask register through the DBI using the VF BAR's address and asserting dbi_cs2 in addition to dbi_cs.</p> <p>If `VF_BARn_ENABLED_N = 0 and `VF_BARn_MASK_WRITABLE_N = 0, then BAR n is excluded from the core hardware configuration.</p> <p>Value Range:</p> <ul style="list-style-type: none"> True (1): Mask is writable False (0): Mask is not writable <p>Default Value:</p> <p>Parameter Name: VF_BARn_MASK_WRITABLE_N (VF_BAR0-5_MASK_WRITABLE_N through VF_BAR0-5_MASK_WRITABLE_N)</p> <p>Device Type: EP / DM (EP mode)</p>
VF BAR_n Mask	<p>Function: Determines the default value of the VF BAR Mask register. The VF BAR Mask register specifies which bits of the VF BAR are non-writable by host software, which determines the size of the BAR.</p> <p>See Section 4.1.6.12, “BAR Mask Registers” on page 350 for more information in EP mode.</p> <p>Value Range: 0xFF–0xFFFFFFFFFFFFFF</p> <p>Default Value:</p> <p>Parameter Name: VF_BARn_MASK_N (VF_BAR0-5_MASK_0 through VF_BAR0-5_MASK_7)</p> <p>Device Type: EP / DM (EP mode)</p>

TABLE 5-1 Function Configuration Parameters (Continued)

coreConsultant Option Name	Description
Specify Target Interface for VF BAR_n	<p>Function: Direct incoming Requests that pass filtering and match VF BAR_n to either RTRGT0 or RTRGT1.</p> <p>For example, setting `VF_MEM_FUNC0_BAR0_TARGET_MAP to 1 maps all incoming Requests for VF BAR0 of function 0 to RTRGT1.</p> <p>If `TRGT1_POPULATE = 0 (no RTRGT1 interface), then the map-by-BAR parameters have no effect; all Requests that pass filtering are routed to RTRGT0.</p> <p>Value Range: 0 (Target_0) or 1 (Target_1)</p> <p>Default Value:</p> <p>Parameter Name: VF_MEM_FUNCN_BARn_TARGET_MAP (VF_MEM_FUNC0_BAR0-5_TARGET_MAP through VF_MEM_FUNC7_BAR0-5_TARGET_MAP)</p> <p>Device Type: EP / DM (EP mode)</p>

5.1.8 RAM Configuration Parameters

TABLE 5-1 lists the RAM configuration parameters.

TABLE 5-1 RAM Configuration Parameters

coreConsultant Option Name	Description
Inhibit Simultaneous RAM Access in a 2-Port RAM	<p>Function: Inhibits the RAM's read enable or modifies the read address when the read and write addresses are equal. Turning this option off will improve timing, but may not be supported by some 2-port RAM implementations.</p> <p>NOTE: The core only requires that the write data be written to the RAM in this situation. The read data is not used and can be Xs.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 1: inhibits • 0: does not inhibit <p>Default Value: 0: does not inhibit</p> <p>Parameter Name: CX_RAM_ADDR_COMP</p> <p>Device Type: All</p>
Use External RAMs	<p>Function: Determines whether RAMs are instantiated internally or RAM interfaces are instantiated as top-level interfaces for connection of external RAM modules.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 1: RAMs are external • 0: RAMs are internal <p>Default Value: 1: RAMs are external</p> <p>Parameter Name: CX_RAM_AT_TOP_IF</p> <p>Device Type: All</p>
RAM Type	<p>Function: Selects the type of RAM models to be used.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • Simple (0) • DesignWare (1) • FPGA RAM (2) <p>Default Value: Simple, unless FPGA then value is FPGA RAM</p> <p>Parameter Name: CX_RAM_TYPE</p> <p>Device Type: All</p>
RAM Data Error Protection Config	<p>Function: Selects the RAM protection mode: parity, error checking, correction (ECC), or none. If ECC is selected, the RAMs must be internal to the core.</p> <p>Value Range: 0 (None), 1 (Parity), or 2 (ECC)</p> <p>Default Value: 0 (None)</p> <p>Parameter Name: CX_RAM_PROTECTION_MODE</p> <p>Device Type: All</p>

TABLE 5-1 RAM Configuration Parameters

coreConsultant Option Name	Description
Parity Config	<p>Function: Number of bits over which one parity bit is calculated and checked.</p> <p>Value Range: 0 (None), 32 (32-bit), 16 (16-bit), or 8 (8-bit)</p> <p>Default Value: 0 (None)</p> <p>Parameter Name: CX_PAR_MODE</p> <p>Device Type: All</p>
RAM Timing Model	<p>Function: Specifies whether to use black box timing or physical RAM timing model. If black box timing is specified, black box RAMs will be used for synthesis, and timing constraints for RAM interfaces will be derived from the RAM*P_RD_ACCESS and RAM*P_ADDR_SU parameters. If physical RAM timing is specified, the timing will be specified by your physical RAM model.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 1: black box timing • 0: use physical RAM timing model <p>Default Value: 0: use physical RAM timing model</p> <p>Parameter Name: RAM_TIMING_MODEL</p> <p>Device Type: All</p>
Single port RAM Read Access Time [ps]	<p>Function: Specifies the single port RAM read access time. Used by the synthesis timing model.</p> <p>Value Range: 0-32 bit</p> <p>Default Value: 1249 ps</p> <p>Parameter Name: RAM1P_RD_ACCESS</p> <p>Device Type: All</p>
Single port RAM Address/Data Setup Time [ps]	<p>Function: Specifies the single port RAM data setup. Used by the synthesis timing model.</p> <p>Value Range: 0-32 bit</p> <p>Default Value: 893 ps</p> <p>Parameter Name: RAM1P_ADDR_SU</p> <p>Device Type: All</p>
Dual port RAM Read Access Time [ps]	<p>Function: Specifies the dual port RAM read access time. Used by the synthesis timing model.</p> <p>Value Range: 0-32 bit</p> <p>Default Value: 1444 ps</p> <p>Parameter Name: RAM2P_RD_ACCESS</p> <p>Device Type: All</p>

TABLE 5-1 RAM Configuration Parameters

coreConsultant Option Name	Description
Dual port RAM Read Address/Data Setup Time [ps]	Function: Specifies the dual port RAM data setup. Used by the synthesis timing model. Value Range: 0-32 bit Default Value: 794 ps Parameter Name: RAM2P_ADDR_SU Device Type: All



5.1.9 PHY Configuration Parameters

TABLE 5-1 lists the PHY configuration parameters.

The CX_COMM_NFTS field (bit 23:16) of the Ack Frequency register is writable only if parameters are selected as follows during configuration of the core. Otherwise, the CX_COMM_NFTS field will be hard coded to CX_COMM_NFTS.

```
CX_NFTS != CX_COMM_NFTS
DEFAULT_LOS_EXIT_LATENCY != DEFAULT_COMM_LOS_EXIT_LATENCY
DEFAULT_L1_EXIT_LATENCY != DEFAULT_COMM_L1_EXIT_LATENCY
```

TABLE 5-1 PHY Configuration Parameters

coreConsultant Option Name	Description
PHY Selection	<p>Function: Select the targeted PHY</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 1 (Xilinx V2P) • 2 (Xilinx V4) • 4 (Synopsys PHY) • 5 (NXP PXPipe) • 6 (Altera S2GX) • 7 (Custom PHY) • 8 (Generic PHY) <p>Default Value: 4 (Synopsys PHY)</p> <p>Parameter Name: PHY_TYPE</p> <p>Device Type: All</p> <p>To synthesize with the Synopsys PHY, the PHY_PATH and PHY_LIBNAME variables must be set to the PHY installation location, and the target technology library selection in the coreConsultant Synthesize activity must be the same for the core and the PHY.</p>
PHY Gen 2 Mode	<p>Function: Specifies PCI Express Core operation mode</p> <p>Value Range:</p> <ul style="list-style-type: none"> • Dynamic Frequency (0) • Dynamic Width (1) • Disabled (2) <p>Default Value: Disabled (2)</p> <p>Parameter Name: CX_PHY_GEN2_MODE</p> <p>Device Type: All</p>



TABLE 5-1 PHY Configuration Parameters

coreConsultant Option Name	Description
Base PHY Operating Frequency	<p>Function: Specifies the base operating frequency of the PHY</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 62.5MHz (Simulation only) (3) • 125MHz (2) • 250MHz (1) <p>Default Value: CX_FREQ</p> <p>Parameter Name: CX_PHY_FREQ</p> <p>Device Type: All</p>
External PHY Enabled at Chip Top	<p>Function: Enables the PIPE interface to be activated at PCIe device chip top. This is not a DWC PCIe core related parameter. It is for PCIe device instantiation. This enables a PIPE interface to be exposed at the chip level.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 1: Use External PHY • 0: Use Internal PHY <p>Default Value: 0</p> <p>Parameter Name: EXTERNAL_PHY_ENABLE</p> <p>Device Type: All</p>
Number of Fast Training Sequences (NFTS)	<p>Function: Specifies the number of Fast Training Sequences the core advertises during link training. This is used to inform the link partner of the cores ability to recover synchronization after a low power state. This number should come from the SerDes vendor. This parameter can be changed only when a PHY other than Synopsys PHY is selected.</p> <p>Value Range: 1-255</p> <p>Default Value: Value is set automatically unless Custom PHY or Generic PHY is used.</p> <p>Parameter Name: CX_NFTS</p> <p>Device Type: All</p>
Number of Fast Training Sequences (NFTS) - Common Clock	<p>Function: Specifies the number of Fast Training Sequences (NTFS) the core advertises during link training, when common clock configuration is set. This parameter can be changed only when a PHY other than Synopsys PHY is selected.</p> <p>Value Range: 1-255</p> <p>Default Value: Value is set automatically unless Custom PHY or Generic PHY is used.</p> <p>Parameter Name: CX_COMM_NFTS</p> <p>Device Type: All</p>

TABLE 5-1 **PHY Configuration Parameters**

coreConsultant Option Name	Description
Number of Fast Training Sequences (NTFS) - Gen2	<p>Function: Specifies the number of Fast Training Sequences (NTFS) the core advertises during link training.</p> <p>Value Range: 1-255</p> <p>Default Value: 15</p> <p>Parameter Name: DEFAULT_GEN2_N_FTS</p> <p>Device Type: All</p>
PHY Tx Delay	<p>Function: Transmitter delay (PHY) in clock cycles. This parameter can be changed only when a PHY other than Synopsys PHY is selected.</p> <p>The value of this parameter should be specified by your PHY provider and is the (worst case) delay in PIPE clock cycles from the PIPE interface to the serial pins. This delay (along with PHY Rx Delay) is used to calculate expected round-trip delays and to size the buffers in the core.</p> <p>Value Range: 0-500</p> <p>Default Value: Value is set automatically unless Custom or Generic phy is used.</p> <p>Parameter Name: CX_PHY_TX_DELAY_PHY</p> <p>Device Type: All</p>
PHY Rx Delay	<p>Function: Receiver delay (PHY) in clock cycles. This parameter can be changed only when a PHY other than Synopsys PHY is selected.</p> <p>The value of this parameter should be specified by your PHY provider and is the (worst case) delay in PIPE clock cycles from the PIPE interface to the serial pins. This delay (along with PHY Tx Delay) is used to calculate expected round-trip delays and to size the buffers in the core.</p> <p>Value Range: 1-255</p> <p>Default Value: Value is set automatically unless Custom or Generic phy is used.</p> <p>Parameter Name: CX_PHY_RX_DELAY_PHY</p> <p>Device Type: All</p>
**MAC Tx Delay (not visible to coreConsultant)	<p>Function: Transmit delay through the PCS layer of the PHY in clock cycles, used for retry buffer auto-size calculation.</p> <p>Value Range: 0-500</p> <p>Default Value: 4</p> <p>Parameter Name: CX_PHY_TX_DELAY_MAC</p> <p>Device Type: All</p>

TABLE 5-1 PHY Configuration Parameters

coreConsultant Option Name	Description
**MAC Rx Delay (not visible in coreConsultant)	<p>Function: Receive delay through the PCS layer of the PHY in clock cycles, used for retry buffer auto-size calculation.</p> <p>Value Range: 0–500</p> <p>Default Value: 4</p> <p>Parameter Name: CX_PHY_RX_DELAY_MAC</p> <p>Device Type: All</p>
Selectable De-emphasis	<p>Function: Select the de-emphasis level.</p> <p>Value Range:</p> <ul style="list-style-type: none"> 0: -6 dB 1: -3.5 dB <p>Default Value: 0</p> <p>Parameter Name: SEL_DE_EMPHASIS</p> <p>Device Type: DM / RC / SW (not EP)</p>

5.1.10 Design Pipelining Configuration Parameters

TABLE 5-1 lists the design pipelining configuration parameters.

TABLE 5-1 Design Pipelining Configuration Parameters

coreConsultant Option Name	Description
Design Pipelining Granularity	<p>Function: This parameter controls the granularity of inter-module and other stages internal to the core to trade-off latency and gates for ease of timing closure.</p> <ul style="list-style-type: none"> • Coarse: CX_TECHNOLOGY provides broad stroke pipeline control. • Fine: Customization of each pipeline is independently controllable. <p>Note: This parameter applies to non-FPGA applications only</p> <p>Value Range: 0 (Coarse) or 1 (Fine)</p> <p>Default Value: 0 (Coarse)</p> <p>Parameter Name: CX_CUSTOM_PIPELINING</p> <p>Device Type: All</p>
Technology Speed	<p>Function: The speed of the target technology relative to the clock frequency and architecture. Possible values are SLOW or FAST. This option is used to enable additional pipeline stages internal to the core to trade off latency and gates for ease of timing closure. For example, an FPGA technology would normally be SLOW, even though it is operating at 125 MHz. An ASIC technology might be FAST if running at 125 MHz, but the SLOW setting might be needed to meet timing at 250 MHz in an x8/x16 architecture.</p> <p>Value Range: 0 (SLOW) or 2 (FAST)</p> <p>Default Value: 2 (FAST)</p> <p>Parameter Name: CX_TECHNOLOGY</p> <p>Device Type: All</p>
RAM ECC Pipeline Enable	<p>Function: Inserts an extra pipe for RAM ECC logic to improve timing.</p> <p>Value Range: True (1) or false (0)</p> <p>Default Value: False (0)</p> <ul style="list-style-type: none"> • True when CX_NB = 1 • False when CX_NB = 2 <p>Parameter Name: CX_ECC_PIPE_EN</p> <p>Device Type: All</p>

TABLE 5-1 Design Pipelining Configuration Parameters (Continued)

coreConsultant Option Name	Description
Transmit LCRC Latency Selection	<p>Function: Transmit LCRC pipeline latency value. This value represents the number of pipeline stages needed to calculate transmit LCRC.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 128 bit core supported latencies: 0, 3 • 64 bit core supported latencies: 0, 1 • 32 bit core supported latencies: 0, 1. <p>Default Value: 0</p> <p>Parameter Name: CX_XDLH_CRC_LATENCY</p> <p>Device Type: All</p>
Transmit ECRC Latency Selection	<p>Function: Transmit ECRC pipeline latency value. This value represents the number of pipeline stages needed to calculate transmit ECRC.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 128 bit core supported latencies: 0, 3 • 64 bit core supported latencies: 0 • 32 bit core supported latencies: 0 <p>Default Value: 0</p> <p>Parameter Name: CX_XTLH_CRC_LATENCY</p> <p>Device Type: All</p>
Receive LCRC Latency Selection	<p>Function: Receive LCRC pipeline latency value. This value represents the number of pipeline stages needed to calculate and compare received LCRC.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 128 bit core supported latencies: 0, 2 • 64 bit core supported latencies: 0, 1 • 32 bit core supported latencies: 0, 1 <p>Default Value: 0</p> <p>Parameter Name: CX_RDLH_CRC_LATENCY</p> <p>Device Type: All</p>
Receive ECRC Latency Selection	<p>Function: Receive ECRC pipeline latency value. This value represents the number of pipeline stages needed to calculate and compare receive ECRC.</p> <p>Value Range:</p> <ul style="list-style-type: none"> • 128 bit core supported latencies: 0, 3 • 64 bit core supported latencies: 0, 1 • 32 bit core supported latencies: 0, 1 <p>Default Value: 0 for 32/64 bit cores; 1 for 128 bit cores</p> <p>Parameter Name: CX_RTLH_CRC_LATENCY</p> <p>Device Type: All</p>

**TABLE 5-1 Design Pipelining Configuration Parameters (Continued)**

coreConsultant Option Name	Description
**Receive Checksum Pipeline Enable (not visible in coreConsultant)	<p>Function: Receive Checksum pipeline latency value. This value represents the number of pipeline stages used to calculate the RDLH check sum for DLLPs.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RDLH_PIPELINE_CHECKSUM</p> <p>Device Type: All</p>
Retry Buffer Pipeline Enable	<p>Function: Retry Buffer pipeline latency value. This value represents the number of pipeline stages used in retry buffer, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XDLH_PIPELINE_RBUF</p> <p>Device Type: All</p>
Transmit Flow Control Calculation Pipeline Enable	<p>Function: Transmit Flow Control Calculations pipeline enable, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XADM_FC_PIPELINE</p> <p>Device Type: All</p>
Receive MAC Layer Packet Finder Input Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the inputs of RMLH Sequence finder, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RMLH_PKT_FINDER_REGIN</p> <p>Device Type: All</p>

TABLE 5-1 Design Pipelining Configuration Parameters (Continued)

coreConsultant Option Name	Description
Receive MAC Layer Sequence Finder Input Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the inputs of RMLH Sequence finder, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RMLH_SEQ_FINDER_REGIN</p> <p>Device Type: All</p>
Receive MAC Layer Scrambler Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RMLH Scrambler, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RMLH_SCRAMBLE_REGOUT</p> <p>Device Type: All</p>
Receive MAC Layer Deskew Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RMLH Deskew, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RMLH_DESKEW_REGOUT</p> <p>Device Type: All</p>
Receive MAC Layer PIPE Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RMLH PIPE, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RMLH_PIPE_REGOUT</p> <p>Device Type: All</p>

**TABLE 5-1 Design Pipelining Configuration Parameters (Continued)**

coreConsultant Option Name	Description
Receive Data Link Layer TLP Extract Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RDLH TLP Extract, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RDLH_TLP_EXTRACT_REGOUT</p> <p>Device Type: All</p>
Receive Transaction Layer TLP Check Input Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the inputs of RTLH TLP Check, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0, 1</p> <p>Default Value: 0</p> <p>Parameter Name: CX_RTLH_TLP_CHECK_REGIN</p> <p>Device Type: All</p>
Receive Application Target 1 Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RADM_TRGT1 outputs, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RADM_TRGT1_REGOUT</p> <p>Device Type: All</p>
Receive Application Completion Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of RADM_CPL outputs, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RADM_CPL_REGOUT</p> <p>Device Type: All</p>



TABLE 5-1 Design Pipelining Configuration Parameters (Continued)

coreConsultant Option Name	Description
Segment Buffer Input Queue Manager Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of SEGBUF RADM In Queue Manager outputs, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RADM_INQ_MGR_REGOUT</p> <p>Device Type: All</p>
Segment Buffer Order Manager Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of SEGBUF RADM Order Manager outputs, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RADM_ORDER_MGR_REGOUT</p> <p>Device Type: All</p>
Transmit Transaction Layer Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of XTLH, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XTLH_CTRL_REGOUT</p> <p>Device Type: All</p>
Transmit Data Link Layer Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of XDLH, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) or 64/128 bit cores Disabled (0) for 64/128 bit cores and 32 bit cores <p>Default Value: 0</p> <p>Parameter Name: CX_XDLH_TLP_REGOUT</p> <p>Device Type: All</p>

TABLE 5-1 Design Pipelining Configuration Parameters (Continued)

coreConsultant Option Name	Description
Transmit MAC Layer Scrambler Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of XMLH Scrambler, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XMLH_SCRAMBLE_REGOUT</p> <p>Device Type: All</p>
Transmit MAC Layer PIPE Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs of XMLH PIPE, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XMLH_PIPE_REGOUT</p> <p>Device Type: All</p>
Error Log Output Pipeline Enable	<p>Function: Specifies insertion of register pipeline at the outputs Error Log, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_ERROR_LOG_REGOUT</p> <p>Device Type: All</p>
Pipeline Enable for LTSSM	<p>Function: Adds selected pipelines in the LTSSM, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> Enabled (1) Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_XMLH_PIPELINE_LTSSM</p> <p>Device Type: All</p>

TABLE 5-1 Design Pipelining Configuration Parameters (Continued)

coreConsultant Option Name	Description
Pipeline Enable Between RX Filter and RX Queue	<p>Function: Specifies insertion of register pipeline between filter and queue, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_FLT_Q_REGOUT</p> <p>Device Type: All</p>
Pipeline Enable in the rid_to_pfvf	<p>Function: Specifies insertion of register pipeline for the rid_to_pfvf module output in the RADM_FILTER and RADM_CPL_LUT blocks, providing tradeoff of latency and gates for ease of timing closure.</p> <p>Value Range: 0 or 1</p> <ul style="list-style-type: none"> • Enabled (1) • Disabled (0) <p>Default Value: 0</p> <p>Parameter Name: CX_RID_REGOUT</p> <p>Device Type: All</p>

5.2 TP Parameter

The TP parameter is a simple delay value (1 by default) that is used to simulate a Clock-to-Q delay on all of the flops in the design. This prevents issues caused by delta delay or simulator differences in the processing events, which can cause incorrect simulation results. By simulating a unit delay for the flop delays, the 0-delay deltas that may be introduced by clock tree placeholders or unequal number of buffers on different clock trunks will not cause false 0-cycle paths. This ensures that transitions caused by a clock edge will not affect any registers in the same clock domain until the next clock edge.

A**Core Frequencies and Sizes**

The core should be configured based on desired bandwidth and frequency. [Table A-1](#) provides supported core frequencies and sizes. This information should help identify the core data bus selection and core frequency desired. PHY PIPE interface frequency and the number of lanes will determine the core data bus size.

Table A-1 Supported Core Frequencies and Sizes

Core Databus Width	Core Frequency				
	x1 core frequency	x2 core frequency	x4 core frequency	x8 core frequency	x16 core frequency
GEN1 supported core frequency when PIPE is running at 125MHz					
32bits	62.5MHz 125MHz	125MHz	NA	NA	NA
64bits	NA	62.5MHz 125MHz	125MHz	NA	NA
128bits	Supported, but non Min	Supported, but non Min	62.5MHz 125MHz	125MHz	NA
GEN1 supported core frequency when PIPE is running at 250MHz					
32bits	62.5MHz 125MHz 250MHz	125MHz 250MHz	250MHz	NA	NA
64bits	Supported, but non Min	62.5MHz 125MHz 250MHz	125MHz 250MHz	250MHz	NA
128bits	Supported, but non Min	Supported, but non Min	62.5MHz 125MHz 250MHz	125MHz 250MHz	250MHz
GEN2 variable-width supported core frequency when PIPE is running at 125MHz in GEN1 mode					
32bits	125MHz	NA	NA	NA	NA

Table A-1 Supported Core Frequencies and Sizes

		Core Frequency				
Core Databus Width		x1 core frequency	x2 core frequency	x4 core frequency	x8 core frequency	x16 core frequency
64bits		NA	125MHz	NA	NA	NA
128bits		Supported, but non Min	62.5MHz 125MHz	125MHz	NA	NA
GEN2 variable-width supported core frequency when PIPE is running at 250MHz in GEN1 mode						
32bits		125MHz 250MHz	250MHz	NA	NA	NA
64bits		NA	125MHz 250MHz	250MHz	NA	NA
128bits		NA	NA	125MHz 250MHz	250MHz	NA
GEN2 variable-frequency supported core frequency						
32bits		125/250MHz	125/250MHz	125/250MHz	NA	NA
64bits		NA	125/250MHz	125/250MHz	125/250MHz	NA
128bits		NA	NA	125/250MHz	125/250MHz	125/250MHz
GEN2 variable-frequency supported core frequency when PIPE is running at 250MHz in GEN1 mode						
32bits		125/250MHz 250/500MHz	125/250MHz 250/500MHz	125/250MHz 250/500MHz	NA	NA
64bits		NA	125/250MHz 250/500MHz	125/250MHz 250/500MHz	125/250MHz 250/500MHz	NA
128bits		NA	NA	125/250MHz 250/500MHz	125/250MHz 250/500MHz	125/250MHz 250/500MHz

B

AHB Bridge Module

B.1 Product Overview

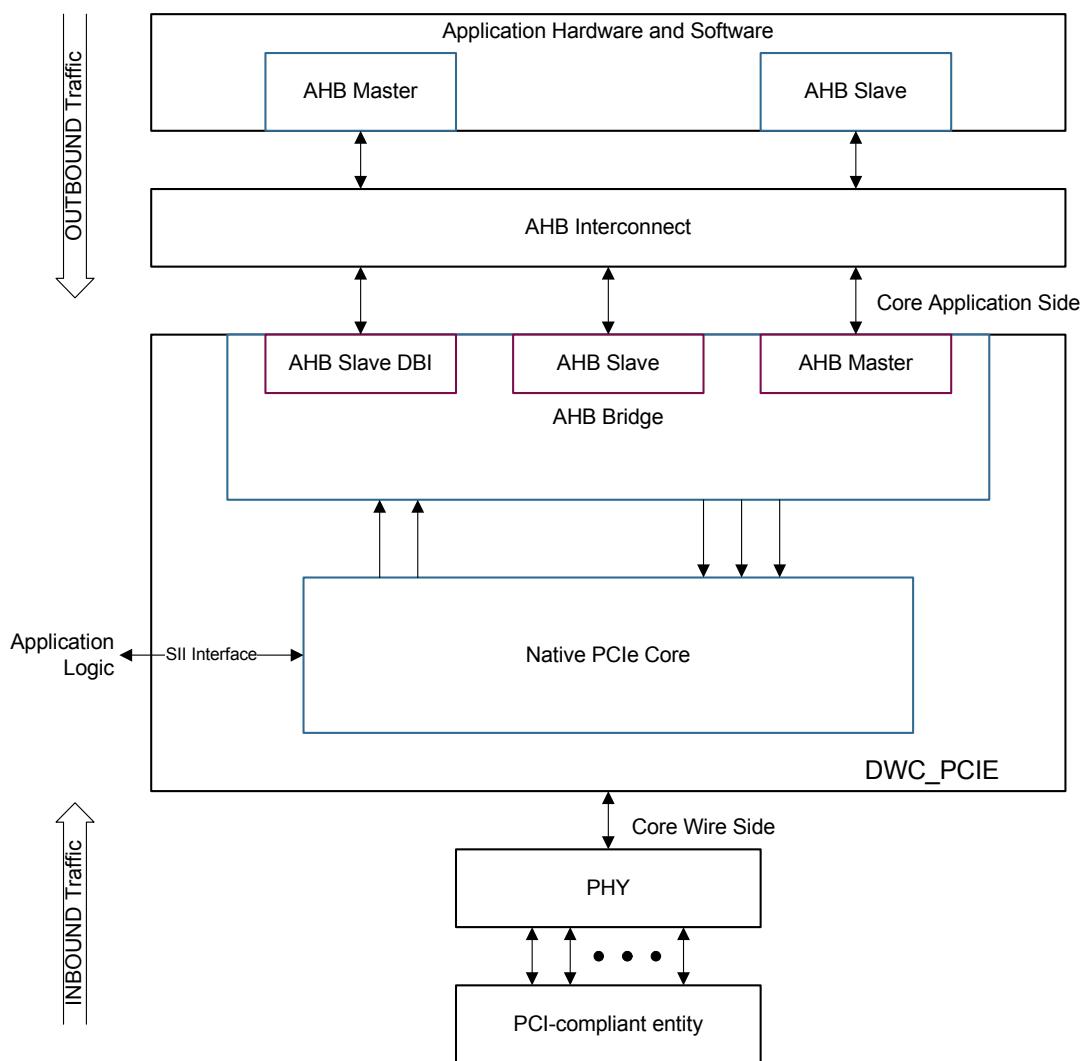
B.1.1 General Product Description

The Synopsys DWC PCI Express solution provides an optional AHB Bridging capability for directly adding a PCI Express link to an AHB system fabric. This can significantly reduce the time to design PCI Express into an AXI-based SOC. Refer to [Figure B-1](#) for a system-level view of the DWC PCIe AHB core and [Table B-1](#) for a definition of terms and acronyms.

The AHB Bridge Module acts as a bridge between the standard AXI interfaces and the Synopsys DesignWare PCIe core native interfaces. The bridge interconnects the AHB interfaces within an AMBA-embedded system with a remote PCIe link, as either a root complex port, or as an endpoint port. The bridge supports up to three AHB interfaces, one for an AHB master, one for an AHB slave, and one for DBI access to the native PCIe core.

The AHB master interface enables a remote PCIe device to read and write to an AHB slave connected to the AHB bridge. The AHB slave interface enables an AHB master to read and write through the AHB bridge to a remote PCIe device. The slave DBI interface enables an AHB master to read and write to registers inside the native PCIe core, or the device-specific registers attached to the PCIe native core's ELBI interface.

Throughout this document, the terms inbound and outbound are defined with respect to the AHB fabric. That is, inbound transactions are defined as the transactions presented by the native PCIe core's AHB master interface. Outbound transactions are defined as the transactions generated by an AHB master that target a remote PCIe device.

Figure B-1 DWC PCIe AHB Core, System-Level View**Table B-1** Definitions of Terms and Acronyms

Term or Acronym	Definition
Core	Identifies the entire core. The core includes the native PCIe-core and the AHB bridge.
Native PCIe core	Identifies the basic DesignWare library PCIe core which has its own non-standard, proprietary dedicated bus interface to the application.
Bridge AHB slave	The AHB slave interface on the core
Bridge AHB master	The AHB master interface on the core
Bridge DBI AHB slave	The AHB slave interface dedicated to the internal DBI (Data Bus Interface). This interface is used to access the internal registers of the core (CDM).

Table B-1 Definitions of Terms and Acronyms (Continued)

Term or Acronym	Definition
DBI	<p>Data Bus Interface</p> <p>This bus is internal to the native core. It is used to access the core's registers and external application's registers. The DBI-AHB slave interfaces provides the direct access to this internal bus.</p>
Inbound traffic	PCIe transactions that enter the native core from the wire side of the core (PCIe wire). These transactions are passed on to the AHB bridge where they will be delivered to the application side.
Outbound traffic	AHB transactions that enter the native core from the application side of the core. These AHB transactions are passed to the native core, where they will be sent out onto the PCIe wire.
MTU	<p>Maximum Transfer Unit</p> <p>Specifies the maximum packet payload size supported. This indicates the maximum allowed transfer size for a write or completion.</p>
Page boundary	Specifies the address page boundary size supported by the bridge. No packet can have an address that crosses the specified address boundary.
CDM	<p>Configuration Dependent Module</p> <p>This is an internal block in the native core that houses the PCI configuration registers and some user-accessible registers that reside in the core. In general, an application may use our core, but will add other registers that are unique to their applications. Those new application registers will be referred to as External Application Registers (EAR).</p>
LBC	<p>Local Bus Controller</p> <p>This is an internal block that resides in the native core. It allows the DBI interface (from the application side) or the wire side interface (via the radm_TRGT0 interface) to access the core's Configuration Dependent Module (CDM) registers and/or the External Application Registers (EAR). (For additional details on this module, please refer to the native core's documentation.)</p>
ELBI	<p>External Local Bus Interface</p> <p>This is an interface on the native core that processes read/write accesses to the external application registers (EAR). For applications that require external registers, the application can access the EAR through the bridge or an entirely different interface (outside the scope of this core). (For additional details on this interface, please refer to the native core's documentation.)</p>
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.

B.1.2 System Overview

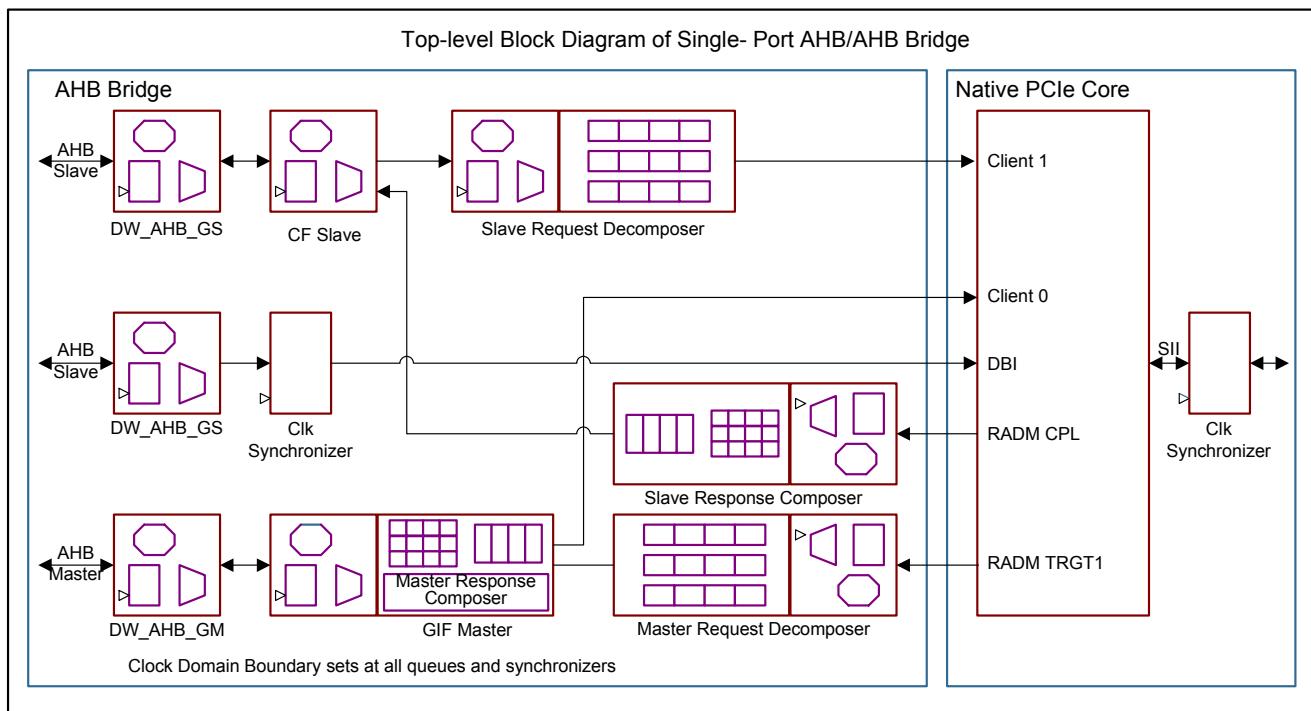
B.1.2.1 Block Diagram

The AHB Bridge Module (see [Figure B-2](#)) bridges the DesignWare PCI Express IP's native application interface and the AHB interconnect. It enables a remote PCIe device to be either an AHB slave, or an AHB master.

This module contains AHB master and slave protocol handlers, internal slave and master control for generic request and response interfaces, a packet composer, and a packet decomposer for response formation.

The slave and master protocol handlers support the AHB protocol conversion between an AHB transfer and a generic transfer within the bridge. The slave and master generic interface supports the conversion of an AHB transfer to a PCIe transaction. The packet composer and decomposer support the segmentation and reassembly of a PCIe transaction.

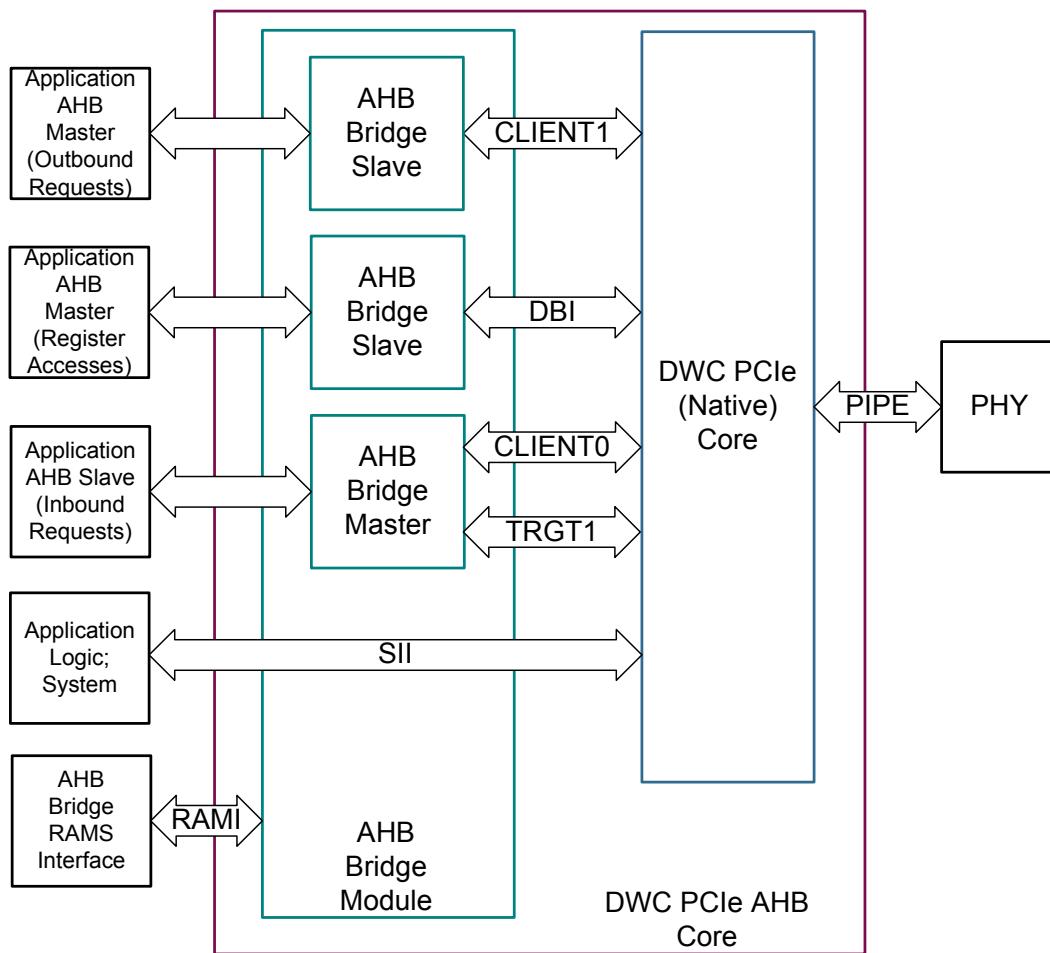
Figure B-2 AHB Bridge Module, Block Diagram



B.1.2.2 Interfaces

Figure B-3 shows the DWC PCIe AHB core top-level interfaces.

Figure B-3 DWC PCIe AHB Core Top-Level Interfaces



B.1.3 Features List

The AHB Bridge Module supports the following features. Please refer to the respective native PCIe core's user manual for a complete list of all PCIe core features.

General Features:

- ❖ AHB Master and Slave interfaces for inbound and outbound PCI Express requests
- ❖ All types of PCI Express transactions supported through the AHB Bridge¹
- ❖ 32-bit PCI Express core data bus support only
- ❖ 32-bit AHB master address and data bus support only
- ❖ Independent maximum read request of PCIe from burst transfer size of AHB bus
- ❖ Endianness support
- ❖ Multi-function support
- ❖ Common AHB clock for AHB master bus, AHB slave bus, and AHB slave DBI bus
- ❖ Independent native PCIe core clock support from AHB clock

Features supported as a bus master:

- ❖ INCR burst type support only as an AHB master
- ❖ Byte alignment support as an AHB master
- ❖ All response types support as an AHB master
- ❖ Early termination support as an AHB master
- ❖ Legacy PCIe Interrupt and MSI support
- ❖ Programmable buffer sizes for all queues that support in bound transfers

Features supported as a bus slave:

- ❖ AHB Slave read/write for CDM register access or application specific register access through ELBI
- ❖ 32-bit AHB read/write to CDM register or ELBI register only
- ❖ Programmable buffer sizes for all queues that support outbound transfers
- ❖ All burst type support as an AHB slave
- ❖ Byte alignment support as an AHB slave
- ❖ Okay, ERROR, retry and Split responses support as an AHB slave
- ❖ Sideband signals support for all attributes of PCIe completion
- ❖ Early termination is supported as an AHB slave
- ❖ Legacy PCIe Interrupt and MSI support
- ❖ PCIe completion timeout support
- ❖ In-order outbound service support only

**Note**

¹ Except the PCIe Message type without payload, due to the AHB bus always requiring data payload for any write. Configuration and IO outbound transfer must be within one beat.

SR-IOV support over the AXI bridge is not yet provided.

B.1.4 Feature Limitations

There are many configurable parameters that the DWC PCIe native core and the AHB bridge support. [Table B-2](#) identifies the limitations encountered when using the AHB bridge with the DWC PCIe native core.

Table B-2 Limitations

AHB Bridge Limitation	Module	Requirement
Page boundary must be greater than or equal to the maximum payload size.	The Decomposer block performs page boundary splitting and requires page boundary to be larger than or equal to the MTU.	For the AHB slave, we only support a 4K page boundary for outbound transfers. For the AHB master, the page boundary is programmable with one limitation; the page boundary must be larger than the master MTU size.
No support for AHB lock transfer.	N/A	AHB lock transfer is not currently supported in the AHB bridge.
No support for AHB exclusive transfer.	N/A	AHB lock transfer is not currently supported in the AHB bridge
PCIe Advanced Error Reporting (AER) is not supported in the AHB bridge.	N/A	AER is only supported with respect to the native PCIe core. Errors detected by the AHB bridge are not reported as part of AER.
Protection Control Data Access Mode only.	N/A	The AHB bridge only supports the Protection Control Data Access Mode.
PCIe read/write request with holes converted into sequential continuous access.	N/A	If an inbound request contains holes in its read and write, the AHB bridge converts this request into sequential master requests.
PCIe completions of a split AHB read request in the outbound direction must be returned in-order.	Composer block performs only in-order reassembly.	If two completions for two reads split from an AHB transfer are returned out-of-order from a remote PCIe device, then data will be corrupted at the composer of the AHB bridge.
Users application AHB master returning from SPLIT AHB transfer must not change burst type from undefined length INCR to fixed length INCR e.g. INCR4.	Bridge Slave.	See Section B.2.5, “SPLIT Response Mode,”

B.1.5 Clocks and Resets

The AHB Bridge Module employs two clock domains. The AHB master, slave, and DBI slave channels use hclk. The native PCIe core uses core_clk. Each of these clock signals has a corresponding reset signal. We recommend that the application provides the core's reset and AHB reset, and the application logic under the same reset condition. This is due to the lack of a graceful reset protocol built-in between the AHB bridge and the native PCIe core. For example, if the bridge is reset, transactions within the bridge may never get a response such that it may hang the AHB interface.

The reset signal is asynchronous with a synchronous deassertion, relative to its corresponding clock. Only one reset cycle is required to reset the logic associated with the corresponding clock.

The modules within the AHB bridge are designed to be in either the hclk domain or the core_clk domain, or both. The logic or FIFO to cross the clock domain are supported within the AHB bridge. The application can configure hclk and core_clk to be at a different clock domain, or the same clock domain.

B.1.6 Supported Core Configurations

The AHB Bridge should be configured based on desired core bandwidth and frequency. [Table B-3](#) provides supported core configurations for the AHB Bridge. All supported lanes, frequencies, and databus widths for the DWC PCIe core are also supported for the AHB Bridge. Refer to the DesignWare Cores PCI Express Reference Manual, Appendix A, for the supported core frequencies and sizes.

Table B-3 Core Configurations Supported by AHB Bridge

	Core Databus Width		
	32bits DWC PCIe core	64bit DWC PCIe core	128bit DWC PCIe core
AHB Slave	32bit databus	not supported	not supported
AHB Master	32bit databus	not supported	not supported
AHB DBI Slave	32bit databus	not supported	not supported
AHB Shared DBI	supported	not supported	supported

B.1.6.1 AHB Bridge Limitations

The AHB Bridge has the following limitations with regard to core configurations:

- ❖ INCR only supported as an AHB master
- ❖ All Burst type supported as an AHB slave
- ❖ No holes supported during a burst

B.2 PCIe AHB Core Operations

B.2.1 Supported AHB Transfer Type

The AHB Bridge Module is compliant with the AMBA 2.0 AHB specification.

- ❖ For outbound transfers (accessing bridge SLAVE):
All transfer types (IDLE, BUSY, NONSEQ, SEQ) are supported.
- ❖ For inbound transfers (using bridge MASTER):
All transfer types (IDLE, BUSY, NONSEQ, SEQ) can be issued to AHB slave.

B.2.2 Supported AHB Burst Operations

- ❖ For outbound transfers (accessing bridge SLAVE):
All burst types are currently supported with one special condition. INCR is an undefined length burst type, but a PCIe transaction must have a fixed length in its header. Therefore, a prefetch mechanism is implemented. A write with an INCR is supported without prefetch. Only a read with an INCR type is supported with a prefetch equal to the maximum configured AHB slave MTU. The extra prefetched data on a per-transfer basis will be discarded once the desired amount response has been returned to an AHB master.
- ❖ For inbound transfers (using bridge MASTER):
SINGLE and INCR burst type are the only supported master burst types. All received requests that are greater than 4 bytes, will be presented as INCR, with an undefined burst length. Otherwise the request is presented as a SINGLE.

Since the PCIe MTU is greater or equal to 128 bytes, a PCIe request with a full MTU worth can be presented onto the AHB bus as an AHB master.

The application can configure the maximum request size that their slaves can take. If the slave's maximum request size is smaller than the PCIe MTU, then the AHB bridge will split an inbound request into two or more requests. The responses of the split inbound request are required to be returned in-order. This requirement is normally satisfied because the split requests are supposed to target the same slave.

B.2.3 Supported AHB Transfer Size

The AHB bridge supports byte transfers.

- ❖ For outbound transfers (accessing bridge SLAVE):

Table B-4 Valid slv_haddr offset supported

slv_hsize	slv_haddr[1:0]
0	0, 1, 2, 3
1	0, 2
2	0

An HSIZE of one byte, a half word, and a word are supported. As an AHB slave, AHB bridge does not generate any memory transfers with holes to the PCIe core. All memory accesses are sequential accesses to continuous memory locations.

- ❖ For inbound transfers (using bridge MASTER):

An HSIZE of one byte, a half word and a word are supported. This means that an AHB master can issue a 1-byte, 2-byte, or 4-byte per cycle to the AHB bridge.

The AHB bridge supports inbound reads and writes with holes by assuming the transfer of non-prefetchable memory location. For example, if a read of the PCIe contains a first byte enable of 4'b0101, then the AHB bridge will issue a read request with an HSIZE of 0 (byte transfer) and a burst length of 2 as a master. The same applies for a write.

The first/last byte enable, address, and length of an inbound PCIe transaction will be converted into sequential memory reads and writes over the AHB master channel.

Protection control of data access is the only mode the AHB bridge supports.

B.2.4 Early Burst Termination (EBT) Support

The AHB bridge supports early termination for both the master and slave channels.

- ❖ For outbound transfers (accessing bridge SLAVE):

For a read, the AHB bridge always reads the whole transfer length of a defined AHB master. When a returned completion from the PCIe is received, the response of the exact burst length will be returned to the AHB slave interface under normal conditions. If an early termination occurs, then the response that is beyond the termination point will be discarded. More specifically, a read with a defined size burst will be sent to the PCIe link with an expected length of response. If an early termination occurs, then the rest of response data that the AHB bridge read will be discarded. If the AHB bridge is expecting the master to reconstruct the burst to obtain the rest of the data, the reconstructed burst will cause another PCIe read transaction over the PCIe link. There is a performance penalty for early termination. The application should try to minimize early terminations.

For a write, the AHB bridge takes the entire write burst into transmit FIFO unless an early termination occurs. In the early termination case, the data up to the terminated location will be transmitted onto the PCIe wire. More specifically, a write with a defined size burst will be sent to the PCIe link after all data has been enqueued into the AHB bridge's transmit queue. Therefore, if an early termination occurs, then a shorter write transaction will be sent to the PCIe link. It is expected that the master of the write request reconstruct the transfer after the early termination. The reconstructed transfer will be sent as another PCIe transaction. There is not much of the performance penalty other than one AHB request splitting into two or more PCIe transactions. Note, that when a write is issued during the data discard phase of the previous read, a write will be returned with a retry response. The data discard phase can never be a long-term stall.

- ❖ For inbound transfers (using bridge MASTER):

The AHB bridge as an AHB master is capable of supporting early termination. It will reconstruct the burst after the bus is granted again. This applies for both read and write.

B.2.5 SPLIT Response Mode

The AHB bridge supports SPLIT response mode for both the master and slave channels.

- ❖ For outbound transfers (accessing bridge SLAVE):

The bridge slave can be configured for SPLIT response mode (via the CC_RESPONSE_MODE parameter).

For a read, the AHB bridge slave always SPLIT's off the application master immediately. It supports one SPLIT request for each of the CC_SLV_NUM_MASTERS masters. When the bridge receives the read data from the remote device, it notifies the AHB arbiter via the HSPLIT bus. The relevant application master may now retrieve its read data by replaying the read request.



Note Any data that is not retrieved by the application master is dumped and not retained.

The dump process is triggered by the detection of NSEQ on the HTRANS signal in a bus clock cycle after the first one (i.e. detection of a new transaction) or IDLE. (Note that this is also the EBT detect condition.)

In other words, the application master can only use one transaction (SINGLE or burst) to retrieve its data. Multiple SINGLES or multiple small bursts (using INCR's) will not cause an error but the second transaction will cause the bridge to (1) dump all data that was not read by the first transaction (2) issue a new outbound MemRd request to the remote PCIe device and (3) split off the application master again.

When the application generates an undefined length burst read request (INCR) to the AHB bridge slave and is SPLIT off; it must retrieve data later on (after receiving split completion notification via HSPLIT signal) using an undefined length burst read request (INCR). It must not use a fixed length burst request e.g. INCR4. Otherwise data corruption and possible hanging of the bridge may occur.

However the reverse combination is allowed i.e. using undefined INCR to complete a SPLIT read request that used a defined length burst e.g. INCR4.

Since in the case of a read, the AHB bridge slave always SPLIT's off the application master immediately, it is reasonable to assume that the application master would not change its burst type upon replay.

Table B-5 SPLIT completion/replay burst types supported

Original Request	Replay Request (OK)	Replay Request (NOT ALLOWED)
INCR (undefined)	INCR (undefined)	¹ INCR4 8 16 SINGLE
INCR4	INCR4 INCR (undefined) SINGLE	INCR8 INCR16

Table B-5 SPLIT completion/replay burst types supported

Original Request	Replay Request (OK)	Replay Request (NOT ALLOWED)
INCR8	INCR4 8 INCR (undefined) SINGLE	INCR16
INCR16	INCR4 8 16 INCR (undefined) SINGLE	n/a
SINGLE	INCR (undefined) SINGLE	INCR4 8 16

1. This design restriction may change in a future release.



Note The application master should never attempt to retrieve (in a single burst) more data than it requested. This may cause data corruption or possible hanging of the bridge.

If an application master is EBT (Early Burst Terminated) while it is replaying/retrieving; then when it returns to retrieve the rest of its data it will be split off again.

❖ For inbound transfers (using bridge MASTER):

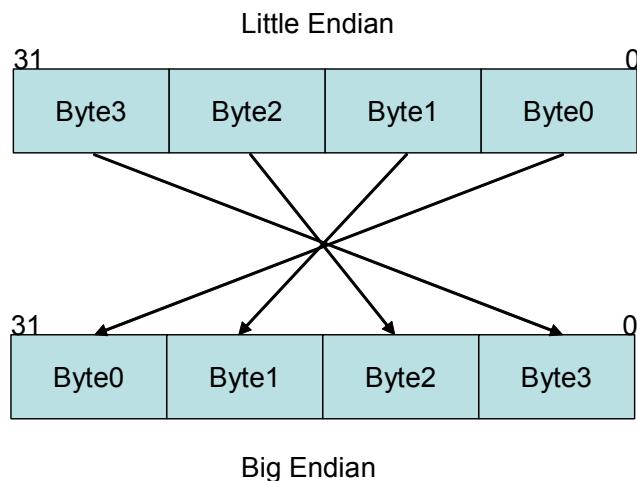
When the bridge master receives a SPLIT response from the AHB bus, it will request the bus again and wait until it is granted the bus by the AHB arbiter.

B.2.6 Endianess Support

The AHB bridge supports the endianess as defined by the AHB specification. There are two levels of controls regarding endianess. The application, at first, must select whether or not dynamic endianess control is desired. There is a parameter, AHB_ENDIANESS, that has three modes (dynamic, static little, and static big). If AHB_ENDIANESS is selected with dynamic mode, then input signals slv_big_endian, mstr_big_endian, and dbi_big_endian appear at the top-level of the AHB bridge, and are used to control big endian or little endian mode of operation. These are active high signals where 1'b1 indicates that the AHB bridge is in big endian operation. The default is little endian operation. The AHB_ENDIANESS parameter default setting is static little.

The big_endian signals are active only when AHB_ENDIANESS is set to dynamic mode. If the AHB_ENDIANESS is in static little mode, then the AHB bridge is hard-coded to only support little endian mode. Data bus byte ordering endianess conversion is shown in [Figure B-4](#).

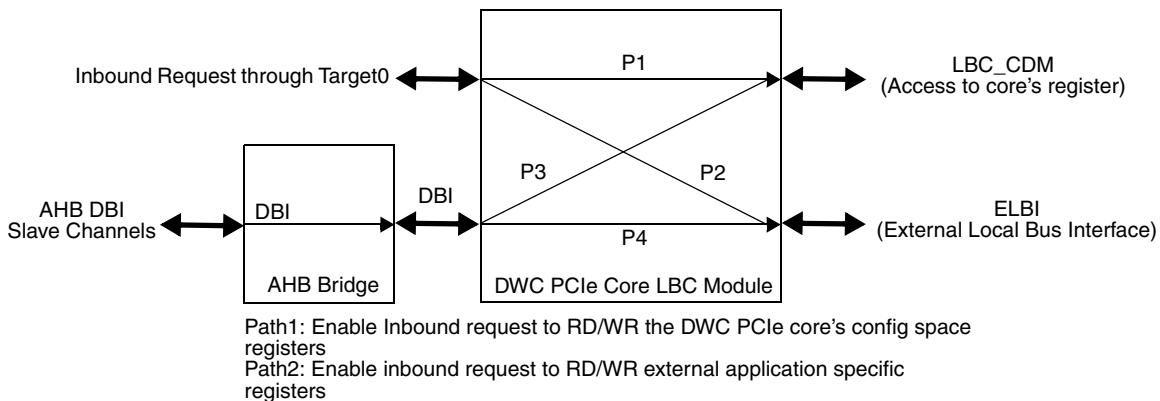
Figure B-4 Byte Ordering Endianess Conversion



B.2.7 Accessing Native PCIe Core Registers and Application-Specific Registers

The PCIe core's DBI interface is used to access the core's CDM or application-specific registers (see [Figure B-5](#)). The application can read from, or write to, native PCIe core registers through the AHB bridge slave DBI interface. The AHB channel read/write address maps to the address of the internal native PCIe core's CDM registers and application-specific register space.

Figure B-5 DWC AXI/AHB-PCIe Core Register Access



The DWC AHB/PCIe bridge IP provides a dedicated slave interface to access all registers (either DWC native core configuration registers or external application-specific registers). This interface is called the DBI slave.

The DWC native PCIe core's LBC (local bus controller) module handles the path switching between the native core CDM registers or the application's registers over the native core's ELBI (external local bus interface). The control to the path switching mechanism is indicated by the application through the address bits of the AHB slave interface (refer to [Table B-6](#)).

A remote device on the PCIe link can also read and write to and from DWC native core configuration registers and external application-specific registers residing on the ELBI bus through an inbound read or write. Inbound reads/writes can access native core CDM registers for configuration type transactions and the external application-specific configuration registers. Inbound memory reads/writes access the external application-specific registers bus-based on their BAR mapping.

Two Important Configurable Parameters

There is a configurable parameter in the native core that enables the multiple function or multiple BAR support at the DBI interface. This parameter is called DBI_MULTI_FUNC_BAR_EN. See "[Parameters](#)" for parameters in the native core. When this parameter is set to enable, then it is necessary for an AHB master to provide the necessary BAR number and function number through the address field.

VALID_DBI_ADDR_WD is a parameter that sets the valid address width for DBI access. This parameter determines the valid address range for a master that issues the access to the DWC PCIe core's register, or external application registers. This parameter is set automatically by the AHB bridge when the application configures the DWC AHB core. The default will be 13 if the application indicates the single function and single BAR access to ELBI. The default will be 19 if the application indicates the multiple function and single BAR access to ELBI (i.e., DBI_MULTI_FUNC_BAR_EN is not enabled). For multiple function and multiple

BARs, or single function and multiple BARs, the default will be set to 21. The value can be adjusted higher if the application desires to access large BAR sizes that are mapped to ELBI.

[Table B-6](#) identifies the AHB address control mapping for the desired DBI access.

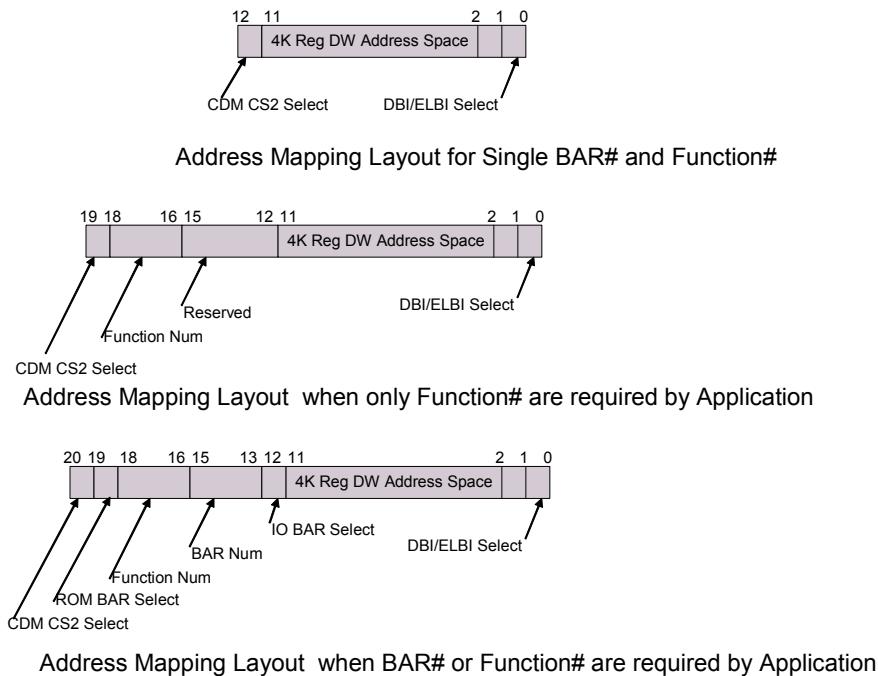
Table B-6 AHB Address Control Mapping for the Desired DBI Access

Address Bit Location	Enabled	Control Function	Descriptions
Bit 0	All	CDM/ELBI select	<ul style="list-style-type: none"> 1'b0: Indicates that the AHB transaction intends to read or write to/from the CDM registers. 1'b1 Indicates that the AHB transaction intends to read or write to/from the application registers located at the ELBI interface. <p>Note: Application configuration or memory-mapped registers are all accessed by setting this bit to 1'b1.</p>
Bit [VALID_DBI_ADDR_WD]	All	CDM CS2 select	<p>1'b1 indicates that the AHB transaction intends to read or write to/from the CDM mask registers. Refer to “Data Bus Interface (DBI)” for information on dbi_cs2 functionality.</p> <p>Note: Bit 0 must be set in order for this to take effect.</p>
Bit [VALID_DBI_ADDR_WD -1]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	ROM BAR access	<p>1'b1 indicates that this AHB transaction is a ROM access to the ELBI interface.</p> <p>Note: ROM is not part of the native PCIe core. Therefore, this transaction should have Bit 0 set to 1'b1.</p>
Bit [VALID_DBI_ADDR_WD -2: VALID_DBI_ADDR_WD -4]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	Memory Read/Write's Function Number	<p>These 3 bits identify the function number of the PCI function that this AHB transaction is attempting to read or write, via the ELBI interface. This function number is only valid for memory accesses within a valid BAR of a particular function.</p> <p>Note: The application must set bit 0 to 1'b0 to access the PL registers from the AHB interface.</p>
Bit [VALID_DBI_ADDR_WD -5: VALID_DBI_ADDR_WD -7]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	Memory Read/Write's BAR number	<p>These 3 bits indicate the BAR number for a memory read or write via the ELBI interface. A value of 3'b111 is reserved to identify that this AHB transaction is not within any valid memory BAR range. This feature allows the application to access the application-specific configuration registers, such as PCIe vendor-specific registers.</p>

Table B-6 AHB Address Control Mapping for the Desired DBI Access (Continued)

Address Bit Location	Enabled	Control Function	Descriptions
Bit [VALID_DBI_ADDR_WD -8]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	IO BAR selected	A value of 1'b1 indicates that this AHB transaction is an access within an IO bar range. Note: This transaction should have Bit 0 set to 1'b1.

For example, if the application has configured the DBI_MULTI_FUNC_BAR_EN to enabled, the address mapping shown in [Figure B-6](#) can be applicable:

Figure B-6 Address mapping with DBI_MULTI_FUNC_BAR_EN enabled

B.2.7.1 Reads/Writes to/from the Native Core CDM Registers

There are two categories of registers residing in the DWC native PCIe core: basic configuration registers that are PCIe-specified; and PL (Port Logic) that are vendor-specific. The DWC AHB bridge has implemented full support to access these registers.

To access CDM registers that belongs to a single function device:

The AHB address of an AHB transaction must be set correctly. Bit 0 set to 1'b0 enables CDM access. If a single function and single BAR at the ELBI is desired, then the DBI_MULTI_FUNC_BAR_EN parameter should not be set when the DWC PCIe core is configured.

To access CDM registers that belong to multiple functions or multiple BARs:

Set the DBI_MULTI_FUNC_BAR_EN parameter when configuring the DWC PCIe core. The function number must be driven by the application onto the proper address location. Address bits[18:16] are the destination function number field for CDM registers.

To access PL registers:

Refer to “[PCIe Registers: Port Logic](#)” for details of PL register mapping. If the application maps a PL register to a BAR, then BAR 0 and function 0 must be set to access PL registers, and bit 0 is set to 1'b0 for CDM access.

B.2.7.2 Reads/Writes to/from the Application-Specific Registers

To access application specific registers that are mapped to memory BAR space:

The application can design its desired registers to map onto either configuration space or memory-mapped BAR space. These registers are accessed through the native DWC PCIe core’s ELBI interface. The AHB address must be set correctly as described in [Table B-6](#). BAR number, function number and other fields must be specified by the application. Bit 0 of the address field must be set to 1'b1 in order to access the application-specific registers.

To access application specific registers that are mapped to configuration space:

As specified in [Table B-6](#), Bit 0 of the address field must be set to 1'b1 to access application registers. The BAR number must be set to 3'b111 to indicate that it is a configuration access.

Default AHB bridge settings:

The default is DBI_MULTI_FUNCT_BAR_EN set to 0, indicating a single function and single BAR at the ELBI interface.

B.2.8 Shared Slave Interface for DBI Access

Not applicable.

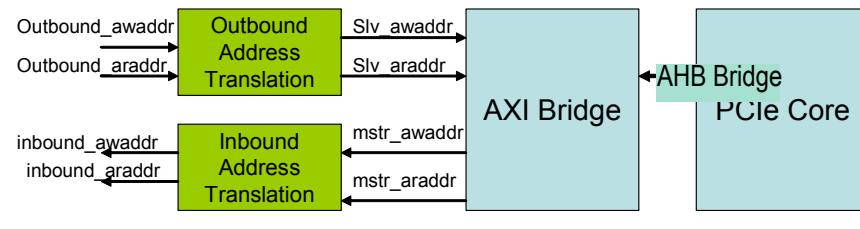
B.2.9 Address Translation When Using the AHB Bridge

The AHB bridge does not provide address translation within the bridge module itself. Normally, for outbound transfers, the address of an AHB master request gets included in a PCIe packet directly as it occurs in AHB. Likewise, the address received from the native PCIe core's TRGT1 interface is presented onto the AHB bridge's master address output, just as it was received in the corresponding PCIe TLP.

There is an address translation assist mechanism built into PCIe native core that allows an AHB master's request or an AHB slave's request being translated. The application is expected to build its desired translations using the native core's address translation assist mechanism.

If the application chooses not to use the native core's address translation assist mechanism, it can always perform its own address translation at the AHB interface. In other words, the desired address translation can be performed through either an application-specific address translation mechanism, through the native core's address translation assist mechanism, or even some combination of the two. [Figure B-7](#) illustrates the two methods of address translation.

Figure B-7 Two Methods of Address Translation



In both mechanisms, the application is required to define how many regions are involved for address translation, and how many types of accesses will be supported (such as memory, I/O, configuration or messages, etc.).

B.2.10 Zero-Byte Transfers Over the AHB Bridge (Flush Semantics)

The PCI Express Base 2.0 Specification supports zero-byte transactions as a means of flushing out previous transactions.

B.2.10.1 Inbound Zero-Byte Transfer

Option 1 (CX_FLT_MASK_HANDLE_FLUSH=1) [RECOMMENDED]:

The native PCIe core has a feature to terminate all zero-byte transactions when it is configured to do so. Enable the bit CX_FLT_MASK_HANDLE_FLUSH (default value is 0) in the Filter Mask Register of the native core to enable this feature. If the native PCIe core is configured to support termination of zero-byte transactions, it will generate completions for zero-byte reads and return them automatically to the requester without sending the transaction to the AHB bridge. It will also drop zero-byte writes. Transactions arriving at the PCIe receive queues will be terminated with the ‘flush’ function.

Configuring the native PCIe core to terminate zero-byte transactions is one way of avoiding handling zero-byte reads at the AHB interface. It has the disadvantage of not flushing out any previous transactions remaining in the AHB bridge’s queue buffer. In other words, the flush semantic terminates at the native PCIe core’s receive queues. See [Section 2.3.8.8, “Filtering Rules Not Defined in PCIe Specification” on page 73](#) and [Section Table 4-296, “Filter Mask 2” on page 520](#).

Option 2 (CX_FLT_MASK_HANDLE_FLUSH=0) [NOT SUPPORTED]:

A second possible means of handling zero-byte transactions is to let the AHB bridge receive the inbound zero-byte transaction from the native PCIe core and present it to the AHB interface.

Write: The AHB specification (and the bridge) does not support zero byte writes.

- ❖ The zero-byte write request could be supported using an extra signal that would be part of the mstr_req_misc_info bus. This is NOT currently supported.

Read: The AHB specification, provides no support mechanism for zero-byte reads.

- ❖ The zero-byte read request could be supported using an extra signal that would be part of the mstr_req_misc_info busses. This is NOT currently supported.

B.2.10.2 Outbound Zero-Byte Transfer

Write: The AHB specification (and the bridge) does not support zero byte writes.

- ❖ The zero-byte write request could be supported using an extra signal that would be part of the slv_req_misc_info bus. This is supported via bits [25:22].

Read: The AHB specification, provides no support mechanism for zero-byte reads.

- ❖ The zero-byte read request could be supported using an extra signal that would be part of the slv_req_misc_info bus. This is supported via bits [25:22].

The AHB bridge IS expected to handle outbound zero-byte write and read transfers.

Bits [25:22] are also used as byte enables for non-contiguous byte access. These bits should be set for both outbound reads and writes.

B.2.11 I/O and CFG Transaction Handling

I/O and CFG-type inbound and outbound transfers are fully supported by the AHB bridge.

[Section B.2.8, "Shared Slave Interface for DBI Access,"](#) also includes information on this type of transfer.

I/O and CFG outbound transfers are signaled to AHB bridge through the slv_req_misc_info bus using the Type field in this bus. Please refer to [Section B.4, "Signal Descriptions,"](#) for the bit location details.

I/O and CFG outbound read/write transfers are issued through the bridge's AHB slave. The address translation glue logic referred to in [Section B.2.8, "Shared Slave Interface for DBI Access,"](#) is responsible for translating an internal AMBA memory access to a PCIe access.

I/O and CFG access by driving the proper type field of slv_req_misc_info. The 5-bit Type field encoding is compliant with the PCI Express Base 2.0 Specification.

I/O and CFG inbound transfers are signaled to AHB master interface through the mstr_req_misc_info bus using the Type field in this bus.

The slv_req_misc_info 5-bit Type fields are encoded as follows:

- ❖ 00000: Memory Read or Write Request
- ❖ 00001: Memory Read Request-Locked
- ❖ 00010: IO Read or Write Request
- ❖ 00100: Configuration Read or Write Type 0
- ❖ 00101: Configuration Read or Write Type 1
- ❖ 10***: Message Request
- ❖ 01010: Completion
- ❖ 01011: Completion for Locked Request

Please refer to [Section B.4, "Signal Descriptions,"](#) for additional details.

The responses to an inbound master I/O read or write are expected to be generated by the AHB slave within the application.

Note that there is a special requirement for the master channel. The application slave is required to drive bit [12] in mstr_resp_misc_info to indicate the type of a response expected. This bit is set to indicate whether the response is to be Posted or Non-Posted.

If a response is for a master-Posted request (such as memory write or message), then the response is a Posted-type response. Therefore, mstr_resp_misc_info[12] should be deasserted during the response cycle.

If a response is for a master Non-Posted request (configuration read/write, I/O read/write, or memory read), then the response is considered as a Non-Posted response. Therefore, mstr_resp_misc_info[12] should be asserted to 1'b1 during the response cycle.

If the application does not support a Non-Posted write or read, then bit [12] will be tied to 1'b0. Please refer to [Section B.4, "Signal Descriptions,"](#) for details.

Note: if there is an ATU (address translation unit) attached to the PCIe core's Address Translation Interface and that ATU is converting MemWr TLP's to CfgWr TLP's, then mstr_resp_misc_info[12] should be set to 1'b1.

Note: Since all types of reads are non-Posted, then mstr_resp_misc_info[12] should always be set to 1'b1 for any type of reads.

B.2.12 Message Transaction Handling

There are many types of PCIe messages defined in the PCI Express Base 2.0 Specification. They are not part of the AHB specification. Therefore, messages of interest must be translated by the AHB bridge or other interface. PCIe specification-defined message types include interrupt messages, power management messages, AER messages, vendor messages, and slot messages.

The core provides two methods for message handling. The first message handling method is to have the native PCIe core handle all messages. Power management messages are required to be handled by the native PCIe core. The AER messages are also handled by the native core through the native core's SII interface. The native core has a configuration option to select whether all messages are handled, or a subset of them. We suggested using this method when the AHB is configured with the PCIe core.

Another message handling method is to allow the native core to pass messages to the AHB bridge, or transmit messages that are generated by the AHB interface. The native core must be configured to pass messages to the AHB bridge in this mode of operation. Inbound messages are handled through the master AHB interface. Outbound messages are handled through the slave AHB interface. The messages are required to have a payload since the AHB bus requires all writes to have a minimum of one burst.

Message codes and other header information of a message are obtained through the misc_info busses of both inbound and outbound direction. Please refer to [Section B.4, "Signal Descriptions,"](#) for details related to misc_info.

B.2.13 Transaction Order Enforcement Through the AHB Bridge

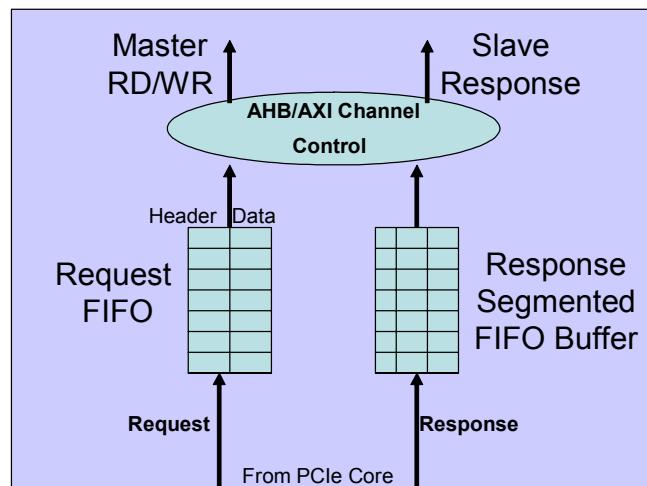
AHB bridge traffic ordering is handled as follows.

B.2.13.1 For inbound traffic:

There is a FIFO for request and segmented buffer queue for response in AHB bridge which enqueues the inbound request received by the native core. The PCIe order rule enforcement is performed within the native core. When an inbound request is passing through the AHB bridge, it is currently done in an in-order service fashion. This means that there is no reordering after an inbound request has left the native core's receive queue. Please refer to the order enforcement application note for additional information. [Figure B-8](#) highlights the internal inbound queuing architecture of the AHB bridge.

The responses for inbound requests in the AHB bridge module are on a first-come first-served basis. This means that the inbound requests targeted to different slaves can be served out-of-order. It is the application's responsibility to ensure that a split inbound read request will not target different AHB slaves.

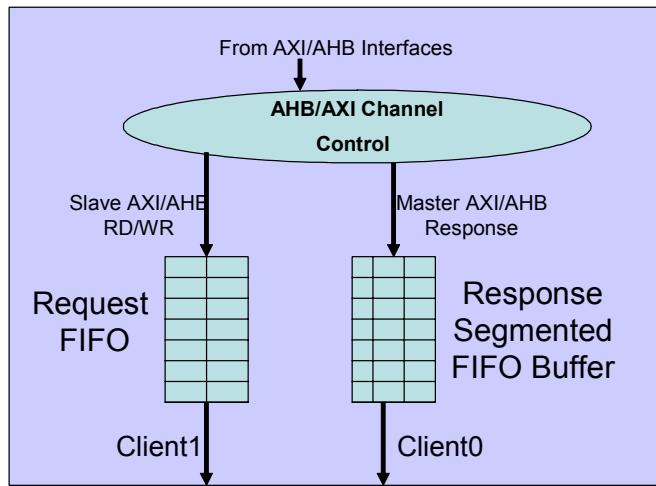
Figure B-8 AHB Bridge Inbound Traffic Queuing Architecture



B.2.13.2 For outbound transfers:

There is a FIFO and composer segmented queue structure designed within the AHB bridge. The queue structure supports only in-order services. Outbound requests are served onto the PCIe wire in the order that it is received from an AHB master. Please refer to the order enforcement application note for additional information.

[Figure B-9](#) highlights the internal outbound queuing architecture of the AHB bridge.

Figure B-9 AHB Bridge Outbound Traffic Queuing Architecture

The AHB bridge can be configured to respond with either a split or non-split mode. If a split mode is selected by configuring CC_RESPONSE_MODE to SPLIT, a read request from one master will be returned with a split response such that another master can issue another read if multiple masters are configured. If a non-split response mode is selected, then slv_hready_resp will be deasserted until a response has been received from the PCIe wire.

When in split response mode, and the PCIe completion has been received by the AHB bridge for a particular read, the master will be notified with a response that matches the request.

A write request from one master will be returned with an “OK” response immediately, if there is no prefetched read response data to be discarded, due to the fact the PCIe protocol does not require completions for Posted requests. Otherwise, a write request will be returned with response retry status if the AHB bridge module is busy at discarding the previously prefetched read data.

B.2.14 Memory Read/Write With Data Gaps (“Holes”)

The PCI Express Base 2.0 Specification supports memory reads and writes with “holes,” only if the read and write requests are less than or equal to 2 DWORDS in length. The PCIe transaction header first and last byte enable fields are used to signal the valid byte location. By “hole,” we mean that it is possible to have non-contiguous byte enables such as 4'b0101.

AHB protocol does not quit support a burst with holes in it. An AHB outbound transfer from AHB bridge will not contain holes. On the other hand, the AHB bridge must support inbound transfers with holes. The mechanism to support inbound reads with holes are described as follows.

In the case of reads, a read received over the PCIe core with holes will be sent to the AHB master channel as a sequential read, assuming holes do not exist. This sequential read burst is associated with a mstr_req_misc_info that contains the first and last byte enable of this PCIe read request. It is up to application to analyze mstr_req_misc_info and the address location to determine whether or not this is a non-prefetchable access. If it is, a sequential read progresses with responses of all bytes with and without holes. If it is determined that this is a non-prefetchable access, then it is the application’s responsibility to split the request. The same applies to writes.

B.2.15 MTU Read Request Size and Page Boundary Support

A PCIe device is programmed to support certain maximum write transfer sizes and maximum read request sizes. The native PCIe core's configuration module contains the device's maximum Transfer Unit (MTU) and maximum read request size information.

The AHB bridge supports single or incremental burst outbound transfers with a maximum of a 64-byte read or write.

The AHB bridge supports single or INCR (undefined length) burst outbound transfers with the maximum length defined by the application.

The AHB bridge supports mismatches that occur when the AHB maximum transfer length is different than the PCIe MTU and maximum read request size.

The application must define its system requirements such as PCIe MTU and maximum read request size. There are resources allocated within the AHB bridge that are set based on these system parameters.

For example, an inbound read transfer has an associated response buffer that is dependent on the remote PCIe device's maximum read request size. An inbound write transfer has a master write buffer that is dependent on the remote PCIe device's maximum transfer size.

The PCI Express Base 2.0 Specification defines a page boundary of 4 KB as a requirement. The AHB interconnect typically also has a 1 KB page boundary requirement which is the default setting. The inbound page boundaries that the AHB bridge currently supports are 128 bytes, 256 bytes, 512 bytes, 1 KB, 2 KB and 4 KB. The outbound AHB page boundary support is set at 4 KB based on the PCIe's requirements.

B.2.16 Legacy Interrupt Handling and Other Message Handling Over the AHB Bridge

Legacy interrupts can be handled in PCIe, based on either legacy interrupt support through PCIe messages over the SII interface, or through the MSI protocol as a Posted write transfer. The AHB bridge can handle both of these methods. Furthermore, the bridge can handle vendor message with a payload. How the bridge handles MSI protocol is described in [Section B.2.18, “MSI Handling Over the AHB Bridge.”](#)

B.2.16.1 Native PCIe Core Transmit Interrupt Messages (Outbound Traffic)

The application is required to drive PCIe native core signal sys_int to enable the core to generate legacy interrupt messages.

B.2.16.2 Native PCIe Core Receive Interrupt Messages (Inbound Traffic)

The native PCIe core receives legacy interrupt messages and issues them via the PCIe native core’s SII interrupt signals. The application should monitor the interrupt assert and deassert signals for inbound legacy interrupt handling.

B.2.17 PCIe Vendor-Defined Messages

The vendor messages are handled in a manner similar to interrupt messages discussed above. An AHB data channel can present a data payload for a PCIe vendor-defined message.

The outbound vendor message can be created by an AHB master through the slv_req_misc_info bus. The inbound vendor message is presented by the AHB bridge through mstr_awmisc_info for the message type and other attributes. Refer to [Section B.2.8, “Shared Slave Interface for DBI Access,”](#) and [Section B.4, “Signal Descriptions,”](#) for details.



Note If the application is required to form a message such as a vendor message, it must support a 64-bit address because of the address field overlay to the message's header3 and header4 field.

Otherwise, the header4 field of a vendor message will be wired to 0.

A vendor messages must contain a payload.

B.2.18 MSI Handling Over the AHB Bridge

There are two methods to send and receive MSI messages. These are discussed in the following two sections.

B.2.18.1 Using the Regular AHB Interface With a Regular Memory Write Operation

One method is to have the AHB bridge send and receive MSI requests in the same manner as a regular memory write. It is the application's responsibility to form the MSI request based on the native PCIe core's configuration. The native PCIe core CDM block contains the MSI address, enable, etc. The application must obtain the MSI information from the native PCIe core SII interface, or from reading the CDM registers through the DBI interface, and then form the MSI request to present onto the AHB bridge AHB slave interface. This method should be used by an application that preserves the order of Posted transfers requested before an MSI request.

The AHB bridge simply sends and receives MSI requests in the same manner as a memory write. To send an MSI to the PCIe link from the AHB bridge, the application presents the MSI address and data onto the AHB slave write channel. To receive a PCIe MSI packet, the AHB bridge presents the MSI address (like a regular memory write address), and data (like regular memory write data), onto the master write channel.

The native PCIe core outputs `cfg_msi_addr`, which should be used to form a memory write for MSI requests by an AHB master. The native PCIe core outputs `cfg_msi_64`, which should be used to form a 64-bit addressed memory write by an AHB master. The native PCIe core also outputs `cfg_msi_data` as one DWORD of memory write data when an AHB master issues an MSI request. The lower MSB of the data, up to 5 bits, determines the MSI vector number. The application can control the vector number by replacing the lower MSB of the memory write data, up to 5 bits, to support a maximum of 32 MSI vectors. The maximum number of MSI vectors supported is a core configuration parameter.

B.2.18.2 Using the Core's Native Interface (SII)

The second method to send MSI messages is to use the native PCIe core's MSI interface (part of the SII interface group) to signal the native PCIe core to send an MSI request. The native PCIe core has an MSI request formation and transmission capability and does not terminate an MSI request received. This second method is used for applications that do not have restricted ordering. When an MSI message is pending to be sent, it can bypass other previously Posted transfers. An MSI request presented through native PCIe core's MSI interface will be transmitted as soon as Posted credit is available. The MSI request will be received through the native PCIe core's receive interface, just as for a regular memory write. The termination of an MSI request is up to the application.

B.2.19 Parity Checking Over AHB Bridge

The AHB bridge does not support either data path parity or RAM parity. It is up to the application to select external AHB bridge RAM capability and add parity check circuitry to RAM.

B.3 Top-Level I/O Diagrams

Figure B-3 shows a block diagram of the DWC PCIe AHB core with respect to signal interfaces. Figures B-10 and B-11 illustrate the AHB Bridge Module top-level I/O signals. The signal interface group names (grouped according to function) in these figures are cross-referenced to the signal description tables that follow in Section B.4. (For example, click on “AHB Clock and Reset Signals” in Figure B-10 to hyperlink to Table B-7 on page 687 for a description of these signals.)

Figure B-10 AHB Bridge Module Top-Level I/O Diagram (1 of 2)

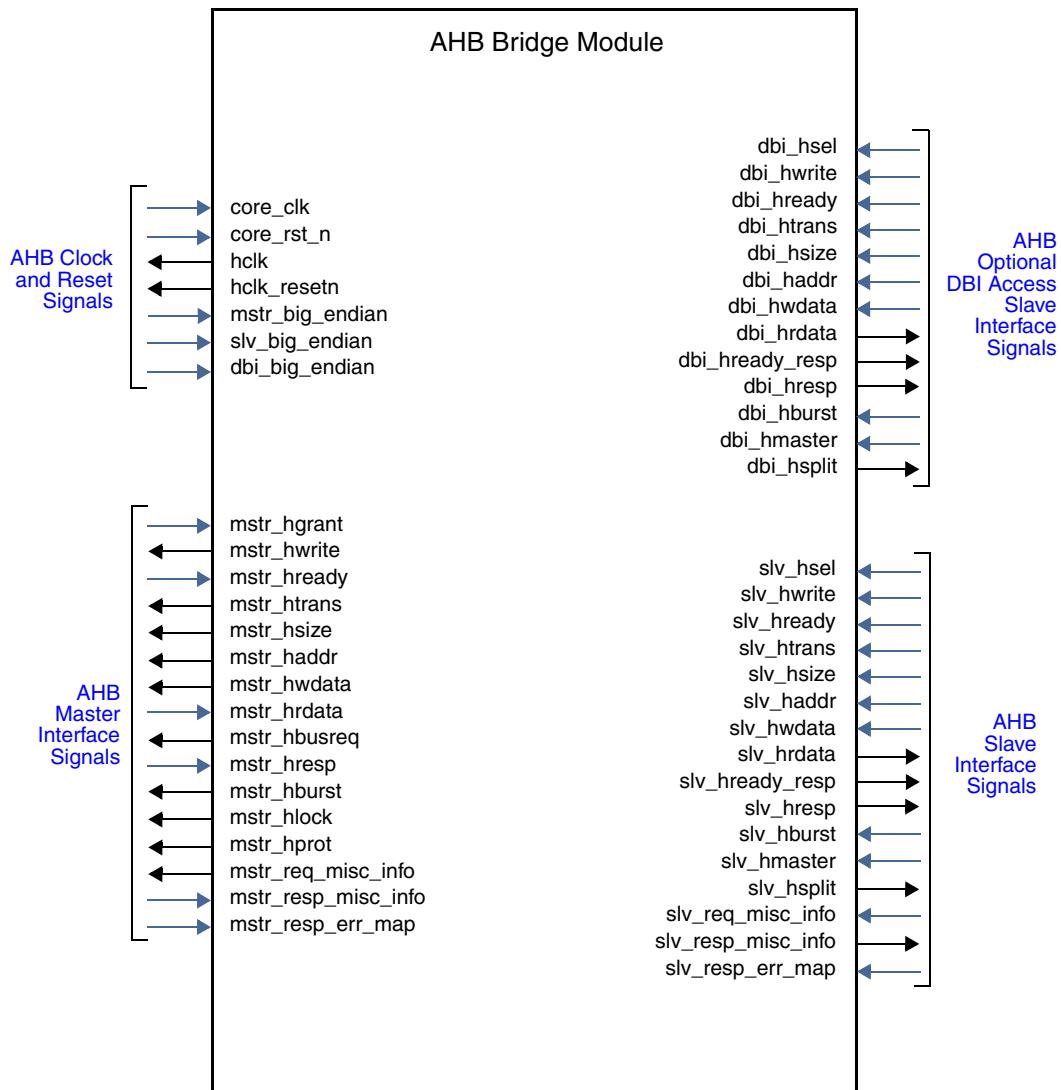
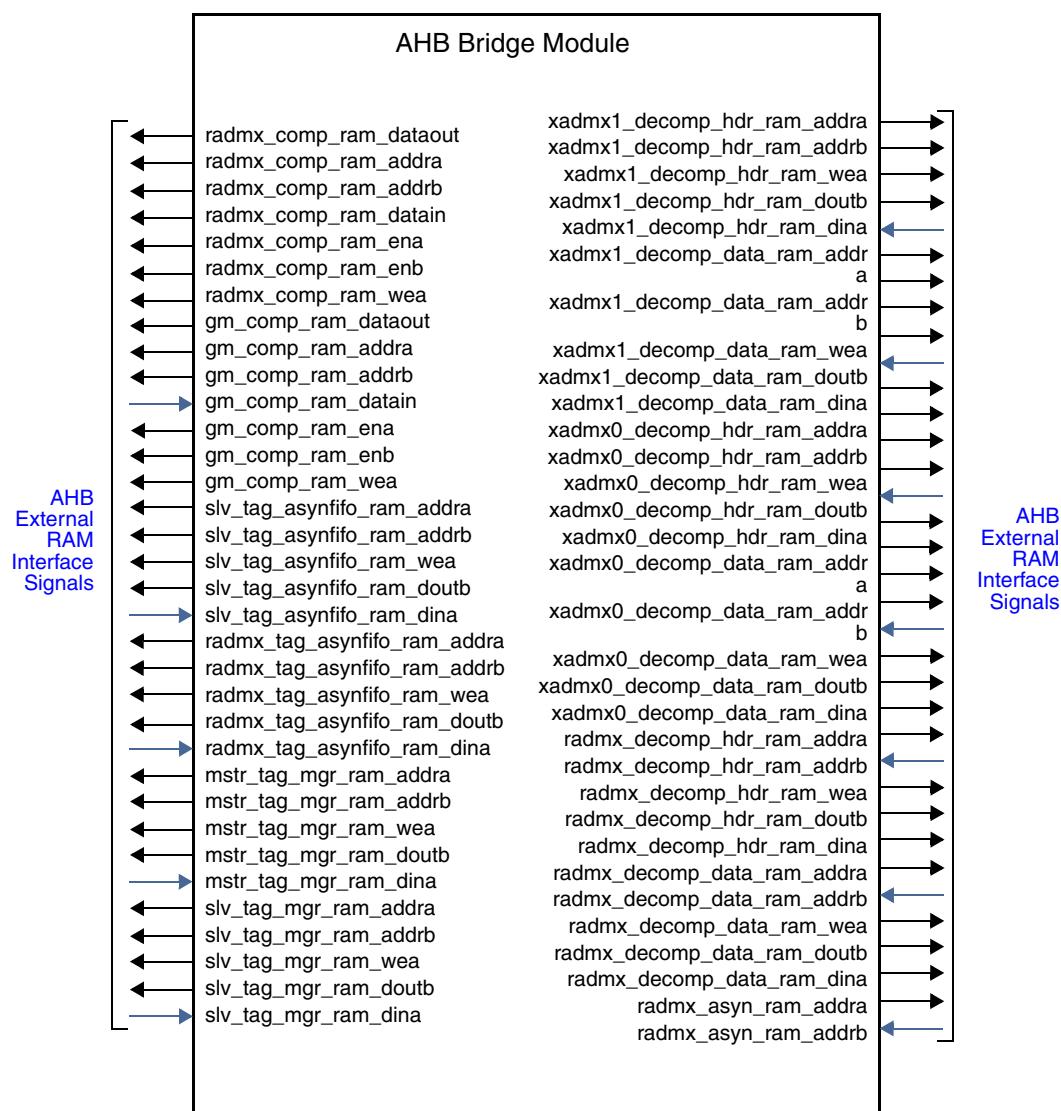


Figure B-11 AHB Bridge Module Top-Level I/O Diagram (2 of 2)

B.4 Signal Descriptions

The following sections describe the AHB Bridge Module top-level interface signals, grouped according to function.



Only the top-level AHB signals associated with the AHB Bridge Module itself are described here.

- ❖ [AHB Clock and Reset Signals](#)
- ❖ [AHB Master Interface Signals](#)
- ❖ [AHB Slave Interface Signals](#)
- ❖ [AHB Optional DBI Access Slave Interface Signals](#)
- ❖ [AHB External RAM Interface Signals](#)
- ❖ [AHB Bridge SII Interface Addition](#)

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

- ❖ Registered: Indicates whether or not the signal is registered directly at the core boundary. 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 core.
- ❖ Synchronous to: Indicates which clock the signal is synchronous to, or if the signal is asynchronous.



- All signals with “_n” are active low. For example, signal core_rst_n is an active low signal; that is, it is asserted low.
- Port reset signals are named according to the AHB protocol, and also required to be named as such for coreAssembler. These are also active low. For example, signal slv_hresetn is an active low reset signal.

B.4.1 AHB Clock and Reset Signals

Table B-7 defines the AHB Bridge Module Clock And Reset signals.

Table B-7 AHB Clock and Reset Signals

Signal	Width (bits)	I/O	Description
core_clk	1	I	<p>Primary Clock</p> <p>Function: The primary clock input to the DM Core. It is assumed that all input signals of the native core are synchronous to this clock, unless otherwise stated. Depending on the DM Core configuration, core_clk is either 62.5 MHz, 125 MHz, 250 MHz, or 500 MHz.</p> <p>You must update this hyperlink manually. Remember the “&fixedin=<version number>” at the end to point the reader straight to the most recent STAR Report.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
core_rst_n	1	I	<p>Primary Core Reset Signal</p> <p>Function: The primary reset to the core</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
hclk	1	O	<p>AHB Clock</p> <p>Function: AHB clock for both AHB master and slave interface. It is configurable to set hclk to the same as core_clk.</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
hresetn	1	O	<p>AHB Lock Reset</p> <p>Function: AHB reset for both AHB master and slave interface.</p> <p>Note: We recommended that the application reset the AHB bridge when it is asserting a reset to the core.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
mstr_big_endian	1	I	<p>Master AHB Interface Big Endian Enable</p> <p>Function: To control the master AHB data bus access using big endian or little endian. This signal is an optional signal that only exists when the AHB_ENDIANNESS parameter is set to DYNAMIC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>

Table B-7 AHB Clock and Reset Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_big_endian	1	I	<p>Slave AHB Interface Big Endian Enable</p> <p>Function: To control the slave AHB data bus access using big endian or little endian. This signal is an optional signal that only exists when the AHB_ENDIANNESS parameter is set to DYNAMIC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
dbi_big_endian	1	I	<p>DBI Slave AHB Interface Big Endian Enable</p> <p>Function: To control the DBI slave AHB data bus access using big endian or little endian. This signal is an optional signal that only exists when the AHB_ENDIANNESS parameter is set to DYNAMIC.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>

B.4.2 AHB Master Interface Signals

Table B-8 defines the AHB Bridge Module Master Interface signals.

Table B-8 AHB Master Interface Signals

Signal	Width (bits)	I/O	Description
mstr_hgrant	1	I	<p>AHB Master Grant</p> <p>Function: Master request is granted when asserted.</p> <p>Active State:</p> <p>Registered: Yes</p> <p>Synchronous to: hclk</p>
mstr_hwrite	1	O	<p>AHB Master Write</p> <p>Function: Write control. Asserted during write transfer, negated during read transfer.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to:</p>
mstr_hready	1	I	<p>AHB Master Ready</p> <p>Function: Ready response from the selected master. Indicates that the previous transfer is complete. This signal is passed to all bus masters and slaves.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to:</p>
mstr_htrans	2	O	<p>AHB Master Transfer</p> <p>Function: Transfer type:</p> <ul style="list-style-type: none"> • 2'b00: IDLE • 2'b01: BUSY • 2'b10: NONSEQUENTIAL • 2'b11: SEQUENTIAL. <p>Registered: No</p> <p>Synchronous to:</p>
mstr_hsize	3	O	<p>AHB Master Transfer Size</p> <p>Function: Transfer size. Valid values are: 3'b000 - 8 bits; 3'b001 - 16 bits; 3'b010 - 32 bits.</p> <p>All other values are not supported currently by AHB Bridge. Transfer size should not exceed 32 bits if data bus width (AHB_DATA_WIDTH) is set to 32 bits or wider and 16 bits if data bus width is set to 16 bits. Violation of this rule results in ERROR response (hresp = 2'b01).</p> <p>Registered: No</p> <p>Synchronous to:</p>

Table B-8 AHB Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_haddr	32	O	AHB Master Address Function: AHB Master Address bus Registered: No Synchronous to:
mstr_hwdata	AHB_DATA_WIDT H	O	AHB Master Write Data Function: AHB Master Write data bus Registered: Yes Synchronous to:
mstr_hrdata	AHB_DATA_WIDT H	I	AHB Master Read Data Function: AHB Master Read data bus Registered: Yes Synchronous to: hclk
mstr_hbusreq	1	O	AHB Master Bus request Function: AHB bus request as a master. When asserted, indicates that the current transfer starts. Active State: Registered: Yes Synchronous to:
mstr_hresp	2	I	AHB Master Response Function: Transfer response. Provides information on the status of the current transfer: <ul style="list-style-type: none">• 2'b00: OKAY• 2'b01: ERROR• 2'b10: RETRY• 2'b11: SPLIT Registered: Yes Synchronous to: hclk
mstr_hburst	3	O	AHB Master Burst Function: Burst type and length control. Registered: Yes Synchronous to:
mstr_hlock	1	O	AHB Master Lock Function: Master issue lock request. Currently not supported Active State: Registered: Synchronous to:

Table B-8 AHB Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_hprot	4	O	<p>AHB Master Protect</p> <p>Function: Protect Mode. Not supported currently.</p> <p>Active State:</p> <p>Registered:</p> <p>Synchronous to:</p>
mstr_req_misc_info	MSTR_MISC_INFO_WD = 24 bits	O	<p>AHB Master Transaction Associated Misc Information (from the TLP received by the native PCIe core)</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AHB interface. This signal is only valid during address phase.</p> <ul style="list-style-type: none"> • [2:0]: Transfer's byte offset • [5:3] Response status suggested by native PCIe core filter • [8:6]: BAR number of Posted TLP • [9]: TLP is an I/O • [10]: TLP is in ROM range • [13:11]: TLP's function number • [15:14]: TLP's attributes • [18:16]: TLP's TC • [23:19]: TLP's Type <p>Notes:</p> <ol style="list-style-type: none"> 1. Bit 0 to Bit7 also indicates the message code when the type is message. 2. The native PCIe core is configured to handle TAG lookup for response of the master request. Therefore it will store the request ID and TC, attributes for its corresponding TAG. 3. Bus number, device number, function number, and register numbers are overlaid to the address bus when type is configuration transactions. <ul style="list-style-type: none"> • bus_number = addr[31:24] • dev_number = addr[23:19] • func_number = addr[18:16] • ext_reg_number = addr[11:8] • reg_number = addr[7:2] <p>Registered: N/A</p> <p>Synchronous to: core_clk or hclk</p>

Table B-8 AHB Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_resp_misc_inf_o	*	I	<p>AHB Master Write Response Transaction Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of standard AHB interface. This signal is only valid during data phase.</p> <ul style="list-style-type: none"> • [1:0]: AHB response's attributes • [2]: AHB response with Bad EOT assigned to signal native PCIe core to drop this response • [5:3]: AHB response's TC • [6]: AHB response's BCM • [7]: AHB response's EP • [8]: AHB response's TD • [11:9]: AHB response's Function number • [12]: Indicates that this response is for a Non-Posted transfer. It is used to indicate to the core that the current AHB response should translate into a PCIe response packet (Cpl/CplID). For example, in the case of a MemWr transfer (or a Msg) this bit should be 1'b0, while for MemRd, IORd, IOWr, CfgRd & CfgWr operations it should be 1'b1. See Section B.2.11, “I/O and CFG Transaction Handling,” <p>Note: if there is an ATU (address translation unit) attached to the PCIe core's Address Translation Interface and that ATU is converting MemWr TLP's to CfgWr TLP's, then bit [12] should be set to 1'b1.</p> <p>*Width: mstr_req_misc_info_WD</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or hclk</p>
mstr_resp_err_map	2	I	<p>AHB Master Response Error Map</p> <p>Function: This map signal is designed to allow the application to select a master response error report mechanism received from an AHB response channel to CPL status of native PCIe core transmission. Only one bit is used. The MSB bit is for future use. When this map signal is set to 1, it will set an AHB response error to a UR of PCIe completion. When the map signal is set to 0, it will set an AHB response error to a CA of PCIe completion. Default is set to all 1s.</p> <p>[0]: AHB response error</p> <p>[1]: Not defined (tie to 0): High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

B.4.3 AHB Slave Interface Signals

Table B-9 defines the AHB Bridge Module Slave Interface signals.

Table B-9 AHB Slave Interface Signals

Signal	Width (bits)	I/O	Description
slv_hsel	1	I	<p>AHB Slave Select</p> <p>Function: Slave select. Asserted when the current transfer on the AHB bus is intended for the DW_hsata.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
slv_hwwrite	1	I	<p>AHB Slave Write</p> <p>Function: Write control. Asserted during write transfer, negated during read transfer.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
slv_hready	1	I	<p>AHB Slave Ready</p> <p>Function: Ready response from the selected slave. Indicates that the previous transfer is complete. This signal is passed to all bus masters and slaves.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
slv_htrans	2	I	<p>AHB Slave Transfer</p> <p>Function: Transfer type:</p> <ul style="list-style-type: none"> • 2'b00: IDLE • 2'b01: BUSY • 2'b10: NONSEQUENTIAL • 2'b11: SEQUENTIAL <p>Registered: No</p> <p>Synchronous to: hclk</p>

Table B-9 AHB Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_hsize	3	I	<p>AHB Slave Transfer Size</p> <p>Function: Transfer size. Valid values are:</p> <ul style="list-style-type: none"> • 3'b000: 8 bits • 3'b001: 16 bits • 3'b010: 32 bits <p>All other values are not supported currently by AHB Bridge. Transfer size should not exceed 32 bits if data bus width (AHB_DATA_WIDT) is set to 32 bits or wider and 16 bits if data bus width is set to 16 bits. Violation of this rule results in ERROR response (hresp = 2'b01).</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
slv_haddr	32	I	<p>AHB Slave Address</p> <p>Function: Address bus</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
slv_hwdata	AHB_DATA_WIDT H	I	<p>AHB Slave Write Data</p> <p>Function: Write data bus</p> <p>Registered: Yes</p> <p>Synchronous to: hclk</p>
slv_hldata	AHB_DATA_WIDT H	O	<p>AHB Slave Read Data</p> <p>Function: Read data bus</p> <p>Registered: Yes</p> <p>Synchronous to: hclk</p>
slv_hready_resp	1	O	<p>AHB Slave Ready Response</p> <p>Function: AHB bridge slave interface ready response. When asserted, indicates that the current transfer is complete.</p> <p>Active State:</p> <p>Registered: Yes</p> <p>Synchronous to: hclk</p>
slv_hresp	2	O	<p>AHB Slave Response</p> <p>Function: Transfer response - provides information on the status of the current transfer:</p> <ul style="list-style-type: none"> • 2'b00: OKAY • 2'b01: ERROR • 2'b10: RETRY • 2'b11: SPLIT <p>Registered: Yes</p> <p>Synchronous to: hclk</p>

Table B-9 AHB Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_hburst	3	I	AHB Slave Burst Function: Burst type and length control. Registered: Yes Synchronous to:
slv_hmaster	4	I	AHB Slave Master Function: Master ID Tag Registered: Yes Synchronous to:
slv_hsplit	16	O	AHB Slave Split Function: Split Master select. Registered: Yes Synchronous to:

Table B-9 AHB Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_req_misc_info	*	I	<p>AHB Slave Transaction Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AHB interface. This signal is only valid during address phase.</p> <ul style="list-style-type: none"> • [4:0]: AHB transaction's type • [5]: AHB Transaction's BCM • [6]: AHB Transaction's EP • [7]: AHB Transaction's TD • [9:8]: AHB Transaction's ATTR • [12:10]: AHB Transaction's TC • [20:13]: AHB Transaction's MSG code • [21]: AHB transaction is a DBI access. This is for SHARED DBI mode only. Currently it is not applicable for AHB configuration. Therefore, this is reserved. • [25:22]: These 4 bits are the byte enables of a request in order to support the zero-byte and non-contiguous byte transfers. See Section B.2.10, “Zero-Byte Transfers Over the AHB Bridge (Flush Semantics)” on page 674 • [28:26]: TLP's function number. This is driven by the application master to identify the function that is issuing the outbound (AHB application -> PCIe wire) request (e.g. MemRd). This signal is used by the PCIe core to generate the RID (Requestor ID) of the outbound PCIE request (e.g. MemRd). <p>Note: The configuration transfer type will have bus number, device number, function number, and register number driven onto address bus[31:0].</p> <ul style="list-style-type: none"> • bus_number = addr[31:24] • dev_number = addr[23:19] • func_number = addr[18:16] • ext_reg_number = addr[11:8] • reg_number = addr[7:2] <p>*Width: SLV_AHB_MISC_INFO_WD</p> <p>Registered: No</p> <p>Synchronous to: core_clk or hclk</p>

Table B-9 AHB Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_resp_misc_info *	*	O	<p>AHB Slave Transaction's Response Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AHB interface. This signal is only valid during data phase.</p> <ul style="list-style-type: none"> • [0]: AHB response transaction's BCM • [1]: AHB response transaction's TD • [4:2]: AHB response transaction's TC • [6:5]: AHB response transaction's attributes • [9:7]: AHB response transaction's status defined as the same location of PCIe completion header status field. • [10]: Indicates that AHB response is for a Non-Posted request. • [13:11]: TLP's function number. This is driven by the AHB bridge slave to identify the application function that the inbound (PCIe wire -> AHB application) completion (e.g. Cpl/CplID) is destined for. This signal is derived by the PCIe core from the RID (Requestor ID) of the inbound PCIE completion (e.g. Cpl/CplID). <p>*Width: slv_req_misc_infomstr_req_misc_info_WD</p> <p>Registered: Only registered</p> <p>Synchronous to: core_clk or hclk</p>
slv_resp_err_map	6	I	<p>AHB Slave Response Error Map</p> <p>Function: This map signal is designed to allow the application to select a slave response error report mechanism received from PCIe completion. Since there are six kinds of PCIe completion errors that the core can report to the AHB interface, there are six bits mapped to select these errors corresponding to an AHB error response. The application can choose to not assert the AHB response error as a slave when these completion errors are received by the PCIe link. The default is 1 which indicates that these errors are reported as an AHB error.</p> <ul style="list-style-type: none"> • [0]: CPL UR • [1]: CPL CA • [2]: Not used • [3]: CPL Poisoned • [4]: CPL Time-out • [5]: CPL Abort (unexpected) <p>Registered: No</p> <p>Synchronous to: CPL Timeout</p>

B.4.4 AHB Optional DBI Access Slave Interface Signals

Table B-10 defines the AHB Bridge Module optional DBI Access Slave Interface signals.

Table B-10 AHB Optional DBI Access Slave Interface Signals

Signal	Width (bits)	I/O	Description
dbi_hsel	1	I	<p>AHB DBI Select</p> <p>Function: Slave DBI interface select. Asserted when the current transfer on the AHB bus is intended for the DW_hasta.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
dbi_hwrite	1	I	<p>AHB DBI Write</p> <p>Function: Write control. Asserted during write transfer, negated during read transfer.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
dbi_hready	1	I	<p>AHB DBI Ready response</p> <p>Function: Ready response from the selected slave. Indicates that the previous transfer is complete. This signal is passed to all bus masters and slaves.</p> <p>Active State:</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>
dbi_htrans	2	I	<p>AHB DBI Transfer</p> <p>Function: Transfer type:</p> <ul style="list-style-type: none"> • 2'b00: IDLE • 2'b01: BUSY • 2'b10: NONSEQUENTIAL • 2'b11: SEQUENTIAL. <p>Registered: No</p> <p>Synchronous to: hclk</p>
dbi_hsize	3	I	<p>AHB DBI Transfer Size</p> <p>Function: Transfer size. Valid values are:</p> <ul style="list-style-type: none"> • 3'b000: 8 bits • 3'b001: 16 bits • 3'b010: 32 bits. <p>All other values are not supported by AHB bridge currently. Transfer size should not exceed 32 bits if data bus width (AHB_DATA_WIDTH) is set to 32 bits or wider and 16 bits if data bus width is set to 16 bits. Violation of this rule results in ERROR response (hresp = 2'b01).</p> <p>Registered: No</p> <p>Synchronous to: hclk</p>

Table B-10 AHB Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_haddr	32	I	AHB DBI Address bus Registered: No Synchronous to: hclk
dbi_hwdata	AHB_DATA_WIDT H	I	AHB DBI Write data bus. Registered: Yes Synchronous to: hclk
dbi_hrdata	AHB_DATA_WIDT H	O	AHB DBI Read data bus. Registered: Yes Synchronous to: hclk
dbi_hready_resp	1	O	AHB DBI Ready Response Function: AHB bridge DBI interface slave ready response. When asserted, indicates that the current transfer is complete. Registered: Yes Synchronous to: hclk
dbi_hresp	2	O	AHB DBI Transfer Response Function: Transfer response - provides information on the status of the current transfer, only OKAY response is supported: <ul style="list-style-type: none">• 2'b00: OKAY Registered: Yes Synchronous to: hclk
dbi_hburst	3	I	AHB DBI Burst Function: Burst type and length control Registered: Yes Synchronous to: hclk
dbi_hmaster	4	I	AHB DBI Master Function: Master ID tag Registered: Yes Synchronous to: hclk
dbi_hsplit	16	O	AHB DBI Split Function: Split master select Registered: Yes Synchronous to: hclk

B.4.5 AHB External RAM Interface Signals

Table B-11 defines the AHB Bridge Module External RAM Interface signals. Descriptions of the RAMs are provided in Section B.6.5.

Table B-11 AHB External RAM Interface Signals

Signal	Width (bits)	I/O	Description
Composer RAM Ports			
Notes:			
<ul style="list-style-type: none"> RADMX composer RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True GM composer RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True and MASTER_POPULATED=True 			
radmx_comp_ram_dataout	*	O	RAM Read Data *Width: [RADMX_COMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_addra	*	O	RAM Port A address (write interface) *Width: [(RADMX_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_addrb	*	O	RAM Port B Address (read interface) *Width: [(RADMX_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_datain	*	I	RAM Write Data *Width: [(RADMX_COMPOSER_DATAQ_WD)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_ena	1	O	RAM Port A Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_enb	1	O	RAM Port B Enable (read enable) Active State: N/A Registered: No Synchronous to: core_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_comp_ram_wea	1	O	RAM Port A Write Enable Active State: N/A Registered: No Synchronous to: core_clk
gm_comp_ram_dataout	*	O	RAM Read Data *Width: [MSTR_COMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_addrA	*	O	RAM Port A Address (write interface) *Width: [(MSTR_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_addrB	*	O	RAM Port B Address (read interface) *Width: [(MSTR_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_datain	*	I	RAM Write Data *Width: [(MSTR_COMPOSER_DATAQ_WD)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_ena	1	O	RAM Port A Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_enb	1	O	RAM Port B Enable (read enable) Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_wea	1	O	Ram Port A Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
Tag RAM Ports			
Notes:			
• Slave TAG asynchronous RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True and AHB_CLK_DIFF_ENABLE=True			
slv_tag_asynfifo_ram_addrA	*	O	Slave Tag Asynchronous FIFO RAM Port A Address *Width: [CLIENT1_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to:
slv_tag_asynfifo_ram_addrB	*	O	Slave Tag Asynchronous FIFO RAM Port B Address *Width: [CLIENT1_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_asynfifo_ram_weA	1	O	Slave Tag Asynchronous FIFO RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
slv_tag_asynfifo_ram_doutB	*	O	Slave Tag Asynchronous FIFO RAM Port B Data Out *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_asynfifo_ram_dinA	*	I	Slave Tag Asynchronous FIFO RAM Port A Address *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
Decomposer RAM Ports			
Notes:			
• RADMX TAG Asynchronous header and data RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True and MASTER_POPULATED and AHB_CLK_DIFF_ENABLE=True			
radmx_tag_asynfifo_ram_addrA	*	O	RADM Tag Asynchronous FIFO RAM Port A Address *Width: [RADMX_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_tag_asynfifo_ram_addrb	*	O	RADM Tag Asynchronous FIFO RAM Port B Address *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
radmx_tag_asynfifo_ram_wea	1	O	RADM Tag Asynchronous FIFO RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_tag_asynfifo_ram_doutb		O	RADM Tag Asynchronous FIFO RAM Port B Data Out *Width: [RADMX_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
radmx_tag_asynfifo_ram_dina	*	I	RADM Tag Asynchronous FIFO RAM Port A Data In *Width: [RADMX_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: core_clk
mstr_tag_mgr_ram_addrb	*	O	Master Tag RAM Port A Address *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
mstr_tag_mgr_ram_addrb	*	O	Master Tag RAM Port B Address *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
mstr_tag_mgr_ram_wea	1	O	Master Tag RAM Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
mstr_tag_mgr_ram_doutb	*	O	Master Tag RAM Port B Data Out *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_tag_mgr_ram_dina	*	I	Master Tag RAM Port A Data In *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
Decomposer RAM Ports			
Notes:			
<ul style="list-style-type: none"> XADMX1 header and data RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True 			
slv_tag_mgr_ram_addrA	*	O	Slave Tag RAM Port A Address *Width: [SLV_DECOMP_TAG_DP_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_mgr_ram_addrB	*	O	Slave Tag RAM Port B Address *Width: [SLV_DECOMP_TAG_DP_PW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
slv_tag_mgr_ram_weA	1	O	Slave Tag RAM Port B Address Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_mgr_ram_doutB	*	O	Slave Tag RAM Port B Data Out *Width: [MAX_WIRE_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
slv_tag_mgr_ram_dina	*	I	Slave Tag RAM Port A Data In *Width: [MAX_WIRE_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_hdr_ram_addrA	*	O	XADMX1 Decomposer Header RAM Port A Address *Width: [XADMX_CLIENT1_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx1_decomp_hdr_ram_addrb	*	O	XADM1 Decomposer Header RAM Port B Address *Width: [XADMX_CLIENT1_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_hdr_ram_wea	1	O	XADM1 Decomposer Header RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_hdr_ram_doutb	*	O	XADM1 Decomposer Header RAM Port B Data Out *Width: [XADMX_CLIENT1_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_hdr_ram_dina	*	I	XADM1 Decomposer Header RAM Port A Data In *Width: [XADMX_CLIENT1_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_addrA	*	O	XADM1 Decomposer Data RAM Port A Address *Width: [XADMX_CLIENT1_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_addrB	*	O	XADM1 Decomposer Data RAM Port B Address *Width: [XADMX_CLIENT1_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_data_ram_wea	1	O	XADM1 Decomposer Data RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_doutb	*	O	XADM1 Decomposer Data RAM Port B Data Out *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx1_decomp_data_ram_dina	*	I	XADM1 Decomposer Data RAM Port A Data In *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
RADMX Asynchronous RAM Ports			
Notes:			
• RADMX asynchronous RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True			
xadmx0_decomp_hdr_ram_addrA	*	O	XADM0 Decomposer Header RAM Port A Address *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_hdr_ram_addrB	*	O	XADM0 Decomposer Header RAM Port B Address *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_hdr_ram_weA	*	O	XADM0 Decomposer Header RAM Write Enable *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_hdr_ram_doutB	*	O	XADM0 Decomposer Header RAM Port B Data Out *Width: [XADMX_CLIENT0_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_hdr_ram_dinA	*	I	XADM0 Decomposer Header RAM Port A Data In *Width: [XADMX_CLIENT0_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
xadmx0_decomp_data_ram_addrA	*	O	XADM0 Decomposer Data RAM Port A Address *Width: [XADMX_CLIENT0_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx0_decomp_data_ram_addrb	*	O	XADM0 Decomposer Data RAM Port B Address *Width: [XADMX_CLIENT0_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_data_ram_wea	1	O	XADM0 Decomposer Data RAM Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_data_ram_doutb	*	O	XADM0 Decomposer Data RAM Port B Data Out *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_data_ram_dina	*	I	XADM0 Decomposer Data RAM Port A Data In *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_hdr_ram_addrA	*	O	XADM Decomposer Header RAM Port A Address *Width: [RADMX_DECOMPOSER_HDRQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_hdr_ram_addrB	*	O	XADM Decomposer Header RAM Port B Address *Width: [RADMX_DECOMPOSER_HDRQ_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_hdr_ram_wea	1	O	XADM Decomposer Header RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_hdr_ram_doutb	*	O	XADM Decomposer Header RAM Port B Data Out *Width: [RADMX_DECOMPOSER_HDRQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_decomp_hdr_ram_dina	*	I	XADM Decomposer Header RAM Port A Data In *Width: [RADMX_DECOMPOSER_HDRQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_addrA	*	O	XADM Decomposer Data RAM Port A Address *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_addrB	*	O	XADM Decomposer Data RAM Port B Address *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_data_ram_weA	*	O	XADM Decomposer Data RAM Write Enable *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_doutB	*	O	XADM Decomposer Data RAM Port B Data Out *Width: [RADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_data_ram_dina	*	I	XADM Decomposer Data RAM Port A Data In *Width: [RADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
RADMX Asynchronous RAM Ports			
Note:			
• RADMX asynchronous RAMs are present when AHB_POPULATED=True and AHB_RAM_EXTERNAL=True and AHB_CLK_DIFF_ENABLE=True			
radmx_asyn_ram_addrA	I	O	RADM Asynchronous RAM Port A Address *Width: [FIFO_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table B-11 AHB External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_asyn_ram_addrb	*	O	RADM Asynchronous RAM Port B Address *Width: [FIFO_PW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
radmx_asyn_ram_wea	1	O	RADM Asynchronous RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_asyn_ram_doutb	*	O	RADM Asynchronous RAM Port B Data Out *Width: [FIFO_WD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
radmx_asyn_ram_dina	*	I	RADM Asynchronous RAM Port A Data In *Width: [FIFO_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk

B.4.6 AHB Bridge SII Interface Addition

The native PCIe core has an SII interface that varies from configuration to configuration. Because the native PCIe core is instantiated together with the AHB bridge core, the top-level SII signals for the AHB bridge will also vary from configuration to configuration. Refer to “[System Information Interface \(SII\)](#)” for details on the PCIe/AHB bridge product’s top-level I/O signals, other than those signals described within this appendix. All SII control signals (inputs of top level PCIe/AHB IP) can be tied to the level that is required in the default setting of the native PCIe core’s signal. All SII status signals (outputs of the top level PCIe/AHB IP) can be ignored if the application does not require these signals.

Most of the SII signals are not necessary for a basic AHB configuration. Refer to “[System Information Interface \(SII\)](#)” when the application has special requirements such as additional power management control, advanced PCIe feature enabled, MSIX additional support, etc.

The following describes typical SII signals within a combined PCIe/AHB IP.

The lookup error signals are used to notify the application that there is an overflow that occurred in a lookup table of either inbound or outbound responses. These indicate that there was a violation for the number of outstanding Non-Posted requests issued for the inbound or outbound direction.

- ❖ gm_cmposer_lookup_err
- ❖ radmx_cmposer_lookup_err

B.5 Parameters

The AHB Bridge Module is populated based on the functions you select with the coreConsultant configuration parameters. [Table B-12](#) defines these configuration parameters.

Table B-12 Configuration Parameters

coreConsultant Option Name	Description
AHB Enable	Function: Enables the AHB Bus Interface to the native PCI core Value Range: On, Off Default Value: Off Parameter Name: AHB_POPULATED
Slave DBI Enable	Function: Enables the AHB Slave DBI interface Value Range: On, Off Default Value: On Parameter Name: DBI_4SLAVE_POPULATED
Master Interface Enable	Function: Enables the AHB Master interface Value Range: On, Off Default Value: On Parameter Name: MASTER_POPULATED
Enable Independent AHB Clock	Function: Enable an independent clock (other than core_clk) for the AHB bus. Value Range: On, Off Default Value: On Parameter Name: AHB_CLK_DIFF_ENABLE
Slave and DBI Slave Response Mode	Function: Selects Normal or SPLIT Response for Slave & DBI Slave Value Range: Selectable, Normal and SPLIT Default Value: Normal Parameter Name: CC_RESPONSE_MODE
AHB Slave and DBI ERROR Response	Function: Enables Immediate ERROR Response from the Slave & DBI Slave on illegal AHB access Value Range: False, True Default Value: False Parameter Name: RETURN_ERR_RESP
AHB Bus Endianness	Function: Selects endianness for the AHB interface Value Range: Selectable, Static Little and Static Big Default Value: Static Little Parameter Name: BIG_ENDIAN
Maximum Master Tags Supported	Function: Specifies the maximum number of transfers supported by the AHB bridge when performing as an AHB master NOTE: A higher value chosen here results in a higher gate count. Value Range: 2, 4, 8, 16, 32, 64, 128, 256 Default Value: 32 Parameter Name: CC_MAX_MSTR_TAG

Table B-12 Configuration Parameters (Continued)

coreConsultant Option Name	Description
Remote Device Max Read Request Size	<p>Function: Specifies the maximum read request size supported by the PCIe remote device when AHB or AHB is enabled. This parameter is used to size the master composer memories inside the AHB adapter. This is a visible parameter only when master interface is enabled.</p> <p>Value Range: 128, 256, 512, 1k, 2k, 4k</p> <p>Default Value: 128</p> <p>Parameter Name: CX_REMOTE_RD_REQ_SIZE</p>
Master Page Boundary Size	<p>Function: Specifies the page boundary size supported within the AHB interconnect when the AHB bridge is performing as a master. No packet can have an address that crosses this boundary. Packets will be split to conform to this requirement.</p> <p>Value Range: 128, 256, 512, 1k, 2k, 4k</p> <p>Default Value: 4k</p> <p>Parameter Name: CC_MSTR_PAGE_BOUNDARY_PW</p>
Master Data Width	<p>Function: Specifies the data bus width of the AHB master interface.</p> <p>Value Range: 32*</p> <p>Default Value: 32</p> <p>Parameter Name: CC_MSTR_BUS_DATA_WIDTH</p> <p>*Contact Synopsys for support for additional data widths (up to 256 bits)</p>
Master Address Width	<p>Function: Specifies the address bus width of the AHB master interface.</p> <p>Value Range: 32*</p> <p>Default Value: 32</p> <p>Parameter Name: CC_MSTR_BUS_ADDR_WIDTH</p> <p>*Contact Synopsys for support for additional address widths (up to 64 bits)</p>
Master Burst Length Width	<p>Function: Specifies the maximum burst length of an AHB transfer at the master interface. This parameter controls the resources that will be assigned inside the AHB bridge for maximum transfer burst length. If undefined length is supported, then this parameter must be set for the maximum of any AHB transfer under undefined length burst. This parameter has the size in burst cycle.</p> <p>Value Range: 16 to 256</p> <p>Default Value: 16</p> <p>Parameter Name: CC_MSTR_BURST_LEN</p>
Master Request Header FIFO Q Depth	<p>Function: Specifies AHB Master Request Header FIFO Queue size. Refer to Figure 7 for architecture details of the FIFO. This FIFO is designed for inbound request to be enqueued into AHB bridge.</p> <p>Value Range: 4 to 260</p> <p>Default Value: 4</p> <p>Parameter Name: CC_RADMX_DECOMPOSER_HDRQ_DP</p>

Table B-12 Configuration Parameters (Continued)

coreConsultant Option Name	Description
Master Request DATA FIFO Q Depth	<p>Function: Specifies AHB Master Request Data FIFO Queue size. Refer to Figure 7 for architecture details of the FIFO. This FIFO is designed for inbound request to be enqueued into AHB bridge.</p> <p>Value Range: Unlimited integer</p> <p>Default Value: 34</p> <p>Parameter Name: CC_RADMX_DECOMPOSER_DATAQ_DP</p>
Slave Interface Enable	<p>Function: Enables AHB Slave Interface</p> <p>Value Range: On, Off</p> <p>Default Value: On</p> <p>Parameter Name: SLAVE_POPULATED</p>
Maximum Slave Tags Supported	<p>Function: Specifies the maximum number of transfers supported by the AMBA Slave. For AHB, this is the number of Masters that will access the Slave.</p> <p>Value Range: 1 to 256</p> <p>Default Value: 16</p> <p>Parameter Name: CC_MAX_SLV_TAG</p>
Slave Data Width	<p>Function: Specifies the data bus width of the AHB slave interface.</p> <p>Value Range: 32*</p> <p>Default Value: 32</p> <p>Parameter Name: CC_SLV_BUS_DATA_WIDTH</p> <p>*Contact Synopsys for support for additional data widths (up to 256 bits)</p>
Slave Address Width	<p>Function: Specifies the address bus width of the AHB slave interface.</p> <p>Value Range: 32*</p> <p>Default Value: 32</p> <p>Parameter Name: CC_SLV_BUS_ADDR_WIDTH</p> <p>*Contact Synopsys for support for additional address widths (up to 64 bits)</p>
Slave Burst Length Width	<p>Function: Specifies the maximum burst length of an AHB transfer at the slave interface. This parameter controls the resources that will be assigned inside the AHB bridge for maximum transfer burst length. If undefined length is supported, then this parameter must be set for the maximum of any AHB transfer under undefined length burst. This parameter has the size in burst cycle.</p> <p>Value Range: 16 to 256</p> <p>Default Value: 32</p> <p>Parameter Name: CC_SLV_BURST_LEN</p>

Table B-12 Configuration Parameters (Continued)

coreConsultant Option Name	Description
Max Number of Masters	<p>Function: Indicates the number of masters that are within AHB interconnect. This is used to support split response.</p> <p>Value Range: 1 to 16</p> <p>Default Value: 1</p> <p>Parameter Name: CC_SLV_NUM_MASTERS</p>
Slave Request Header FIFO Q Depth	<p>Function: Specifies the AHB bridge slave request Header FIFO size. Refer to Figure 8 for architecture details of the FIFO. This FIFO is designed for outbound request to be enqueued into the AHB bridge.</p> <p>Value Range: 4 to 260</p> <p>Default Value: 16</p> <p>Parameter Name: CC_XADMX_CLIENT1_QUEUE_HDP</p>
Slave Request DATA FIFO Q Depth	<p>Function: Specifies the AHB bridge slave request Data FIFO size. Refer to Figure 8 for architecture details of the FIFO. This FIFO is designed for outbound request to be enqueued into the AHB bridge.</p> <p>Value Range: Unlimited integer</p> <p>Default Value: 32</p> <p>Parameter Name: CC_XADMX_CLIENT1_QUEUE_DDP</p>

B.6 Implementation Guidelines

The implementation guidelines presented in this chapter are intended as a guide through a basic implementation of a PCIe IP and AHB bridge instantiation. The reference design demonstrates a basic application where the AHB bridge, the DWC PCIe IP core, and Synopsys PHY are instantiated. The example RTL code is not intended to be part of the completed/validated DWC PCIe IP product offering.

Configuration parameters are discussed in [Section B.6.4 on page 738](#), and RAMs in [Section B.6.5 on page 741](#).

B.6.1 Scope of the Implementation Guidelines

A structural chip-level reference design is provided to facilitate building your PCIe device(s). The structural reference design contains the instantiation and interconnections of the DWC PCIe AHB bridge, IP core, PHY, and additional reference design modules. These other reference design modules demonstrate how to build a PCIe device in a quick and straight-forward fashion. The chip-level reference design provides a basic example of how to drive and monitor the DWC PCIe bridge IP core inputs and outputs.

The example chip top RTL (`chip_top.v`) is located under `<config*>/src/customer/generic/`). The chip top RTL illustrates an example hierarchy for a PCIe device with the PCIe bridge IP core, PHY, and AHB bridge instantiated. There are several example RTL modules other than the IP core that are also provided to support the PCIe bridge IP core integration into a PCIe device. We have also made a number of basic assumptions which are applicable to most applications, and are necessary in order for us to configure and interconnect the AHB bridge, PCIe core and PHY.

These assumptions are as follows:

1. The inbound and outbound AHB transfers are all memory transfers within AMBA memory space. The application intends to use the address translation feature offered by the PCIe core to map AMBA memory regions onto PCIe configuration, IO, message and memory regions. In other words, the application does not intend to use our sideband AHB signals for PCIe Type and Format translations.
2. The application supports both MSI and legacy interrupts via the PCIe core MSI and legacy interrupt interfaces. This means that an MSI and legacy interrupt controller is directly connected to the PCIe core's MSI and legacy interrupt interfaces. The application is responsible to generate the `app_intr_req` and `app_intr_clr` signals. Note, that this assumption requires that no order relation exists between the application AHB transfers and the interrupt event. If ordering is required, the application can modify the outbound address translation module to get MSI requests to properly map into an AMBA memory region.
3. All outbound messages are supported through the address translation provided in the reference design. The PCIe core Vendor Message interface is not activated. All inbound messages are handled by the PCIe core except for Vendor Messages, which are handled by the inbound address translation logic. Inbound Vendor Messages will be translated into AHB memory writes.
4. The Slave DBI interface is activated in this reference design. If an application desires to have a shared DBI slave interface, it will just need to configure the AHB bridge setting for Shared DBI.
5. The External Local Bus Interface (ELBI) is used to design customer-specific registers and reference design-related registers. If the application has its own register access mechanism, it will need to modify the `elbi_driver.v` to accommodate that. This reference design has been set up to use the PCIe core ELBI interface to access extended application registers and registers designed for the reference design.

Since most signals of the PCIe bridge IP are heavily dependent on application-specific functionality, the example RTL modules are there to provide comments on key issues of the core interfaces, and a typical example of how to use them. We expect the customer to modify any or all of the example modules, as needed.

With the above assumptions in mind, this reference design comprises the following major modules:

- ❖ **ahb_master_driver.v**: A generic AHB master driver to drive the master response AHB bridge interface
- ❖ **ahb_master_sideband_driver.v**: A master Sideband driver to drive the AHB bridge master response interface sideband signals
- ❖ **ahb_slave_sideband_driver.v**: A slave Sideband driver to drive the AHB bridge slave interface sideband signal
- ❖ **ahb_slave_driver.v**: A generic AHB slave driver to drive the AHB bridge slave interface
- ❖ **ob_addr_translation_driver.v**: An outbound address translation driver to accomplish the outbound address translation from AMBA memory space to PCIe memory space or PCIe transfer type, such as IO, configuration and message type
- ❖ **ib_addr_translation_driver.v**: An Inbound address translation driver to accomplish the address translation from PCIe memory space, type and format to AMBA memory space
- ❖ **clk_RST.v**: A clock and reset module to drive the clocks and resets that are required by PCIe PHY, DWC PCIe core, and AHB bridge.
- ❖ **pm_ctrl_driver.v**: An extended power management control module to drive the extended power management functionalities the application desires.

All inputs and outputs of the PCIe core and AHB bridge are set or monitored through the reference design based on the above assumptions and modules. This will greatly ease the IP integration effort. Customers need only to concentrate on the connections of the generic (standard) AHB master and slave drivers in this reference design, and application-specific functionalities. The PCIe related functions are completely handled and the interface is simplified to just the SerDes IO pins at this chip-top design.

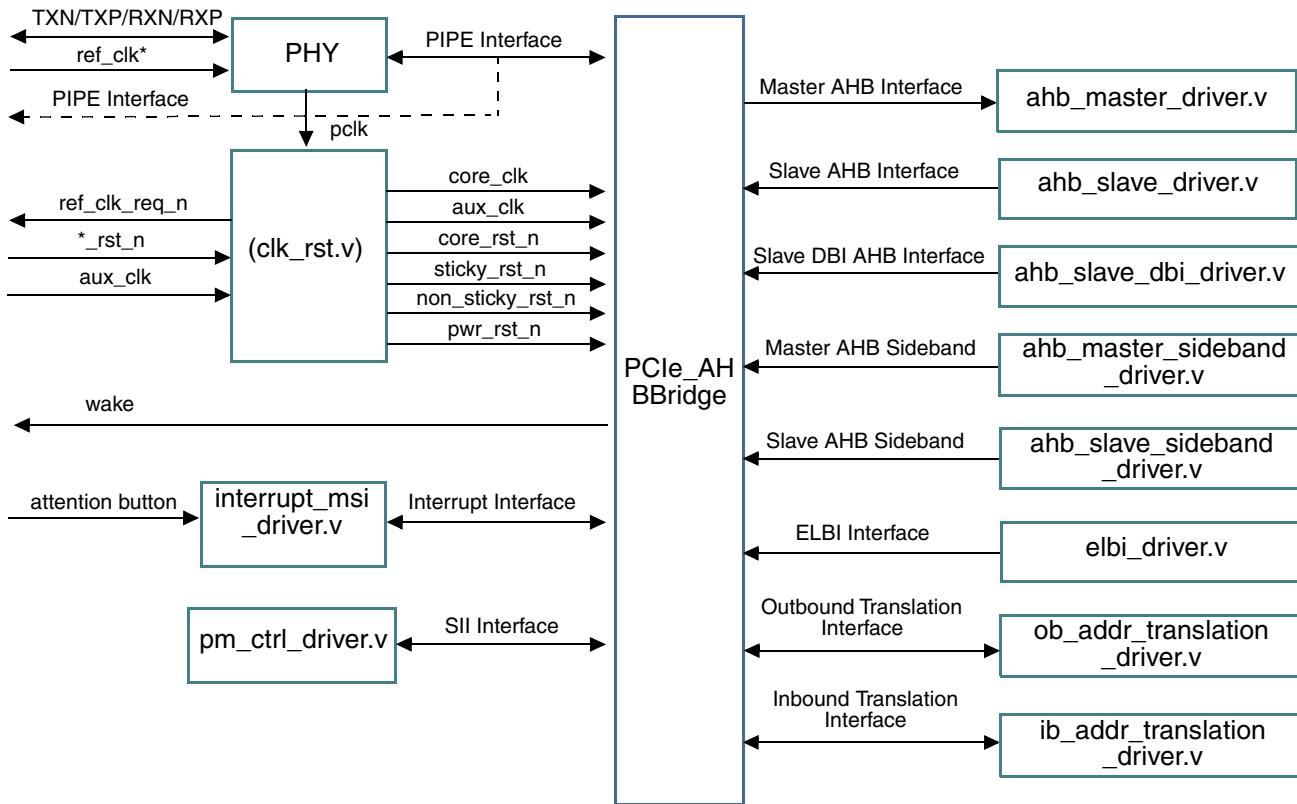
B.6.2 Detailed Descriptions of the Structural Chip-Level Reference Design

The following sections provide an overview of the reference design and then a brief description of each of the major components. The reference code itself is intended to provided additional details on the requirements of each interface and its associated functionality.

B.6.2.1 Chip Top Block Diagram

The architecture is designed to have example modules instantiated together with the IP core, as shown in Figure B-12.

Figure B-12 chip_top.v Architecture



Module functions are as follows:

- ❖ **PCIE_AHB_Bridge**: The configured AHB bridge IP core. It contains the PCIe core and AHB bridge.
- ❖ **PHY**: This is a module for either the SNPS PHY model or another Pipe PHY model configured by the user.
- ❖ **Clock and Reset Control (clk_RST.v)**: Provides an example of how to drive the clocks and resets expected by the core, PHY and AHB bridge.
- ❖ **MSI and Legacy Interrupt Control (interrupt_msi_driver.v)**: Provides an example of how to initiate a legacy interrupt or an MSI request.
- ❖ **PM MISC Control (pm_ctrl_driver.v)**: Provides an example of how to drive the power management inputs that are designed to enable the application logic to have more control over the low-power states of the core.

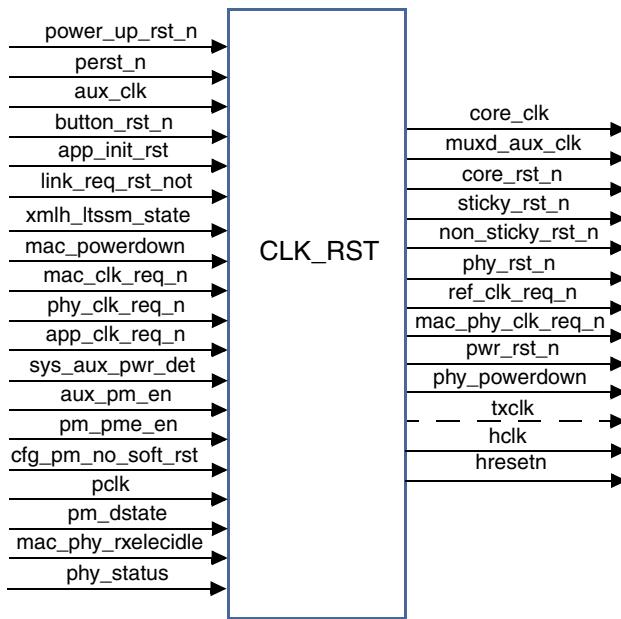
- ❖ **Master AHB Interface Driver (`ahb_master_driver.v`):** Enables customers to build a response driver to the DWC PCIe AHB bridge master interface. This driver is connected directly to the response portion of the DWC PCIe AHB bridge master interface. (This module acts like an AMBA slave to the DWC PCIe AHB bridge.)
- ❖ **Slave AHB Interface (`ahb_slave_driver.v`):** Enables customers to build an AMBA master. This driver is connected directly to the DWC PCIe AHB bridge slave interface. (This module acts like an AMBA master to the DWC PCIe AHB bridge.)
- ❖ **Master AHB Sideband Interface (`ahb_master_sideband_driver.v`):** Provides a driving capability for the DWC PCIe AHB bridge master interface sideband signals to accommodate the extra information associated with a master response. This driver is connected directly to the DWC PCIe AHB bridge master interface sideband signals.
- ❖ **Slave AHB Sideband Interface (`ahb_slave_sideband_driver.v`):** Provides a driving capability for the DWC PCIe AHB bridge slave interface sideband signals to accommodate the extra information associated with a master request. This driver is connected directly to the DWC PCIe AHB bridge slave interface sideband signals.
- ❖ **Application Registers (`elbi_driver.v`):** Provides a reference design to drive the ELBI interface to access the registers required for the reference design. Note, that the application can modify this module to replace these registers with their own application-specific registers.
- ❖ **Outbound Address Translation (`ob_addr_translation_driver.v`):** Provides the address translation on outbound transfers. It maps the AMBA memory region to a PCIe memory region, PCIe Configuration request, IO request, or Message.
- ❖ **Inbound Address Translation (`ib_addr_translation_driver.v`):** Provides the address translation on inbound transfers. It maps an incoming PCIe transfer onto the desired AMBA memory region.

B.6.2.2 Clock and Reset Control Module

The Clock and Reset Control module (`clk_rst.v`) is an example RTL module driving the core's clock and reset inputs for a typical application. This module primarily covers the following functions:

- ❖ The core's clock and reset control
- ❖ AUX clock control
- ❖ The PHY's reset
- ❖ The AHB interface clock and reset
- ❖ Generation of the external reference clock request signal

[Figure B-13](#) illustrates the I/O requirement of this reference module.

Figure B-13 Clock and Reset Controller IO

There are several core output signals that affect the clock and reset circuitry of a PCIe device due to the power state or link status of the IP core.

The inputs to the clock and reset module are from four sources:

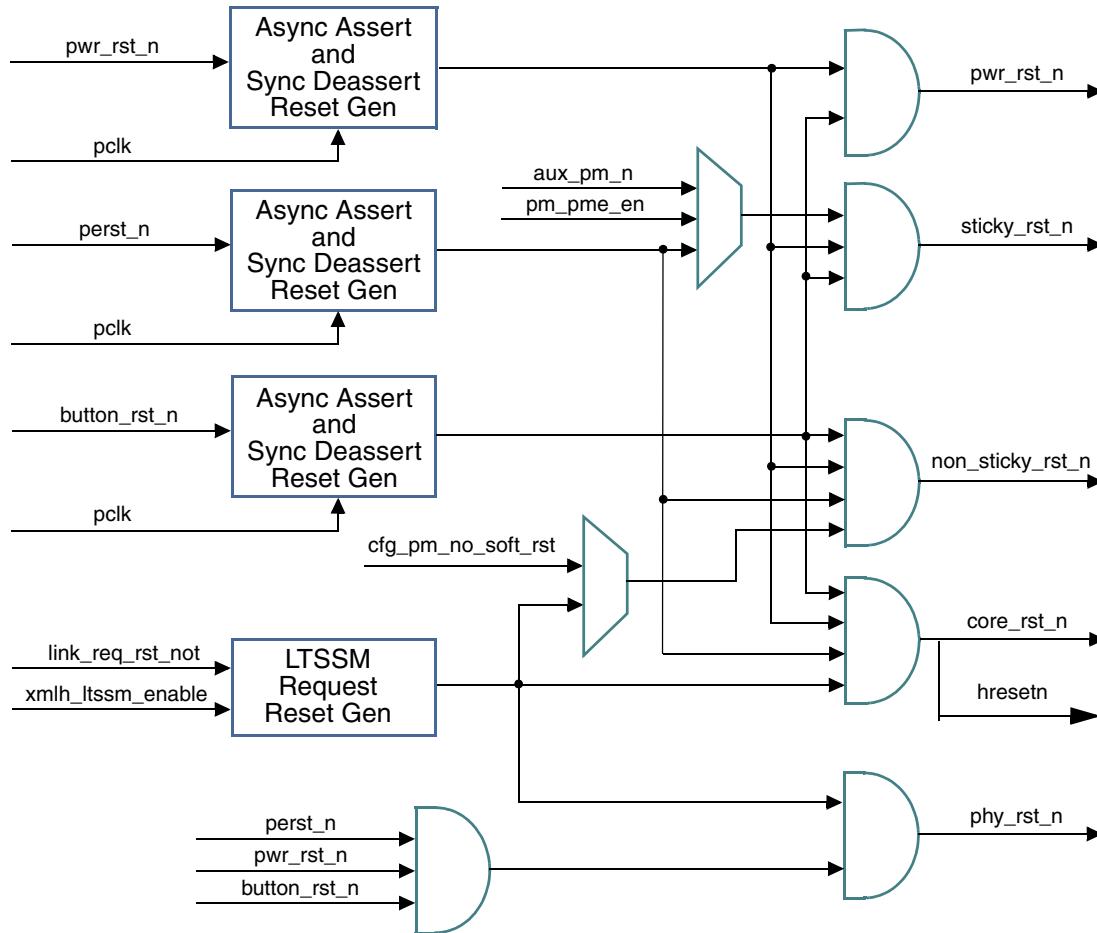
- ❖ Chip inputs: power_up_rst_n, perst_n, aux_clk, button_rst_n, and sys_aux_pwer_det.
- ❖ Inputs from the IP core: link_req_rst_not, xmlh_ltssm_state, mac_powerdown, mac_clk_req_n, aux_pm_en, pm_pme_en, cfg_pm_no_soft_rst, mac_phy_rxidle, and pm_dstate.
- ❖ Inputs from the application's internal logic within the chip: app_clk_req_n, app_ltssm_enable, and app_init_rst.
- ❖ Inputs from the PHY module: pclk, phy_status, and phy_clk_req_n.

The outputs are driven to four destinations:

- ❖ To the IP core: pwr_rst_n, core_clk, core_rst_n, sticky_rst_n, non_sticky_rst_n, and muxd_aux_clk.
- ❖ To the I/O pins of the chip: ref_clk_req_n and txclk.
- ❖ To the PHY module: phy_rst_n, phy_powerdown, and mac_phy_clk_req_n.
- ❖ To AHB bridge and AHB application logic: hclk, hresetn.

The Clock and Reset Control module generates all clock and reset signals required by the core IP. Reset control architecture is illustrated in [Figure B-14](#). Clock control architecture is illustrated in [Figure B-15](#).

Figure B-14 Reset Control

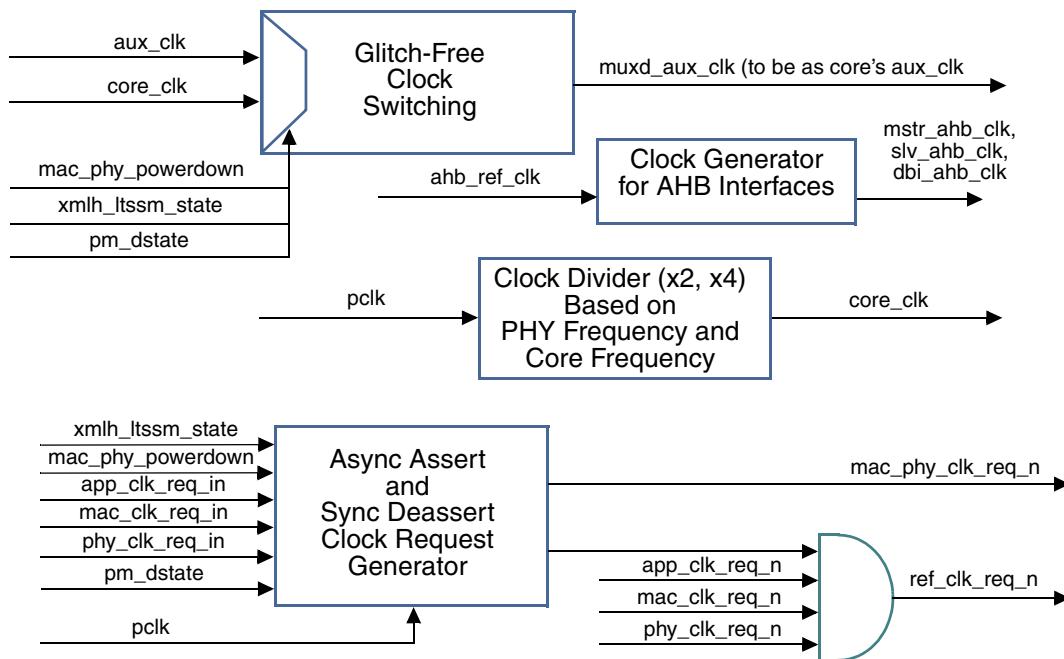


The PCIe core reset inputs can be asynchronously asserted, but must be synchronously deasserted with respect to the core_clk input. The reset implementation should be compliant with DFT methodology. It is recommended that the application logic and the core be reset simultaneously. There are two reset domains in the core logic. In general, `core_rst_n` resets the logic domain powered by main power; `pwr_rst_n` resets the logic domain powered by aux power. The `core_rst_n` signal should be asserted any time `pwr_rst_n` is asserted.

The sticky and non-sticky resets are routed to the CDM.

- ❖ `sticky_rst_n`: Resets registers marked as sticky registers in the PCI Express Base 2.0 Specification.
- ❖ `non_sticky_rst_n`: Resets the rest of the configuration registers not marked as sticky.

The `phy_rst_n` signal is used to reset the PIPE compliant PHY and is allowed to be completely asynchronous.

Figure B-15 Clock Control

Note

- The Clock Control diagram shows the major features of an example clock design to support a typical PCIe application. The aux_clk input is designed to support a low-speed clock that may be used to minimize power when operating in a low-power mode. If the AHB clock is running in sync with the core clock, then all AHB clocks should be sourced from the core clock. Otherwise, the application should generate an AHB clock for the AHB master and slave based on an additional reference clock.
- The bulk of the core logic runs on the core_clk domain. The core_clk output is derived from the pclk output of the PHY. This clock may need to be divided (as shown) in cases where the core is running at a fraction of the pclk rate.
- A small amount of logic within the core needs a clock that is always available, even in low-power states. This logic is placed on the muxd_aux_clk domain. This clock is equivalent to core_clk during normal operation, but may be switched over to a low-speed auxiliary clock when core_clk is removed. This switching is performed with glitch-free clock switching logic in the reference design.
- The ref_clk_req_n output signal can be used to indicate to the system when a reference clock is required. By setting this signal to 1'b1, the device can indicate to the system that refclk may be removed. This can be used to further reduce power. The Synopsys PCIe PHY provides additional support for the ref_clk_req_n feature, and the example design includes this connectivity as well.
- Please refer to clk_rst.v for more details.

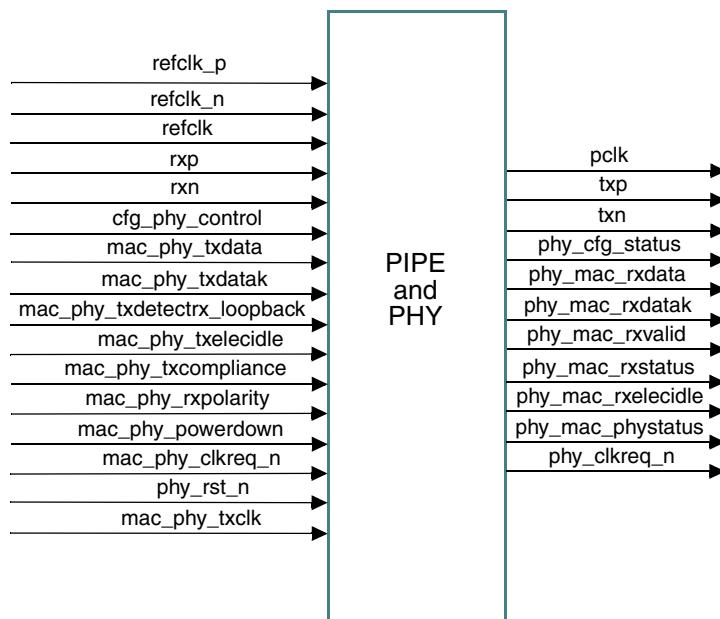
B.6.2.3 PIPE PHY Module

A standard PIPE PHY module for PHY implementation has been introduced. Both a generic PHY model for simulation-only purposes, and a Synopsys PHY model are available for your chip implementation. The PIPE PHY module is illustrated in [Figure B-16](#).

The example design instantiates this standard PIPE PHY module. To use the generic PHY or the Synopsys PHY, all that is required is selecting the appropriate PHY during the coreConsultant configuration. To select an alternate PHY, the intent is to make a wrapper (an example wrapper is provided) to map the desired

PHYS I/O to the standard PHY interface. This should be primarily signal renaming if the PHY provides a standard PIPE interface. A few additional parameters control optional features, such as selecting a differential clock. These can be determined from the example files provided. When configuring the core with the intent to use a custom PHY, select "Custom" from the PHY pull-down and follow the instructions provided in the <workspace>/src/Phy/custom/README.txt file to integrate your PHY model with the core.

Figure B-16 PIPE PHY Module



refclk_p and refclk_n:

Reference clock differential pair from device pins.

refclk:

Reference clock (non differential) from device pins. Only one of these reference clocks will typically be used by the PHY.

rxp and rxn:

Differential receive pair inputs from device pins.

txp and txn:

Differential transmit pair outputs to device pins.

cfg_phy_control:

Input to the PHY from the IP core to allow some external register control over the PHY internals. This is an optional interface and may be ignored.

phy_cfg_status:

Output from PHY to the core to indicate some of the internal status of the PHY. This is an optional interface and may be ignored.

mac_phy_clkreq_n:

Input to the PHY from the IP core and application logic to indicate that the MAC and application logic are ready to get the core clock removed. Supported by the Synopsys PHY.

phy_clkreq_n:

Output from the PHY to indicate that it is ready for the reference clock to be removed. Supported by the Synopsys PHY.

phy_RST_n:

Input from the clock reset control module to indicate that it desires to reset the PHY.

pclk:

Output from the PHY to supply the PIPE clock.

mac_phy_*:	Inputs to the PHY from the core. These inputs are standard PIPE signals. Please refer to the PIPE specification for information about these signals.
phy_mac_*:	Outputs from the PHY to the core. These outputs are standard PIPE signals. Please refer to the PIPE specification for information about these signals.
mac_phy_txclk:	This signal is present if your PHY is a PXPIPE PHY, and provides a source-synchronous transmit clock for data moving from the MAC to the PHY.

B.6.2.4 Master and Slave AHB Driver Modules

The Master and Slave AHB Driver modules (`ahb_master_driver.v` and `ahb_slave_driver.v`) are designed as a place holder for customers to connect an AHB master, an AHB slave, and optionally, an AHB DBI master to the DWC PCIe AHB bridge.

Applications may have an AHB interconnect module loaded into the reference design chip top to replace these modules. Or, an application may follow the hierarchy that the chip top offers to design master and slave drivers within these modules (`ahb_master_driver.v` and `ahb_slave_driver.v`)

The design for these modules should contain an AHB master, a DBI AHB master, and an AHB slave. The interfaces at these modules are all standard AHB interfaces, as specified in the AMBA AHB specification.

These modules are empty aside from the module instantiation. It is fully expected that customers modify and replace these modules based on their applications.

B.6.2.5 Master Sideband AHB Driver Module

The Master Sideband AHB Driver module (`ahb_master_sideband_driver.v`) drives the sideband signals needed to form a response to a master request issued from the DWC PCIe AHB bridge.

This reference design achieves two basic functions. One is to drive the response channel sideband signals such as `mstr_bmisc_info`, `mstr_rmsic_info`, and `mstr_resp_err_map`.

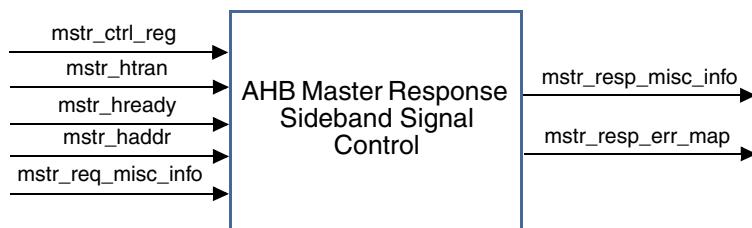
The response channel sideband signals are controlled via registers. These registers allow the application to issue the requests with special PCIe header information. The `mstr_bmisc_info` signal is controlled by `mstr_bctrl_reg`. The `mstr_rmsic_info` signal is controlled by `mstr_rcctrl_reg`, and the `mstr_resp_err_map` signal is controlled by `mstr_ctrl_reg`. Additional information is provided in the ELBI Driver module section.

The outputs of this module are identified as follows:

- ❖ output [MSTR_RESP_MISC_INFO_WD -1:0] `mstr_resp_misc_info`: AHB interface write response channel sideband signal output
- ❖ output [RESP_ERRBUS_WD -1:0] `mstr_resp_err_map`: Output to control the error response mapping

AHB Master Sideband Control architecture is shown in [Figure B-17](#).

Figure B-17 AHB Master Sideband Control



B.6.2.6 Slave Sideband AHB Driver Module

The Slave Sideband AHB Driver module (`ahb_slave_sideband_driver.v`) drives the sideband signals required for an AHB master portion of an application to drive the DWC PCIe AHB bridge. These sideband signals include `slv_awmisc_info`, `slv_armsic_info`, `slv_wmisc_info` and `slv_resp_err_map`.

The DWC PCIe bridge slave sideband signals can be controlled via registers. These registers enable the application to issue the requests with special PCIe header information. The `slv_awmisc_info` signal is controlled by `slv_wctrl_reg`, `slv_armsic_info` by `slv_rctrl_reg`, and `slv_resp_err_map` by `slv_ctrl_reg`.

The outputs of this module are described as follows:

- ❖ output [SLV_MISC_INFO_WD -1:0] `slv_req_misc_info`: sideband signal of AHB slave write channel
- ❖ output [RESP_ERRBUS_WD -1:0] `slv_resp_err_map`: Error response map controlled by application

Slave Sideband AHB Driver architecture is shown in [Figure B-18](#).

Figure B-18 AHB Slave Sideband Control



B.6.2.7 Inbound Address Translation Driver Module

The Inbound Address Translation Driver module translates an inbound PCIe transaction onto an AHB master memory transfer.

The reference design can perform the following address translation:

1. An inbound memory RD/WR that falls into BAR0 of a configured range will be translated as an AHB memory RD/WR transfer. There are two AMBA memory regions defined in this module. A PCIe memory BAR0 region can be mapped into two (or more) AMBA memory regions if an application has this requirement.
2. An inbound IO RD/WR that falls in BAR1 of a configured PCIe range will be translated as an AHB memory RD/WR transfer in a defined AMBA memory region.
3. An inbound vendor message request will be translated as an AHB memory write transfer in a defined AMBA memory region.
4. An inbound memory RD/WR that falls in an extended ROM BAR will be translated as an AHB memory RD/WR transfer at a defined AMBA memory region.

A total of five AMBA memory regions are defined in this module. These are identified as follows:

Starting Address

- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim0_mem_addr_start: First internal memory region (where BAR0 of PCIe transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim1_mem_addr_start: Second internal memory region (where BAR0 of PCIe transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_io_addr_start: Internal IO region (where BAR0 of PCIe IO transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_msg_addr_start: Internal message region
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_rom_addr_start: Internal ROM region (where ROM BAR of PCIe transaction will map to)

Address Limit

- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim0_mem_addr_limit: First internal memory region
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim1_mem_addr_limit: Second internal memory region
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_io_addr_limit: Internal IO region
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_msg_addr_limit: Internal message region
- ❖ input [`MASTER_BUS_ADDR_WIDTH - 1:0] pim_rom_addr_limit: Internal ROM region

Note that all of these regions are controlled via the registers defined in the ELBI driver module. Please refer to "[ELBI Driver Module](#)" for more information.

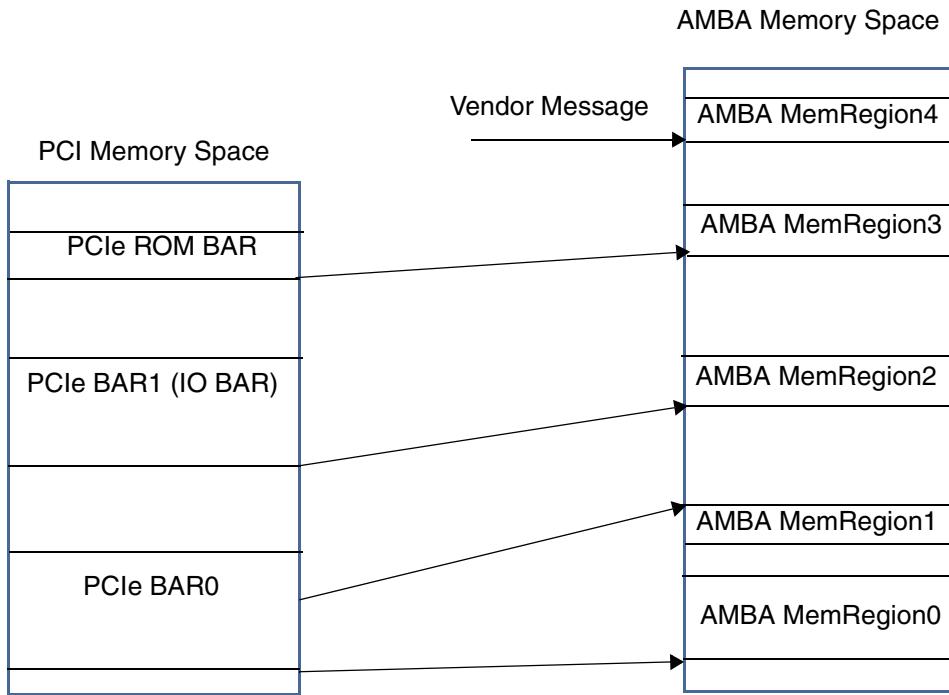
The outputs of this module are described as follows:

- ❖ output [63:0] rtranslated_addr_in_d: The translated address based on the regions defined above
- ❖ output [1:0] rtranslated_err_in_d: Error indicates the status of the translation. 00 -- Normal, 01 -- UR, 10 -- CA, 11-- Undefined (not allowed)

When an address matches a BAR, but fails to match an AMBA memory region, an error response will be sent to the PCIe core as a response.

Inbound Address Translation Driver architecture is shown in [Figure B-19](#).

Figure B-19 Inbound Address Translation



B.6.2.8 Outbound Address Translation Driver Module

The Outbound Address Translation Driver module translates an AHB slave memory transfer into a PCIe transaction. The reference design can perform the following address translation:

1. Two AMBA memory regions are translated onto two different PCIe memory regions. This demonstrates a multiple-function device with two independent functions requiring two memory regions.
2. One AMBA memory region is defined to support an outbound AHB transfer translated into an outbound PCIe IO transaction.
3. Two AMBA memory regions are defined to support an outbound AHB transfer translated into an outbound PCIe configuration transaction. Each of these AMBA memory regions can take care of the PCIe configuration type0 and type1 translation.
4. One AMBA memory region is defined to support an outbound AHB transfer translated into an outbound PCIe message transaction.

There are total of six AMBA memory regions defined in this module. These are identified as follows:

Starting Address

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] pom0_mem_addr_start: PCIe out memory space (where the address of the internal request is in region0)
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] pom1_mem_addr_start: starting address of the PCIe out memory space (where the address of the internal request is in region1)
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in0_mem_addr_start: Internal memory region 0

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in1_mem_addr_start: Internal memory region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_io_addr_start: Internal IO region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg0_addr_start: Internal CFG0 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg1_addr_start: Internal CFG1 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_msg_addr_start: Internal MSG region

Address Limit

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in0_mem_addr_limit: Internal memory region 0
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in1_mem_addr_limit: Internal memory region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_io_addr_limit: Internal IO region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg0_addr_limit: Internal CFG0 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg1_addr_limit: Internal CFG1 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_msg_addr_limit: Internal MSG region

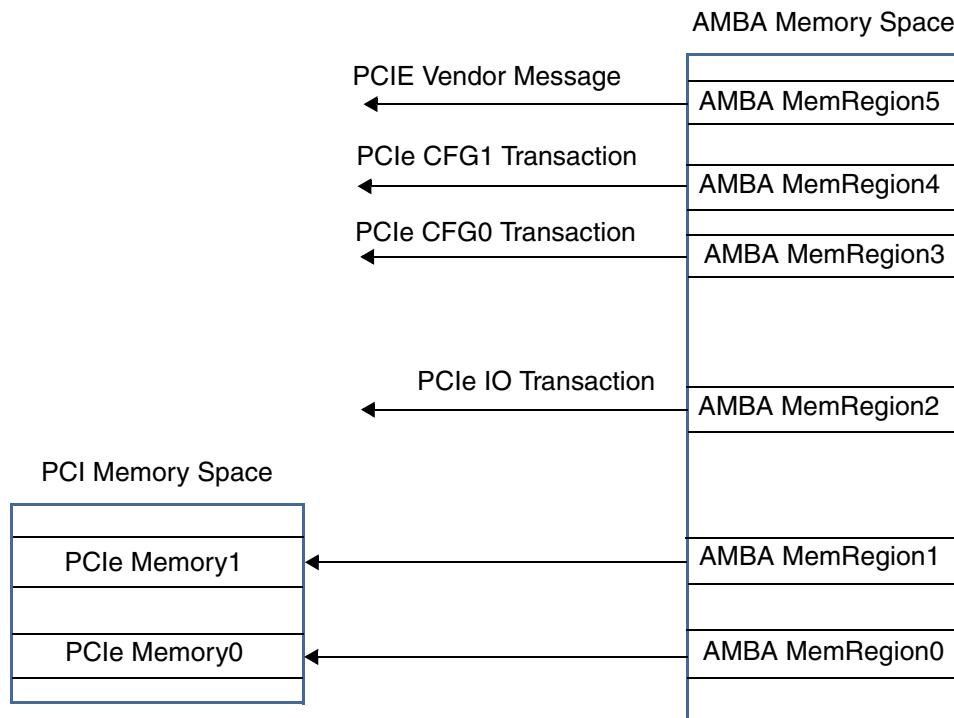
Note that all of these regions are controlled via the registers defined in the ELBI driver module. Please refer to “[ELBI Driver Module](#)” for more information.

The outputs of this module are identified as follows:

- ❖ output [63:0] xtranslated_addr_in_d: The translated address of a PCIe transaction
- ❖ output [4:0] xtranslated_type_in_d: The translated type of a PCIe transaction
- ❖ output [7:0] xtranslated_func_in_d: The translated function number of a PCIe transaction
- ❖ output [7:0] xtranslated_msgcode_in_d: The translated message code of a PCIe transaction
- ❖ output xtranslate_enable: The enable to the translation. When no matching is detected, the enable is off.

Inbound Address Translation Driver architecture is shown in Figure B-20.

Figure B-20 Outbound Address Translation



B.6.2.9 ELBI Driver Module

The ELBI Driver module accesses all registers required to support the reference design. It can be modified by the customer to support application-specific registers (such modifications beyond the scope of this document).

All registers within this module can be accessed by an application AHB master via the slave DBI interface of the AHB bridge (by a common slave interface, if this option is utilized).

Registers designed to support the reference design are identified as follows:

Table B-13 Inbound AMBA Memory region0 Start Address: Address Offset 0 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region0 for inbound address translation

Table B-14 Inbound AMBA Memory region0 Limit: Address Offset 0x08 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region0 for inbound address translation

Table B-15 Inbound AMBA Memory region1 Start Address: Address Offset 0x10 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region1 for inbound address translation

Table B-16 Inbound AMBA Memory region1 Limit: Address Offset 0x18 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region1 for inbound address translation

Table B-17 Inbound AMBA Memory region2 Start Address: Address Offset 0x20 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region2 for inbound address translation of IO transaction

Table B-18 Inbound AMBA Memory region2 Limit: Address Offset 0x28 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region2 for inbound address translation of IO transaction

Table B-19 Inbound AMBA Memory region3 Start Address: Address Offset 0x030 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region3 for inbound address translation of PCIe Expansion ROM address space

Table B-20 Inbound AMBA Memory region3 Limit: Address Offset 0x038 of DBI Transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region3 for inbound address translation of PCIe Expansion ROM address space

Table B-21 Starting Address of Outbound PCIe Memory Transaction for AMBA Memory region 0: Address Offset 0x50 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of PCIe memory RD/WR transaction translated from an AMBA outbound memory region0

Table B-22 Starting Address Of Outbound Pcie Memory Transaction For Amba Memory region1: Address Offset 0x60 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of PCIe memory RD/WR transaction translated from an AMBA outbound memory region1

Table B-23 Outbound AMBA Memory region0 Start Address: Address Offset 0x70 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region0

Table B-24 Outbound AMBA Memory region0 Limit: Address Offset 0x78 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region0

Table B-25 Outbound AMBA Memory region1 Start Address: Address Offset 0x80 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region1

Table B-26 Outbound AMBA Memory region1 Limit: Address Offset 0x88 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region1

Table B-27 Outbound AMBA Memory region2 Start Address: Address Offset 0x90 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region2 for the translation of outbound IO transfer

Table B-28 Outbound AMBA Memory region2 Limit: Address Offset 0x98 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region2 for the translation of outbound IO transfer

Table B-29 Outbound AMBA Memory region3 Start Address: Address Offset 0xA0 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region3 for the translation of outbound CFG0 transfer

Table B-30 Outbound AMBA Memory region3 limit: Address Offset 0xA8 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region3 for the translation of outbound CFG0 transfer

Table B-31 Outbound AMBA Memory region4 Start Address: Address Offset 0xB0 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region4 for the translation of outbound CFG1 transfer

Table B-32 Outbound AMBA Memory region4 Limit: Address Offset 0xB8 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region4 for the translation of outbound CFG1 transfer

Table B-33 Outbound AMBA Memory region5 Start Address: Address Offset 0xC0 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region5 for the translation of outbound message transfer

Table B-34 Outbound AMBA Memory region5 Limit: Address Offset 0xC8 of DBI Transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region5 for the translation of outbound message transfer

Table B-35 Master Control Register: Address Offset 0xD0 of DBI Transfer

Bits	Default	Descriptions
31:0	0	<p>The register is designed to control the error mapping sideband signals of the master AHB interface. Only three of these bits are valid. Others are reserved.</p> <ul style="list-style-type: none">• Bit0: 1'b0 indicates that UR maps to response error, 1'b1 set for decode error• Bit1: 1'b0 indicates that CA maps to response error, 1'b1 set for decode error• Bit2: 1'b0 indicates that CRS maps to response error, 1'b1 set for decode error

Table B-36 Slave Control Register: Address Offset 0xE0 of DBI Transfer

Bits	Default	Descriptions
31:0	0	<p>This register is designed to control the error mapping sideband signals of the master AHB interface. Only three of these bits are valid. Others are reserved.</p> <ul style="list-style-type: none">• Bit0: 1'b0 sets for response error maps to UR, 1'b1 set for decode error• Bit1: 1'b0 set for response error maps to CA1'b1 set for decode error• Bit2: 1'b0 set for response error maps to CRS, 1'b1 set for decode error

Table B-37 Global Translation Control Register: Address Offset 0xFC of DBI Transfer

Bits	Default	Descriptions
0	0	This bit enables the translation that is designed for inbound and outbound traffic. 1'b1 indicates that all inbound and outbound memory translations for address and type designed in this reference example are enabled.

B.6.2.10 Legacy Interrupt and MSI Interface Driver Module

The DWC PCIe core supports three interrupt mechanisms (legacy, MSI, and MSIX interrupt). These are mutually exclusive, and only one of these interrupt mechanisms should be enabled per-function at any given time.

This module may be used for legacy interrupt reference design and for MSI interrupt reference design.

There are two sources of interrupt requests supported in this example:

- ❖ From the application logic when a status is done, or other reasons for generation of an interrupt.
- ❖ From an external button being pushed.

You can add or remove the interrupt sources to/from this reference design.

When one of these sources is asserted, the reference design will either generate a legacy interrupt request on sys_int, or the MSI request to the core's MSI interface.

The input interrupt source signals to this modules are:

- ❖ input [NF-1:0] attention_button: A request to PCIe module to generate interrupt based on external button pushed.
- ❖ input [NF-1:0] app_intr_req: A request from application logic on chip to request PCIe module to generated interrupt
- ❖ input [NF-1:0] app_intr_clr: A clear interrupt from application logic on chip to indicate that the interrupt has been served, and application is ready to clear the interrupt. Normally, the application gets the clear status from a driver status register

These signals are wired at the chip-top. The customer is required to review them and provide the required driving source behind them. The application will need to evaluate their interrupt strategy and get the source of an interrupt identify.

Please refer to interrupt_msi_driver.v for more information about driving the MSI interface and legacy interrupt sys_int.

B.6.2.11 Power Management Control Interface Driver Module

The Power Management Control Interface Driver module may be used by an application that has special requirements for the lower power states, beyond the typical power management functionalities designed into the IP core. Most power management controls are handled inside the core. There are some controls provided to the application that can improve or support further power management functionality other than those specified by the PCIe spec. This module contains the descriptions for these special power management functionalities.

There are four outputs generated by this reference module:

- ❖ app_req_entr_l1
- ❖ app_req_exit_l1
- ❖ app_ready_entr_l23
- ❖ apps_pm_xmt_pme

app_req_entr_l1 controls entering the lower power L1 state. This is for an ASPM-enabled device only. The core has a timeout mechanism implemented to automatically enter L1. The PCI Express Base 2.0 Specification allows for optional L1 entry schemes (for instance, a way of detecting the packet finished from all request sources of the application logic), using this signal to accelerate a request to enter L1.

app_req_exit_l1: Exits L1 when a packet is pending for a transmission. Applications are expected to make a decision whether or not to remove the reference clock during L1. This decision should be affected by the clock lock and recovery latency of the PHY. Please refer to the clock and reset example for additional details about reference clock removal. If the clock will be removed during L1, the application is expected to issue an app_clk_req_n wake up signal (note that there is no core clock or application clock before waking up). After the clock is restored, the application may assert app_req_l1_exit or simply present a PM PME event transmit request or any other packet transmit request to transition the core from the L1 to the L0 state.

app_ready_entr_l23: When a device has been programmed into the D3 state and a turnoff handshake has been performed between PCIe devices, the device is expected to enter the L23 state and the main clock and power will go off. If the application has any other reason that requires the presence of the main clock and power, it should deassert this ready signal so that the core will delay entering L23 (the application cannot delay it indefinitely without violating the PM protocol).

apps_pm_xmt_pme: The application is responsible for generating the conditions for a power management event. This is typically tied to a wake-up event. When the PCIe device is in lower power, such as the L1, L2/L3 state, this signal can be used to help wake up the device. An event detected by the application or device addition/removal are the typical sources of the wake up. When a wake-up is decided upon, it is possible that the core is in a no-clock state. In that case, a wake-up sequence must follow after the application asserts the apps_pm_xmt_pme signal.

When the core is in the L1 state, the application should perform two steps:

1. Assert app_clk_req_n.
2. When the clock is up, then assert apps_pm_xmt_pme for one cycle.

When the core is in the L2/L3 state, the application should:

1. Assert apps_pm_xmt_pme. This causes the core to start beacon while aux clock and aux power is supplied.
2. Also assert app_clk_req_n, then wait for the PHY clock to be restored. A PM PME message will be transmitted from the core to notify the PME event.

B.6.3 Considerations During Integration of the DWC PCIe AHB IP

A chip_top.v is present in the *src/customer/generic/* directory. We strongly recommended that you review the top-level RTL code and understand all of the modules provided as a reference design.

Here is a basic summary of the modules to that you should pay special attention to:

- ❖ Clock and Reset module: Most of the reference design modules are synthesizable. The Clock And Reset module is not. It requires additional work on the part of the chip designer to perform the necessary clock generation and to define the clock and reset trees required by the IP and the rest of the chip. The clock and reset module is a place holder for the customer. This module is heavily commented so that customers may find where the clock generation logic should be placed.
- ❖ The AHB master, slave and DBI standard interface drivers also require additional customer attention, as these modules are where the other application AMBA IP blocks or interface logic to the DWC PCIe AHB bridge should be connected to.
- ❖ Interrupt source generation is required from the application logic. Please refer to the above interrupt section for details on the interrupt source signals.
- ❖ PCIe Power management is handled within the PCIe core unless there is a special requirement from the application. Please refer to the Power Management module ([Section B.6.2.11](#)) for information to control the power states with respect to special requirements.

Most of the other modules should have connections made correctly to the interfaces that they are intended to service, and only the internal logic should require modification. These modifications may also require additional inputs and outputs to be added to the reference designs, but the designs are intended to represent a "complete" interconnection.

B.6.4 Configuration Parameters

Consider the following guidelines when selecting the AHB bridge configuration parameters.

B.6.4.1 Queue Depths

In general, when selecting the queue depths, the major trade-off to be made is between area and performance. Larger queue sizes will have a larger area impact but may also reduce the occurrence of back pressure, depending on system settings. In addition, synthesis QOR may also be negatively impacted by selecting larger queue sizes.

B.6.4.1.1 Master Request Header Queue Depth

This queue depth, set via `coreConsultant`, specifies the number of master requests that the application requires the AHB bridge to enqueue in the inbound direction. There is a receive queue in the native PCIe core for all inbound requests therefore it is not necessary to have a large queue depth for this master request header queue. This queue depth is set to a default of 4 (recommended value) which means that there can be a total of 4 outstanding inbound read or write requests enqueued in the AHB bridge.

There are pipe stages within the AHB adapter and internal generic interface adapter modules. This queue depth is used to minimize the start/stop pipe delay within the AHB bridge. Increasing this queue depth (during configuration) can improve the overall performance of the AHB interconnect based on the pipe stages within the AHB interconnect. The disadvantage of increasing this depth is an increase in gate count and RAM size. Under normal operation, the default value should be adequate.

B.6.4.1.2 Master Request Data Queue Depth

This queue depth is sized based upon the header queue depth discussed in the previous section. This queue depth is also dependent upon the native PCIe core's data bus width. For example, when the data bus is set to 32 bits, and assuming 1 inbound write for every 4 inbound requests, and the MTU (max transfer size) is 128 bytes, the calculated default depth is $MTU/4 + 2 = 34$.

Increasing this queue buffer depth parameter can improve the overall performance of the AHB interface based on the pipe stages within the AHB interconnect. The disadvantage of increasing this depth is larger gate counts and RAM sizes. It is important for the application to have some understanding of the patterns of inbound requests, such as the average number of reads and writes, how they are interleaved, and typical write request sizes.

The suggested default value should be adequate under normal operation. The default value is set to store one `CC_MAX_MSTR_MTU` worth of data +2.

B.6.4.1.3 Slave Request Header Queue Depth

This queue depth specifies the number of outbound requests that can be enqueued in the AHB bridge. Since there is no transmit queue in native PCIe core, this queue depth can impact the performance of the overall outbound transitions, depending on the Posted and Non-Posted credit availability of the remote PCIe device. If the device is balanced with the AHB interface bandwidth and native PCIe core's bandwidth, and the credits are available from the PCIe link, then this queue is not required to be very large. Otherwise, increasing this queue buffer depth parameter can improve the overall performance. In some applications however, it can cause more blocking issues in a credit-limited system since both Posted and Non-Posted requests are enqueued into this same buffer.

The suggested default value should be adequate under normal operation. The default value is set to `CC_MAX_SLV_TAG`.

B.6.4.1.4 Slave Request Data Queue Depth

This data queue depth is related to the header queue depth discussed in the previous section. It is the data queue depth based on headers in the header queue.

This depth is dependent on the number of read or write requests that are enqueued and the maximum transfer size of write requests.

The suggested default value should be adequate under normal operation. The default value is set to store one CC_MAX_SLV_MTU worth of data +2.

B.6.4.2 Remote Device Maximum Read Request Size

This configuration parameter sets the size of the buffer inside the AHB bridge composer module. It should be set to a size large enough to support the maximum read transfer size expected by the PCIe remote device for outbound read transfers. If this configuration parameter is not set large enough, it will get a partial completion returned to the AHB read channel and in general will not function properly. If it is set larger than necessary, gate area will be wasted.

B.6.4.3 Master Page Boundary Size

This parameter specifies an address page boundary of either 128, 256, 512, 1K, 2K or 4K. The default value is 4K. No inbound or outbound transactions can have an address that crosses this boundary. Typically, 4K would be the recommended setting for outbound transfer. 1K would be the recommended setting for inbound transfer. The AHB bridge will ensure that the inbound request will not cross the selected page boundary when driven onto AHB master interface.

B.6.5 RAMs

This section describes the RAMs that may be used in the AHB Bridge, dependent upon configuration settings in coreConsultant. Some of RAMs may be required for one application, but not others. All RAM options are described in the following sections.

There are two sets of RAMs. One is for the PCIe native core. The other for the AHB bridge. The descriptions for the AHB bridge RAMs are as follows.

RAM size can be printed from coreConsultant after an AHB bridge is configured. Most RAMs are sized based on application's implied requirements. Few RAMs are sized directly by the application via coreConsultant.

B.6.5.1 XADMX0 Decomposer Header and Data RAMs

XADMX0 decomposer header and data RAMs are used to implement FIFOs to buffer AHB bridge inbound responses when the maximum PCIe inbound transfer and AHB transfer sizes may be different. Since the header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AHB master interface clock is running at the same frequency as the base core clock. The hclk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.2 XADMX1 Decomposer Header and DATA RAMs

XADMX1 decomposer header and data RAMs are used to implement FIFOs to buffer AHB bridge outbound responses when the maximum PCIe outbound transfer and AHB bus burst sizes may be different. Since the header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AHB master interface clock is running at the same frequency as the base core clock. The hclk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.3 Slave TAG Asynchronous FIFO RAM

Slave TAG asynchronous FIFO RAM is used to cross a TAG management interface clock domain between the outbound decomposer and outbound composer. It has two clock inputs that can be tied together if the AHB slave interface clock is running at the same frequency as the core clock. The hclk is the slave interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.4 Slave TAG Manager RAM

Slave TAG manager RAM is designed as an asynchronous FIFO for outbound transfer TAG control/recycle. It has two clock inputs that can be tied together if the AHB slave interface clock is running at the same frequency as the core clock. The hclk is the slave interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.5 RADMX Asynchronous RAM

RADMX asynchronous RAM is used to cross a clock domain between the outbound composer and the AHB response channel. It has two clock inputs which can be tied together if the AHB slave interface clock running at the same frequency as core clock. The hclk is the slave interface clock. The parameters used in the

instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.6 RADMX Decomposer Header and Data RAMs

RADMX decomposer header and data RAMs are used to implement FIFOs to buffer AHB bridge inbound requests when the PCIe transfer size may be different from the AHB bus burst size. Since header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AHB master interface clock is running at the same frequency as the base core clock. The hclk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.7 Master TAG Manager RAM

Master TAG Manager RAM is designed as an asynchronous FIFO for inbound transaction TAG control/recycle. It has two clock inputs that can be tied together if the AHB slave interface clock is running at the same frequency as core clock. The hclk is the master interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.8 GM Composition RAM

GM Composition RAM is designed as a segmented-buffer queue for inbound response composition. It runs under the master AHB clock domain. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.9 RADMX Composition RAM

RADMX composition RAM is designed as a segmented-buffer queue for outbound response composition. It is running under the PCIe base core clock domain.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AHB bridge's CC constant file. Please refer to this file for detailed sizing information.

B.6.5.10 RADMX Asynchronous TAG FIFO RAM

This RAM is designed for an AHB configuration where the AHB clock is different than the core clock. TAG Integration

B.6.6 The Suggested Integration Method for Application Logic

Refer to the “Building and Verifying Your Core” chapter in the applicable DesignWare core’s user manual for instructions to configure, synthesize, and simulate the core. This section gives you an idea of how to build a chip with the IP core instantiated.

The recommended application integration flow is as follows:

Step 1. Read the implementation guidelines presented in this addendum very carefully.

Step 2. Build a core with your desired features.

Step 3. Review the structured chip level reference design (i.e., chip_top.v, clk_rst.v, etc.)

Step 4. Integrate the application logic into the chip_top.v.

The main considerations for Steps 1 through 3 are:

1. Decide on the number of client interfaces and the type of traffic going through them. There are template (.v) files to help you to integrate the application logic required to drive the client interfaces. The client interfaces are where the application logic can send traffic to the PCIe link.
2. Decide on how you want received inbound traffic handled. Two interfaces (ELBI and Target1) are available within the IP core. Two template (.v) files are available to integrate the required application logic.
3. You should review the clk_rst.v RTL code carefully and read all comments in the RTL file. There is logic (such as clock switching circuitry, etc.) in this file that you will need to replace.
4. You should review the strategy for interrupt implementation in the application and implement the respective interrupt source logic. Signals app_intr_req and app_intr_clr (at chip_top.v) should be driven by customer logic. You should also review the interrupt reference design module.
5. There are a few signals that you will use to control the IP core’s power state transitions. You should provide the driving source logic or tie these signals. There is a power management template (.v) in the reference design that you should review as well.
6. Complete Step 3 for other miscellaneous functionalities.

B.6.7 VTB Test Bench Integration Method (for EP Core Only)

The example Verilog testbench (VTB-AHB) for the Synopsys PCI Express Cores, shown in [Figure B-21](#), is a demonstration testbench that illustrates:

- ❖ How to instantiate and connect the Synopsys PCI Express Verification IP (VIP) components, Synopsys PCI Express Core, PCI Express PHY, and the AHB VIP models (master, slave, and Bus VIP models) in a Verilog testbench. The PCI Express Core is instantiated as the device under test (DUT).
- ❖ How to use Verilog tasks to drive both the VIP and DUT. The sequences demonstrated include initialization, configuration, PCI Express traffic generation, and data send/receive (upstream and downstream traffic) value checking.



The Synopsys PCI Express Verification IP (VIP) and AMBA AHB verification IP are separate products packaged with their own set of documentation. Please refer to the *DesignWare PCI Express VIP User Manual* and the *DesignWare AHB Verification IP Databook* for more information.

Figure B-21 VTB-AHB Block Diagram

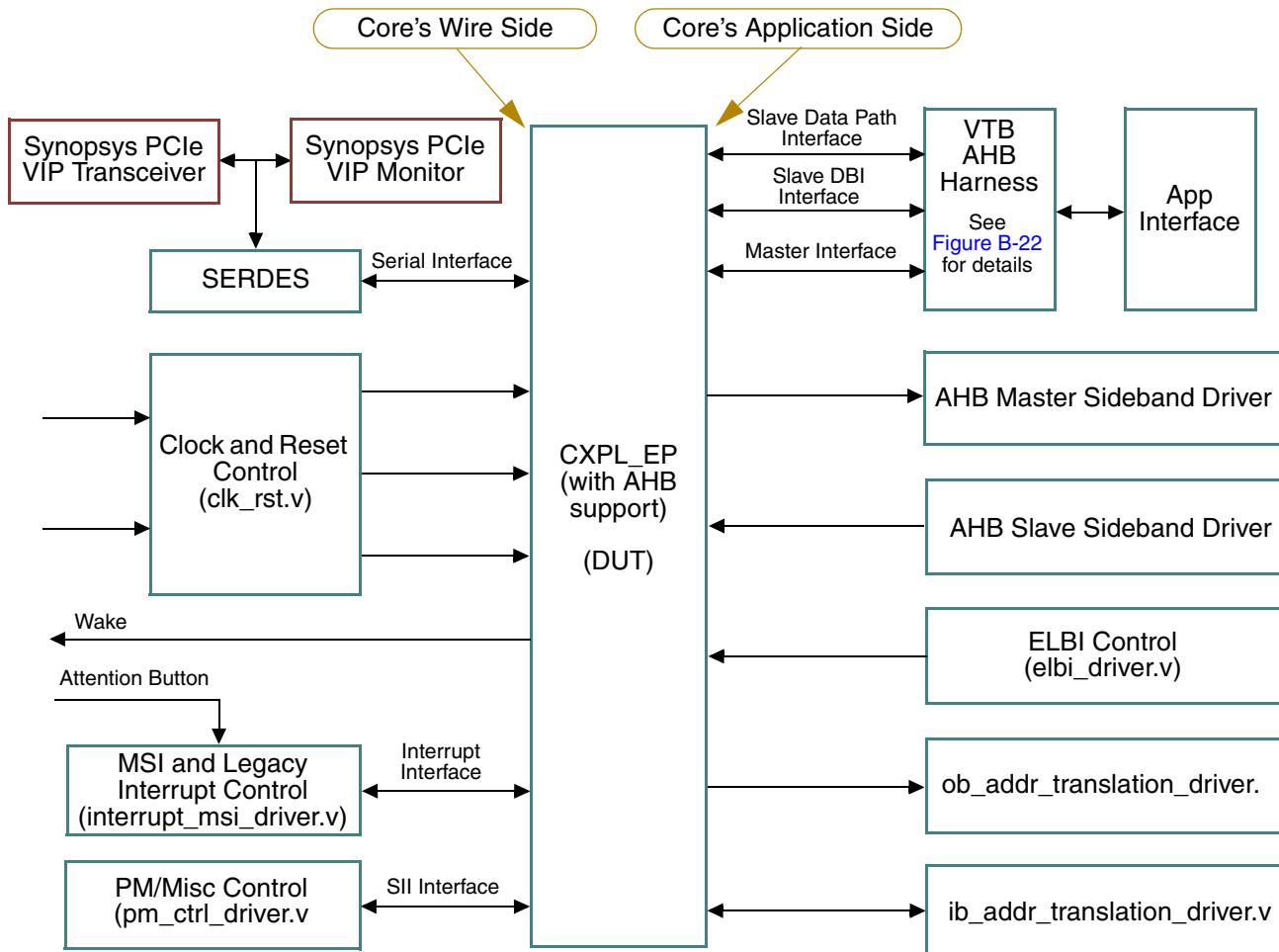
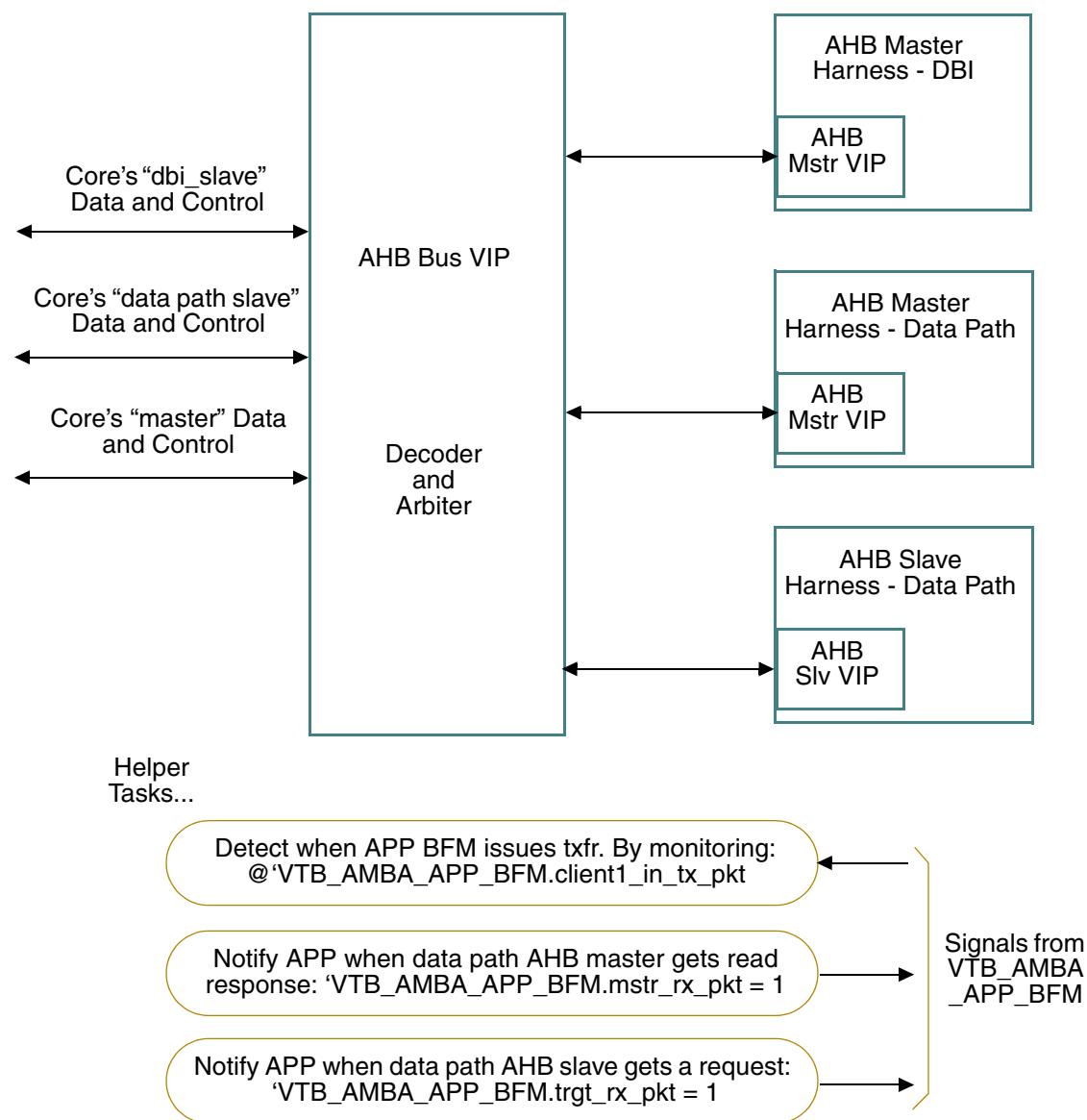


Figure B-22 VTB AHB Harness, Block Diagram

B.6.7.1 VTB Features

To help you get started with your system-level verification, the VTB demonstrates several important elements of working with the Synopsys PCI Express Cores and VIP products, including:

- ❖ How to connect the core to the PCIe VIP transceiver (`pcie_txrx_vmt`) and monitor (`pcie_monitor_vmt`) in a Verilog testbench with the PIPE-compliant PHY
- ❖ How to connect the core to the AHB master (`ahb_master_vmt`), AHB slave (`ahb_slave_vmt`), and AHB bus (`ahb_bus_vmt`) VIP models in a Verilog testbench.
- ❖ How to initialize the VIP's and core
- ❖ How to initiate the LTSSM

- ❖ How to configure the PCIe VIP transceiver and monitor
- ❖ How to configure and initialize the AHB harness (using various AHB VIP models)
- ❖ How to send PCI Express Configuration transactions from the PCIe VIP to the core
- ❖ How to send “Upstream” Memory Read and Write transactions. Upstream traffic, for EP core, is defined to be as requests issued from the application side of the core (AHB master VIP model issues the request).
- ❖ How to send “Downstream” Memory Read and Write transactions. Downstream traffic, for EP core, is defined to be as requests issued from the wire side of the core (PCIe VIP model issues the request).

B.6.7.2 Setting Up and Running the VTB

The VTB package includes a script to compile and simulate the VTB-AHB using the Synopsys VCS tool (NC and MTI are also supported). Before you can run the simulation, you need to install the core.



For step 1, refer to the associated installation instructions to install the core and the VIP and set their required environment variables. For your convenience, an example setup file is included in [Section B.6.7.4, “Example Setup Commands ” on page 750](#).

The steps to build the working directory and run the simulation are:

1. If you have not already done so, install the PCI Express Core into your \$DESIGNWARE_HOME directory. Follow the procedure in the Installation Guide for the version of the core you are working with, including setting up your environment for the required tool versions.
2. Invoke coreConsultant and create a workspace for one of the PCI Express Cores:
 - a. Go to the your working directory and invoke coreConsultant:


```
% cd <my_work_path>
% coreConsultant
```
 - b. Use File>New Workspace to a new workspace for the EP Core.
 - c. Use the AHB configuration provided at:


```
<workspace_dir>/example/VTB/vtb_default_ahb_1sx4.config
```

 Type the selected configuration in the command box at the bottom of the coreConsultant window as follows:


```
source <workspace_dir>/example/VTB/<selected vtb config>
```

 Then click the Apply button below the command box.
 - d. Click on Simulate, then Apply. This will run the VTB demo. It will take about 10 minutes before the window will update with first test messages. This is due to the VIP files being compiled.



If you want to run the VTB demo from the command line there are two steps to perform:

1. cd <workspace>/sim/vtb_demo_test_s1 or s2, s3, or s4
2. test.startsim

When the simulation completes, the following log files are available in the test directory:

- ❖ logfile: Simulation log file, includes compile messages and simulation execution messages.

- ❖ pcie_monitor_sym.log: Generated by the PCIe Monitor VIP; contains a log of all symbols that are transferred over the PIPE.
- ❖ pcie_monitor_trans.log: Generated by the PCIe Monitor VIP; contains a log of transactions that are transferred over the PIPE.

B.6.7.3 Examining the Example Code

The VTB is intended only as an example testbench and not as a full functional verification environment. Therefore, the most effective use of the VTB is to examine the source code to understand how the connections are made and how stimulus is applied to the core and to the VIP.

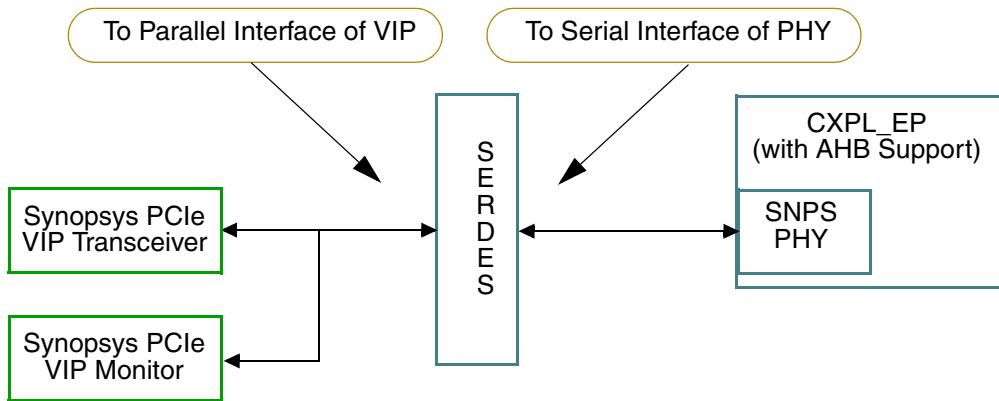
B.6.7.3.1 Top-Level Connections

The top-level connections are in the file sim.v, which instantiates the following components as shown in [Figure B-21](#):

- ❖ dut under chip_top(): The PCI Express Core that you are working with is the device under test. The dut instance of the PCI Express Core is linked directly to the PCI Express Core configuration (workspace) that you created with coreConsultant in “[Setting Up and Running the VTB](#)” on page [746](#).
- ❖ VIP_Transceiver: The PCIe VIP Transceiver (pcie_txrx_vmt) component, which functions as an upstream device. The VIP Transceiver is configured to use the 8b/10b interface. However, a behavioral “serdes” model is inserted between it and the Synopsys PHY in order to use the serial interface of the PHY (see file ‘vip_instance.vh’ which is included in the ‘sim.v’ file).
- ❖ Monitor: The VIP Monitor (pcie_monitor_vmt) component. Note that the Verilog code which instantiates the VIP is contained in the file named vip_instance.vh. (file ‘vip_instance.vh’ is included in the ‘sim.v’ file).
- ❖ u_ahb_harness: This is the AHB harness (vtb_ahb_harness) used for driving and interacting with the application side of the core (file ‘amba_bfm_instance.vh’ is included in the ‘chip_top.v’ file). Please refer to [Figure B-22](#) for a block diagram of this AHB harness.
- ❖ u_endpoint_bfm: This is a simple application model (vtb_app_ifc) trying to show a signal based interface that communicates with the AHB harness. This model is used to trigger the harness to schedule MEM_RD/MEM_WR txfrs. Also, this model is always waiting for input from harness in the case where the application is targeted for memory request. (see file ‘vtb_app_bfm_instance.vh’ which is included in the ‘chip_top.v’ file).

B.6.7.3.2 Serial Interface Connections

The main connections of interest in sim.v are the serial bit connections between the Synopsys PHY and the VIP Transceiver. For obvious performance reasons, the VIP Transceiver is designed to work with parallel interface (byte level or symbol level interface), however, in order for the PCIe VIP to connect to the PHY, we need to add a separate “serdes” model (see file ‘vip_instance.vh’ which is included in the ‘sim.v’ file). Refer to [Figure B-23](#) to see where the “serdes” model is inserted in relation to the PHY. (Note: Typically, for verification purposes of a core, the VIP transceiver will be connected directly to the PIPE interface of the core and bypassing the PHY’s functionality. However, in some cases where the PHY model is being tested or demonstrated, the use of an additional ‘serdes’ model is warranted as is the case with this VTB).

Figure B-23 Connecting the PCIe VIP to the PHY Serial Interface via SerDes

B.6.7.3.3 PCIe PIPE Compliant PHY Connectivity

The PHY is instantiated in the chip_top() module. The hierachal path to the PHY instance is: sim.u_chip_top.u_phy. By default, the VTB instantiates the Synopsys UP3 (version 2.5a) PCIe PIPE Compliant PHY. The top-level of this PHY simulation model is contained in the file, phy_pcs_up3.v under the ./src/Phy/pcs_up3 path. Refer to [Figure B-12](#) for a diagram of the chip_top architecture.

B.6.7.3.4 VIP Control

The following sections will provide basic information on how to control Synopsys verification IP models (VIP).

PCIe VIP Control

The Verilog tasks to control the VIP components using the VIP commands are defined in the *DesignWare PCI Express VIP User Manual*. For example, the VIP command 'set_config_param' is used in the file 'configuring_vip.v' to configure the VIP Transceiver and Monitor. See the table below for various tasks used to control the PCIe VIP in this VTB.

Task Examples Used in VTB to Control the VIP Models

For more comprehensive examples of how to control and monitor the DUT and the VIP in a Verilog testbench, it is best to examine the Verilog files listed in [Table B-38](#). You can use the files listed in [Table B-38](#) as examples of how to build your own testbench using the Synopsys PCI Express Cores and PCI Express VIP.

Table B-38 VTB Files

File	Description
sim.v	Main top-level of the simulation environment which instantiates the PCIE Core as well as the PCIE VIP.
tb.v	Testbench file for Verilog testbench base tasks, such as reset control (global_init), waiting for link establishment (wait_linkup), as well as basic traffic generation tasks.

Table B-38 VTB Files (Continued)

File	Description
config_regs	Used to initialize the PCIE Core. Has tasks for Inband configurations as well as sideband (CPU_BFM). This is also where the inband BAR discovery is performed for Endpoint products.
configuring_vip.v	Demonstrates how to configure the PCIe VIP Transceiver and Monitor.
mem_block_read.v	Demonstrates how to create a Memory Read packet for the PCIe VIP Transceiver to send to the core.
mem_block_write.v	Demonstrates how to create a Memory Write packet for the PCIe VIP Transceiver to send to the core.
vip_cfg_tasks.v	Demonstrates how to create a Config Write and Read packet for the PCIe VIP Transceiver to send to the core.
vip_instance.vh	Demonstrates how to instantiate the PCIe VIP Transceiver and monitor.
vtb_ahb_slave_harness.v	Demonstrates how to instantiate the AHB slave VIP model ('ahb_slave_vmt', with instance name 'slave_vip'). This harness also contains few supporting tasks to configure the VIP (task 'configure') and to intercept control for handling memory read and write (task 'task_watch_for_end_of_xfer');
vtb_ahb_master_harness.v	Demonstrates how to instantiate the AHB master VIP model ('ahb_master_vmt', with instance name 'master_vip'). This harness also contains few supporting tasks to configure the VIP (task 'configure') and to the testbench can issue transfers (task 'schedule_ahb_transaction_blocking').
vtb_ahb_harness.v	Demonstrate how to instantiate and connect the AHB VIP Bus, master harness, and slave harness (these instantiations exist in an included file called 'ahb_harness_wires.vh'). Furthermore, this file shows examples of a basic detector to determine when the 'vtb_app_ifc' bfm needs to be serviced (task 'detect_app_to_core_tx_packet_c1'). The detector in turn will service the request by scheduling a txfr by calling 'tb_ahb_master.schedule_ahb_transaction'.
ahb_defines.vh	Contains various AHB VIP related constants needed/used in the AHB harness files.
vtb_check.v	Contains data checkers routines for txfrs in both directions. Tasks names would indicate the direction of the txfr flow (task with string 'a2w' in them, specify txfrs from "application side" of core to the "wire side" of the core).
vtb_app_bfm_instance.vh	Contains instantiation of the simple application BFM ("vtb_app_ifc" with instance name 'u_endpoint_bfm') which is used to interact with the programmed interface of the AHB VIP's (see description of 'vtb_app_ifc.v' below).

B.6.7.4 Example Setup Commands

The following example shows a command sequence to install and set up the PCI Express Core and VIP, and run the example VTB.

Before executing the `compile_script_vcs` command below, be sure to edit the `compile_script_vcs` file to specify the name of your coreConsultant workspace for the PCI Express Core, as described in [Section B.6.7.2, "Setting Up and Running the VTB."](#)

```
# Set DESIGNWARE_HOME to your DesignWare IP installation directory setenv  
DESIGNWARE_HOME <my_dw_home>  
  
# Set up licenses setenv LM_LICENSE_FILE ${LM_LICENSE_FILE}:<my_license_file> setenv  
SPNSLMD_LICENSE_FILE ${LM_LICENSE_FILE}  
  
# Set up the required tools  
# coreConsultant  
set path = (<cC_install_dir>/bin $path)  
  
#VCS  
setenv VCS_HOME <vcs_install_dir>  
set path = ($VCS_HOME/bin $path)  
  
#Vera  
setenv VERA_HOME <vera_install_dir>  
set path = ($VERA_HOME/bin $path) setenv LD_LIBRARY_PATH  
$VERA_HOME/lib:$LD_LIBRARY_PATH  
  
# Install the PCI Express Core that you downloaded from Synopsys  
# (if not already installed in $DESIGNWARE_HOME).  
dw_iip_DWC_pcie_<core>_<release>.run  
  
# This step is only needed if you know that the model supplied with the core is too old.  
In general you can skip this step. Install the PCI Express VIP that you downloaded from  
Synopsys  
# (if not already installed in $DESIGNWARE_HOME).  
dw_vip_pcie_<release>.run  
  
# This step is only needed if you know that the AMBA model that you have is too old. In  
general you can skip this step. Install the AMBA VIP that you downloaded from Synopsys #  
(if not already installed in $DESIGNWARE_HOME). dw_vip_amba_<release>.run  
  
# Create a user configuration (workspace) for the PCI Express Core  
coreConsultant  
  
# Note: Use the coreConsultant GUI to create a workspace in  
# the current directory. Follow source configuration directions from Section B.6.7.2.  
Then, quit coreConsultant.  
  
# If you choose to run the simulation without the use of coreConsultant tool, then you  
can Go to the VTB working directory.  
cd <workspace>/sim/vtb_vtb_test_s1  
  
Run the simulation using test.startsim.
```

B.6.7.5 Test Descriptions

Test 'vtb_ahb_test.v' has subtests 1 through 4. Here is a description of each of the subtests.

1. subtest 1: (a2w) generates Memory Write requests from AHB application BFM targeting the WIRE side BFM (i.e. targeting the PCIe VIP). To generate these transfer, the test uses task "create_transaction()" to create the txfr, and then uses task 'load_transaction()' to load it into the VIP for execution. The test will check the data as it arrives at the PCIe VIP model. The test will loop to generate 5 write transfers starting with:

```
adrs = 0x2000_0500, payload = 8 bytes
```

With each iteration, the address will be incremented by 0x80, and payload incremented by 0x4 bytes.

2. subtest 2: (a2w) generates Memory Read requests from AHB application BFM targeting the WIRE side BFM (i.e. targeting the PCIe VIP). This test is similar to subtest 1, except that it generates a read txfr. The test will check the response data as it arrives at the AHB master VIP model. The test will loop to generate 9 read transfers starting with:

```
adrs = 0x2000_0100, payload = 8 bytes
```

With each iteration, the address will be incremented by 0x80, and payload incremented by 0x4 bytes.

3. subtest 3: (w2a) generates Memory Write request from Wire-BFM (i.e. PCIe VIP) targeting the AHB Application BFM. To generate these transfers, the test will call task 'core_rx_write()' which in turn will call these tasks to setup the buffer and issue the txfr via the PCIe vip:

```
tb.mem_wr_buffer(pkt_addr, num_pyld_bytes,data_in);  
tb.mem_block_write(pkt_addr, num_pyld_bytes);
```

The test will check the data as it arrives to the APP BFM (by calling task 'w2a_check_wr()'). The test will loop to generate 10 write transfers starting with:

```
adrs = 0xd000_0000, payload = 12 bytes
```

With each iteration, the address will be incremented by 0x100, and payload incremented by 0x4 bytes.

4. subtest 4: (w2a) generates Memory Read request from Wire-BFM (i.e. PCIe VIP) targeting the AHB Application BFM. To generate these transfers, the test will call task 'core_rx_read()' which in turn will call these tasks to setup the buffer and issue the txfr via the PCIe vip:

```
tb.mem_rd_buffer(pkt_addr, num_pyld_bytes,data_in);  
tb.mem_block_read(pkt_addr, num_pyld_bytes);
```

The test will check the data as it arrives to the APP BFM (by calling task 'w2a_check_rd()'). The test will loop to generate 10 read transfers starting with:

```
adrs = 0xd000_0000, payload = 12 bytes
```

With each iteration, the address will be incremented by 0x100, and payload incremented by 0x4 bytes.

B.6.8 How to Proceed

Once you have created and configured the RTL core, you can start adding your application interface code. Refer to [Section B.6, “Implementation Guidelines.”](#) This chapter describes the different application interface blocks and what you need to do to start adding your own application code. You can also look at the VIP code from the VTB to understand how to create traffic for the VIP. You need to start adding testbench code to drive both the application side of the core, and to generate traffic on the VIP side.

C

AXI Bridge Module

C.1 Product Overview

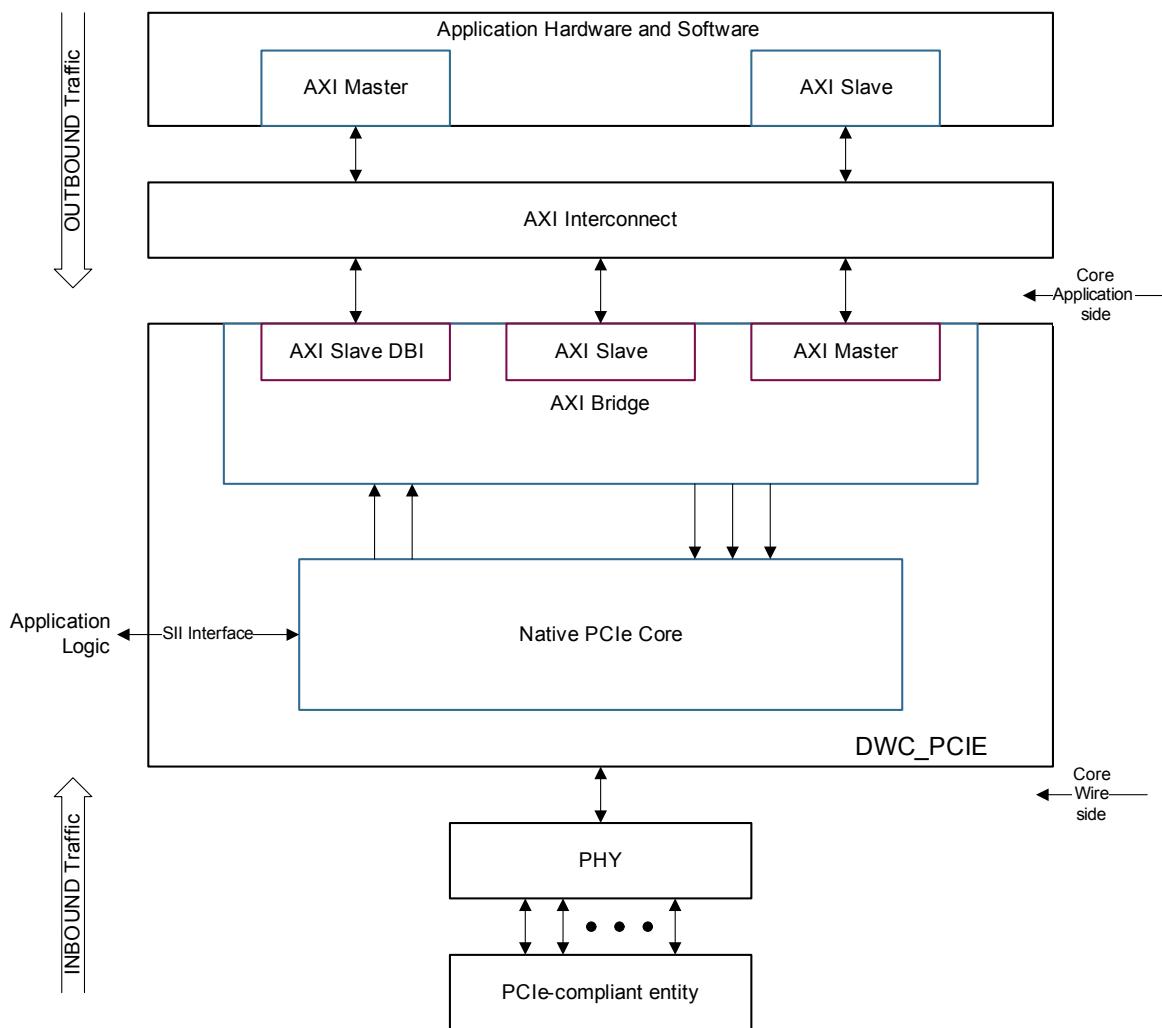
C.1.1 General Product Description

The Synopsys DWC PCI Express solution provides an optional AXI Bridging capability for directly adding a PCI Express link to an AXI system fabric. This can significantly reduce the time to design PCI Express into an AXI-based SOC. Refer to [Figure C-1](#) for a system-level view of the DWC PCIe AXI core and [Table C-1](#) for a definition of terms and acronyms.

The AXI Bridge Module acts as a bridge between the standard AXI interfaces and the Synopsys DesignWare PCIe core native interfaces. The bridge interconnects the AXI interfaces within an AMBA-embedded system with a remote PCIe link, as either a root complex port, or as an endpoint port. The bridge supports up to three AXI interfaces, one for an AXI master, one for an AXI slave, and one for DBI access to the native PCIe core.

The AXI master interface enables a remote PCIe device to read and write to an AXI slave connected to the AXI bridge. The AXI slave interface enables an AXI master to read and write through the AXI bridge to a remote PCIe device. The slave DBI interface enables an AXI master to read and write to registers inside the native PCIe core, or the device-specific registers attached to the PCIe native core's ELBI interface.

Throughout this document, the terms inbound and outbound are defined with respect to the AXI fabric. That is, inbound transactions are defined as the transactions presented by the native PCIe core's AXI master interface. Outbound transactions are defined as the transactions generated by an AXI master that target a remote PCIe device.

Figure C-1 System-Level View of the DWC PCIe AXI Core**Table C-1 Definitions of Terms And Acronyms**

Term or Acronym	Definition
Core	Identifies the entire core. The core includes the native PCIe-core and AXI bridge.
Native PCIe core	Identifies the basic DesignWare library PCIe core which has its own non-standard, proprietary dedicated bus interface to the application.
Bridge AXI slave	The AXI slave interface on the core.
Bridge AXI master	The AXI master interface on the core.
Bridge DBI AXI slave	The AXI slave interface dedicated to the internal DBI (Data Bus Interface). This interface is used to access the core's internal registers (CDM).

Table C-1 Definitions of Terms And Acronyms (Continued)

Term or Acronym	Definition
DBI	Data Bus Interface
	This bus is internal to the native core. It is used to access the core's registers and external application's registers. The DBI-AXI slave interfaces provides the direct access to this internal bus.
Inbound traffic	PCIe transactions that enter the native core from the wire side of the core (PCIe wire). These transactions are passed on to the AXI bridge where they will be delivered to the application side.
Outbound traffic	AXI transactions that enter the native core from the application side of the core. These AXI transactions are passed to the native core, where they will be sent out onto the PCIe wire.
MTU	Maximum Transfer Unit
	Specifies the maximum packet payload size supported. This indicates the maximum allowed transfer size for a write or completion.
Page boundary	Specifies the address page boundary size supported by the bridge. No packet can have an address that crosses the specified address boundary.
CDM	Configuration Dependent Module
	This is an internal block in the native core that houses the PCI configuration registers and some user-accessible registers that reside in the core. In general, an application may use our core, but will add other registers that are unique to their applications. Those new application registers will be referred to as External Application Registers (EAR).
LBC	Local Bus Controller
	This is an internal block that resides in the native core. It allows the DBI interface (from the application side) or the wire side interface (via the radm_TRGT0 interface) to access the core's Configuration Dependent Module (CDM) registers and/or the External Application Registers (EAR). (For additional details on this module, please refer to the native core documentation.)
ELBI	External Local Bus Interface
	This is an interface on the native core that processes read/write access to the external application registers (EAR). For applications that require external registers, the application can access the EAR through the bridge or an entirely different interface (outside the scope of this core). (For additional details on this interface, please refer to the native core's documentation.)

C.1.2 System Overview

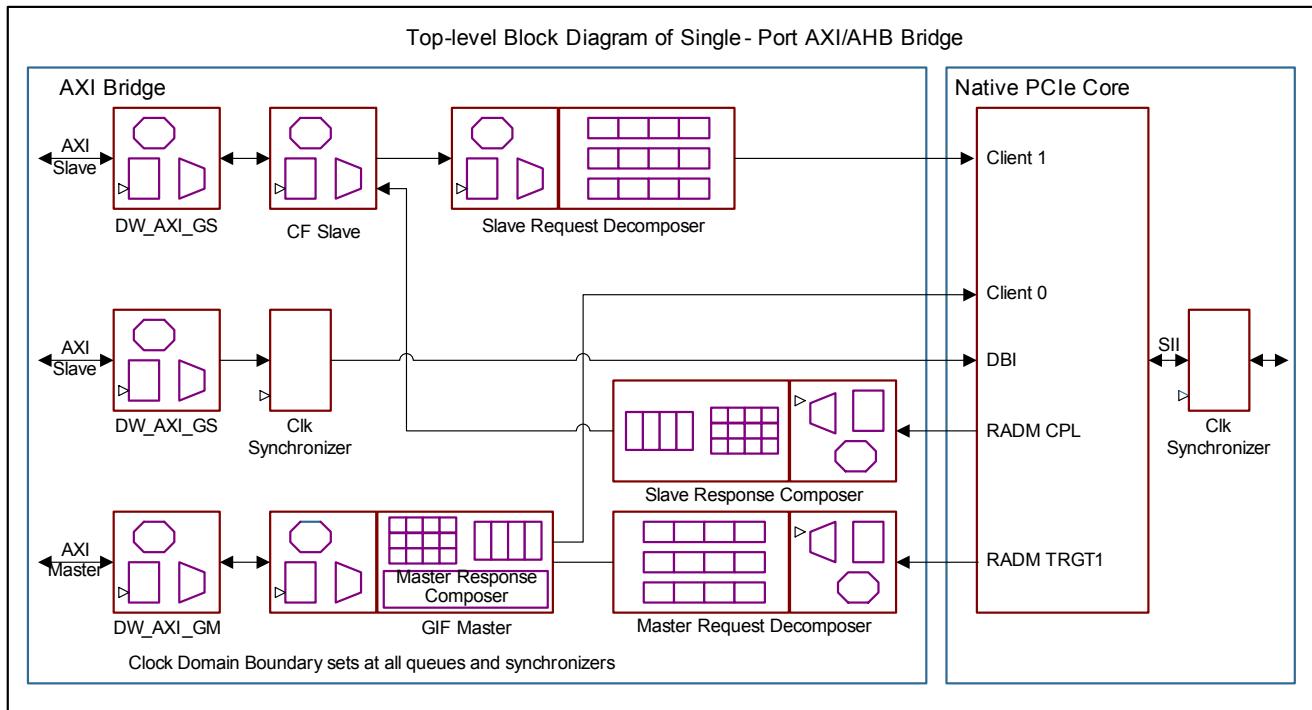
C.1.2.1 Block Diagram

The AXI Bridge Module (see [Figure C-2](#)) provides an interface between the DesignWare PCI Express IP's native application interface and the AXI interconnect. It enables a remote PCIe device to be either an AXI slave or an AXI master.

This module contains AXI master and slave protocol handlers, internal slave and master control for generic request and response interfaces, a packet composer, and a packet decomposer for response formation.

The slave and master protocol handlers support the AXI protocol conversion between an AXI transfer and a generic transfer within the bridge. The slave and master generic interface supports the conversion of an AXI transfer to a PCIe transaction. The packet composer and decomposer support the segmentation and reassembly of a PCIe transaction.

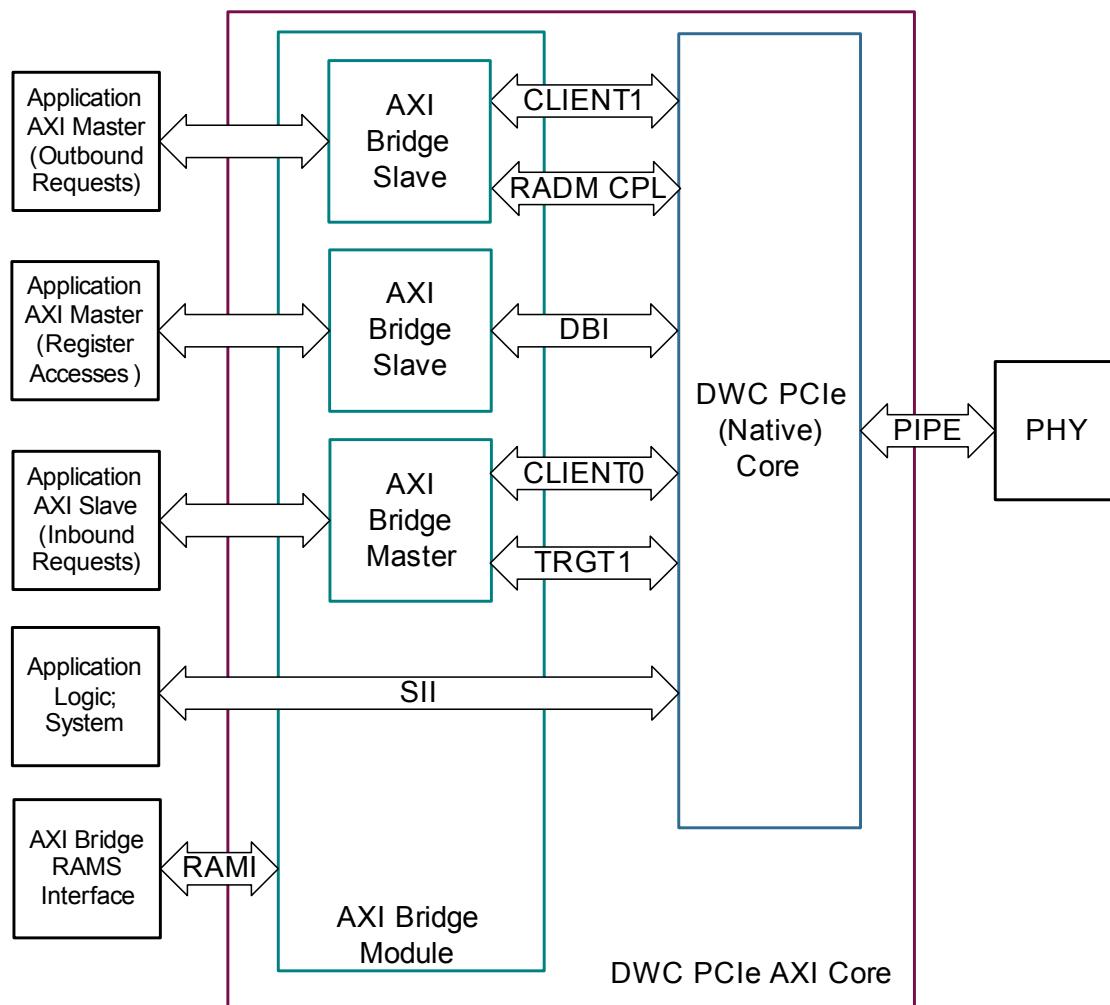
Figure C-2 AXI Bridge Module, Block Diagram



C.1.2.2 Interfaces

Figure C-3 shows the DWC PCIe AXI core top-level interfaces.

Figure C-3 DWC PCIe AXI Core Top-Level Interfaces



C.1.3 Features List

The AXI Bridge Module supports the following features. Please refer to the respective native PCIe core's user manual for a list of all PCIe core features.

- ❖ AXI Master and Slave interfaces for inbound and outbound PCI Express requests
- ❖ All types of PCI Express transactions supported through the AXI Bridge
- ❖ An optional independent AXI Slave interface (or a single, shared AXI Slave interface) to access native core's CDM registers and application specific registers via ELBI
- ❖ Programmable AXI master and slave address widths, data bus widths, and ID bus widths
- ❖ Independent configuration of bus width for PCIe core data bus, AXI master bus and AXI slave bus
- ❖ Programmable buffer sizes for AXI master and slave requests
- ❖ Independent programmable user-defined clock rates for the PCIe core, AXI master bus, AXI slave bus, and AXI slave DBI bus
- ❖ Programmable maximum number of inbound and outbound read requests for AXI
- ❖ All burst-sizes supported for both AXI master and slave interfaces
- ❖ Programmable/extended burst lengths to support up to 4K read/write burst lengths over AXI master and slave interfaces
- ❖ Unaligned AXI transfers using WSTRB for both AXI master and slave interfaces
- ❖ Independent maximum read request and transfer sizes between AXI and PCIe (transfers can be split into multiple transfers)
- ❖ Response to AXI slave request combined gathering from split PCIe completions that are received in-order
- ❖ Response to AXI master request combined gathering from multiple AXI responses
- ❖ Out-of-order response support for transactions with unique IDs
- ❖ PCIe legacy interrupt or MSI and MSIX support
- ❖ User-defined error mapping between PCIe errors (UR, CA, CRS, poisoned, and ECRC error) and AXI slave response errors (SLVERR and DECERR)
- ❖ User-defined error mapping between PCIe errors (UR, CA, CRS, poisoned, and ECRC error) and AXI master response error (DECERR_W and DECERR_R)
- ❖ Byte parity check for the address and data buses throughout the bridge
- ❖ PCIe completion timeout handling



Note SR-IOV support over the AXI bridge is not yet provided.
Multi-function support over the AXI bridge is not yet provided.

C.1.4 Feature Limitations

There are many configurable parameters that the DWC PCIe native core and the AXI bridge support. [Table C-2](#) identifies the limitations encountered when using the AXI bridge with the DWC PCIe native core.

Table C-2 Limitations

AXI Bridge Limitation	Module	Requirement
For a burst transaction, if an external AXI slave is going to respond with DECERR or SLVERR for any data beats, it must respond with DECERR or SLVERR for all data beats in the transfer.	NA	NA
Page boundary has to be greater than or equal to the maximum payload size	The Decomposer block performs page boundary splitting and requires page boundary to be larger than or equal to the MTU.	For AXI slave, we only support a 4K page boundary for outbound transfers. For the AXI master, the page boundary is programmable with one limitation; the page boundary must be larger than the master MTU size
The AXI slave interface has only static selection of whether or not to support out-of-order service.	ts_tag_gen.v is designed to have static configuration for AXI slave out or in-order service.	Dynamically changing the transfer ID from unique mode to non-unique mode is not supported. The slave interface of AXI bridge expects the ID to be unique in out-of-order service mode. Otherwise, AXI bridge must be operated as in-order service mode.
No support for AXI lock transfer when parameter slv_in_order_en is disabled.	N/A	N/A
No support for AXI exclusive transfer.	N/A	N/A
PCIe Advanced Error Reporting (AER) is not supported in the AXI bridge.	N/A	AER is only supported with respect to the native PCIe core. Errors detected by the bridge are not reported as part of AER.
PCIe completions of a split AXI read request in the outbound direction must be returned in-order.	Composer block performs only in-order reassembly.	If two completions for two reads split from an AHB transfer are returned out-of-order from a remote PCIe device, then data will be corrupted at the composer of the AHB bridge.

C.1.5 Clocks and Resets

The AXI Bridge Module employs multiple clock domains. The AXI master channel uses the mstr_aclk clock. The AXI slave channel uses slv_aclk. The native core uses core_clk. The DBI slave channel uses dbi_aclk.

It is possible to have a different clock for each of the PCI Express core, AXI master bus, AXI slave bus, and AXI slave DBI bus. Please see the MSTR_CLK_DIFF_ENABLE, SLV_CLK_DIFF_ENABLE and DBISLV_CLK_DIFF_ENABLE configuration parameters in the Parameters section.

Each of these clock signals has a corresponding reset signal.

We recommend that the application provides the core's reset and AXI resets under the same reset condition. This is due to the lack of a graceful reset protocol built-in between the AXI bridge and the native PCIe core.

The reset signal is asynchronous with a synchronous deassertion, relative to its corresponding clock. Only one reset cycle is required to reset the logic associated with the corresponding clock.

The following is a list of modules associated with each clock domain:

- ❖ core_clk: native PCIe core, generic interface master handler module (gif_core_gm), xadmx module, radmx module, generic interface slave DBI handler module (gif_dbi_gs)
- ❖ mstr_aclk: AXI master interface handler module (DW_PCIE_AXI_GM), generic interface master handler module (gif_core_gm), radmx module for clock crossing of inbound requests, xadmx module for clock crossing of inbound responses
- ❖ slv_aclk: AXI slave interface handler module (DW_PCIE_AXI_GS), generic interface slave handler module (gif_core_gs), xadnx module for clock crossing of outbound requests, radmx module for clock crossing of outbound responses
- ❖ dbi_aclk: AXI slave DBI interface handler module (DW_PCIE_DBI_GS), generic interface slave handler module (gif_dbi_gs). This module supports clock crossing from dbi_aclk to core_clk.

C.1.6 Supported Core Configurations

The AXI Bridge should be configured based on desired core bandwidth and frequency. [Table C-3](#) provides supported core configurations for the AXI Bridge. All supported lanes, frequencies, and databus widths for the DWC PCIe core are also supported for the AXI Bridge. Refer to the DesignWare Cores PCI Express Reference Manual, Appendix A, for the supported core frequencies and sizes.

Table C-3 Core Configurations Supported by AXI Bridge

Core Databus Width			
	32bits DWC PCIe core	64bit DWC PCIe core	128bit DWC PCIe core
AXI Slave	32bit databus	64bit databus	32bit databus
	64bit databus	128bit databus	64bit databus
	128bit databus	256bit databus	128bit databus
	256bit databus		256bit databus
AXI Master	32bit databus	64bit databus	64bit databus
	64bit databus	128bit databus	128bit databus
	128bit databus	256bit databus	256bit databus
	256bit databus		
AXI DBI Slave	32bit databus	32bit databus	32bit databus
AXI Shared DBI	supported	supported	supported

C.1.6.1 AXI Bridge Limitations

The AXI Bridge has the following limitations with regards to core configurations:

- ❖ AXI bus interleave is not supported
- ❖ AXI burst data with holes in the middle of the burst is not supported
- ❖ AXI burst size supported as an AXI master are: one bytes and full bus width in size
- ❖ No order enforcement done in AXI bridge for the responses of inbound requests
- ❖ Order enforcement can be enabled in AXI bridge for responses of outbound requests. The responses for an outbound AXI requests can be in a guaranteed order when configured.

C.2 PCIe AXI Core Operations

C.2.1 Supported AXI Transfer Type

The AXI Bridge Module is compliant with the AMBA 3.0 AXI specification.

C.2.2 Supported AXI Burst Operations

- ❖ For outbound transfers (accessing bridge SLAVE):

The AXI bridge supports only the incremental burst type (INCR) and not the WRAP and FIXED burst types. INCR is used in conjunction with ARLEN and WLEN to define any length of burst.

- ❖ For inbound transfers (using bridge MASTER):

The AXI bridge supports only the incremental burst type (INCR) and not the WRAP and FIXED burst types. INCR is used in conjunction with ARLEN and WLEN to define any length of burst.

Since the PCIe MTU is greater or equal to 128 bytes, a PCIe request with a full MTU worth can be presented onto the AXI bus as an AHB master.

The application can configure the maximum request size that their slaves can take. If the slave's maximum request size is smaller than the PCIe MTU, then the AHB bridge will split an inbound request into two or more requests. The responses of the split inbound request are required to be returned in-order.

C.2.3 Supported AXI Transfer Size

As an AXI master, the AXI bridge only generates transfers with set sizes to simplify the logic. For inbound read requests, master transfers are interpreted to two sizes: the first transfer size is one byte, and the second is the size of the full AXI data bus width. Therefore, when the core's master wants to perform a narrow transfer (i.e., any transfer that is less than the full bus-width), it could choose one of the following two methods.

- ❖ Method 1: Use an “exact” byte count. Assume that memory is not prefetchable. With this method, the core master will perform its access using “1-byte” size (arsize=0). While this method provides precision, it sacrifices performance (to access 3 bytes, the core's master will issue a burst transfer 3-beats long).
- ❖ Method 2: Use an “approximate” byte count. Assume that memory is prefetchable. With this method, the core master will use the entire bus width (assize is set to match the full width of the bus). While this method makes more efficient use of the AXI bus, the core's master reads in more bytes than the intended narrow transfer.

To address the problem of mis-aligned, narrow-transfers, the AXI bridge will automatically select between the “exact” and “approximate” methods based on this threshold number of bytes called “Mis-aligned Narrow Transfer Threshold” (MNT_THRESHOLD). The MNT_THRESHOLD is a number of bytes that is statically selected by the core. It depends on parameter “MASTER_BUS_DATA_WIDTH,” as shown in [Table C-4](#).

Table C-4 MNT_THRESHOLD Values

Core Master Bus Width	MNT_THRESHOLD
MASTER_BUS_DATA_WIDTH = 32 bit	3
MASTER_BUS_DATA_WIDTH = 64 bit	7
MASTER_BUS_DATA_WIDTH = 128 bit	7

When an inbound read is received with a length more than MNT_THRESHOLD bytes, the AXI bridge master generates “full burst size” (i.e., assize matches full width of the bus) reads on its AXI data bus if the request has a length. The assumption is that all inbound reads with a length over MNT_THRESHOLD bytes are headed to the prefetchable memory.

When an inbound read is received with a length less than or equal to MNT_THRESHOLD bytes, the AXI bridge master will use a read burst size of one byte to the non-prefetchable memory at the precise location.

The decision to do a full data bus size read burst or one byte data bus size read is made based on the inbound request's exact byte length.

This automatic selection may be overridden by setting the hidden configuration parameter MSTR_FIXED_SIZE_ENABLE to 1, which will select the “approximate” method i.e. the core master will use the entire bus width.

Refer to the following table for correlations when the address is DWORD-aligned.

The bridge-generated request has these characteristics.			
	arsize	arlen	arburs
If a 3-byte inbound read is received by an AXI bridge with a 32-bit master data bus	0	2	1
If a 4-byte inbound read is received by an AXI bridge with a 32-bit master data bus	2	0	1
If a 5-byte inbound read is received by an AXI bridge with a 32-bit master data bus	2	1	1
If a 3-byte inbound read is received by an AXI bridge with a 64-bit master data bus	0	2	1
If a 4-byte inbound read is received by an AXI bridge with a 64-bit master data bus	0	3	1
If a 8-byte inbound read is received by an AXI bridge with a 64-bit master data bus	3	0	1
If a 9-byte inbound read is received by an AXI bridge with a 32-bit master data bus	3	1	1

Refer to the following table for correlations when the address is not DWORD-aligned (such as address offset of 1).

The bridge-generated request has this characteristics.			
	arsize	arlen	arburs
If a 3-byte inbound read is received by an AXI bridge with a 32-bit master data bus	0	2	1
If a 4-byte inbound read is received by an AXI bridge with a 32-bit master data bus	2	1	1
If a 5-byte inbound read is received by an AXI bridge with a 32-bit master data bus	2	1	1
If a 3-byte inbound read is received by an AXI bridge with a 64-bit master data bus	0	2	1
If a 8-byte inbound read is received by an AXI bridge with a 64-bit master data bus	3	1	1
If a 9-byte inbound read is received by an AXI bridge with a 64-bit master data bus	3	1	1

For AXI master inbound writes, the write strobe ensures a precise write to the intended write request location. All write requests are served accurately.

As an AXI slave, the AXI bridge is designed to support all sizes and burst lengths of an incremental type.

The AXI bridge supports all non-aligned starting addresses for both inbound and outbound transfers.

C.2.4 Early Termination Support

The AHB bridge supports early termination for both the master and slave channels.

- ❖ For outbound transfers (accessing bridge SLAVE):

As per the AXI specification, under EBT conditions, the application master must generate every individual best of the burst with all of the write stores disabled.

- ❖ For inbound transfers (using bridge MASTER):

The AXI bridge as an AXI master is capable of supporting early termination. It will reconstruct the burst after the bus is granted again. This applies for both read and write.

As per the AXI specification, under EBT conditions, the bridge master will generate every individual best of the burst. The application slave must discard the data.

C.2.5 SPLIT Response Mode

Not applicable.

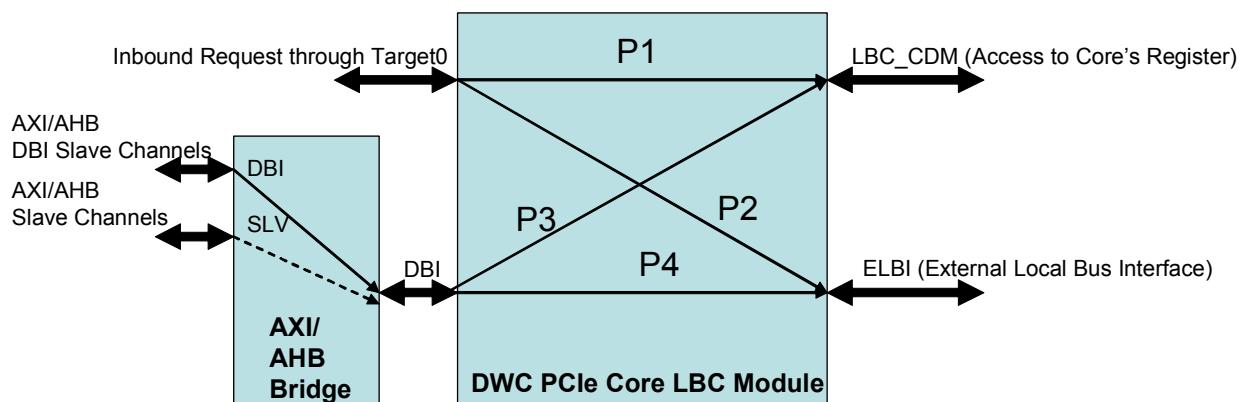
C.2.6 Endianess Support

Not applicable.

C.2.7 Accessing Native PCIe Core Registers

The PCIe Core's DBI interface is used to access the core's CDM or application-specific registers (see [Figure C-4](#)). The application can read from, or write to, native PCIe core registers through the AXI bridge slave DBI interface. The AXI channel read/write address maps to the address of the internal native PCIe core's CDM registers and application-specific register space.

Figure C-4 DWC AXI/AXI-PCIe Core Register Access



- Path1: Enable Inbound request to RD/WR the DWC PCIe core's config space registers
- Path2: Enable inbound request to RD/WR external application specific registers
- Path3: Enable application though DBI to RD/WR DWC PCIe core's config space registers
- Path4: Enable application through DBI to RD/WR application specific registers at ELBI

Note:

Dotted line means optional. This is for Shared DBI mode enabled only at AXI/AHB bridge.

The DWC AXI/PCIe bridge IP optionally provides a dedicated slave interface to access all registers (either DWC native core configuration registers or external application-specific registers). This interface is called the DBI slave.

The DWC AXI/PCIe bridge IP optionally supports a shared register access interface. The application can configure the bridge to have one slave interface where this interface supports access to all core registers, as well as outbound transfers.

The DWC native PCIe core's LBC (local bus controller) module handles the path switching between the native core CDM registers or the application's registers over the native core's ELBI (external local bus interface). The control to the path switching mechanism is specified by the application through the address bits of the AXI slave interface (refer to [Table C-5](#)).

A remote device on the PCIe link can read and write to and from DWC native core configuration registers and external application-specific registers residing on the ELBI bus through an inbound read or write.

Two Important Configurable Parameters

There is a configurable parameter in the native core that enables the multiple function or multiple BAR support at the DBI interface. This parameter is called DBI_MULTI_FUNC_BAR_EN. See "[Parameters](#)" for parameters in the native core. When this parameter is set to enable, an AXI master is required to provide the BAR number and function number through the address field when a register access is issued by the master.

The VALID_DBI_ADDR_WD is a parameter that sets the valid address width for DBI access. This parameter determines the valid address range for a master that issues the access to DWC PCIe core's register, or external application registers. This parameter is set automatically by the AXI bridge when the application configures the DWC AXI core. The default will be 13 if the application indicates the single function and single BAR access to ELBI. Default will be 20 if application indicates the multiple function and single BAR access to ELBI (i.e., DBI_MULTI_FUNC_BAR_EN is not enabled). For multiple function and multiple BARs, or single function and multiple BARs, the default will be set to 21. The value can be adjusted to higher if the application desires to access large BAR sizes that are mapped to ELBI.

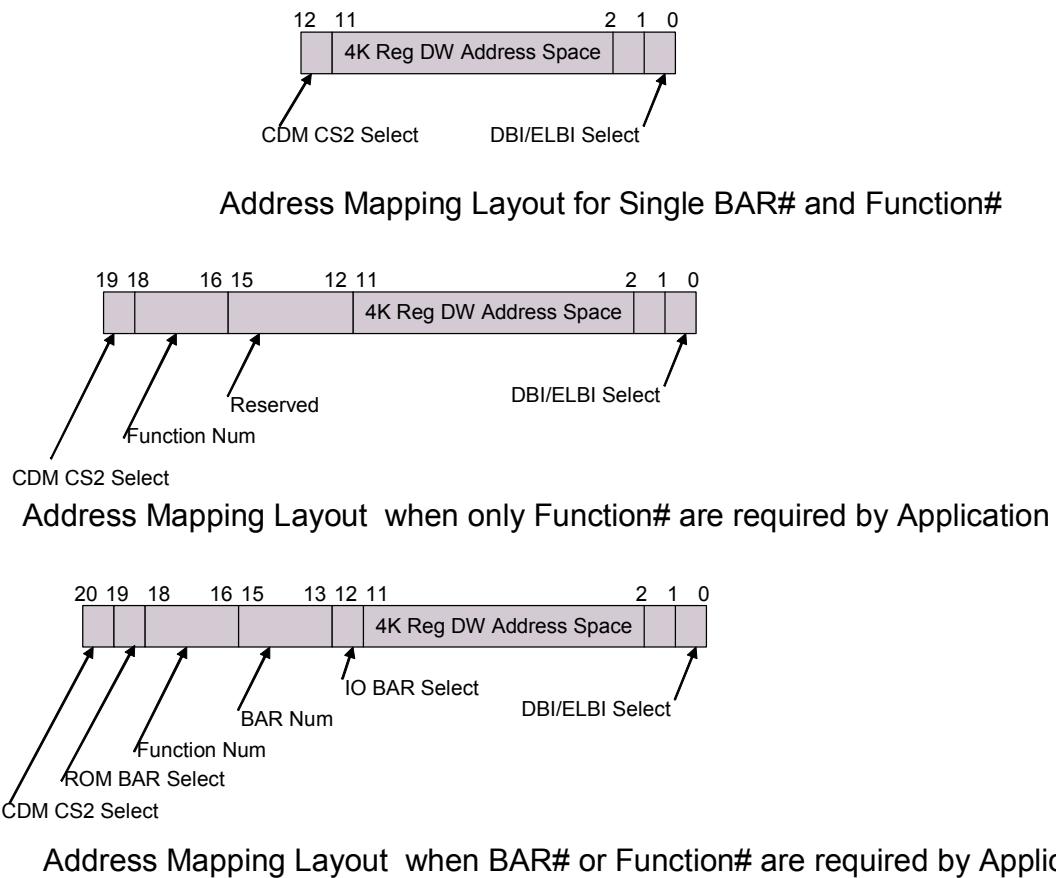
Below is a table of address mapping to DBI access.

Table C-5 AXI Address Control Mapping for the Desired DBI Access

Address Bit Location	Enabled	Control Function	Descriptions
Bit 0	All	CDM/ELBI select	<ul style="list-style-type: none"> 1'b0: Indicates that the AXI transaction intends to read or write to/from the DWC PCIe core's CDM registers. 1'b1 Indicates that the AXI transaction intends to read or write to/from the application registers located at the ELBI interface. <p>Note: Application configuration or memory-mapped registers are all accessed by setting this bit to 1'b1.</p>
Bit [VALID_DBI_ADDR_WD]	All	CDM CS2 select	1'b1 indicates that the AXI transaction intends to read or write to/from the CDM mask registers. Refer to " Data Bus Interface (DBI) " for information on dbi_cs2 functionality.
Bit [VALID_DBI_ADDR_WD -1]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	ROM BAR access	1'b1 indicates that this AXI transaction is accessing a ROM address space residing on the ELBI interface.
Bit [VALID_DBI_ADDR_WD -2: VALID_DBI_ADDR_WD -4]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	Memory Read/Write's Function Number	These 3 bits identify the function number of the PCI function that this AXI transaction is attempting to read or write, via the ELBI interface. This function number is only valid for memory accesses within a valid BAR of a particular function.
Bit [VALID_DBI_ADDR_WD -5: VALID_DBI_ADDR_WD -7]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	Memory Read/Write's BAR number	These 3 bits indicate the BAR number for a memory read or write via the ELBI interface. A value of 3'b111 is reserved to identify that this AXI transaction is not within any valid memory BAR range. This feature allows the application to access the application-specific configuration registers, such as PCIe vendor-specific registers.
Bit [VALID_DBI_ADDR_WD -8]	Enabled only when DBI_MULTI_FUNC_BAR_EN is defined	IO BAR selected	A value of 1'b1 indicates that this AXI transaction is an access within an IO bar range.

Figure C-5 shows address mapping layouts of valid DBI address ranges for various configurations selected by the application.

Figure C-5 Address mapping layout for valid DBI address ranges



C.2.7.1 Reads/Writes to/from the Native Core CDM Registers

There are two categories of registers inside the DWC native PCIe core: basic configuration registers that are PCIe-specified; and PL (Port Logic) that are Synopsys-specific. The DWC AXI bridge has implemented full support to access these registers.

To access CDM registers that belongs to a single function device:

The AXI address of an AXI transaction must be set correctly. Bit 0 set to 1'b0 enables CDM access. If a single function and single BAR at the ELBI is desired, then the DBI_MULTI_FUNC_BAR_EN parameter should not be set when the DWC PCIe core is configured.

To access CDM registers that belong to multiple functions or multiple BARs:

Set the DBI_MULTI_FUNC_BAR_EN parameter when configuring the DWC PCIe core. The function number must be driven by the application onto the proper address location. Address bits[18:16] are the destination function number field for CDM registers.

To access PL registers:

Refer to “[PCIe Registers: Port Logic](#)” for details of PL register mapping. If the application maps a PL register to a BAR, then BAR 0 and function 0 must be set to access PL registers, and bit 0 is set to 1'b0 for CDM access.

C.2.7.2 Reads/Writes to/from the Application-Specific Registers

To access application specific registers that are mapped to memory BAR space:

The application can design its desired registers to map onto either configuration space or memory-mapped BAR space. These registers are accessed through the native DWC PCIe core’s ELBI interface. The AXI address must be set correctly as described in [Table C-5](#). BAR number, function number and other fields must be specified by the application. Bit 0 of the address field must be set to 1'b1 in order to access the application-specific registers.

To access application specific registers that are mapped to configuration space:

As specified in [Table C-5](#), Bit 0 of the address field must be set to 1'b1 to access application registers. The BAR number must be set to 3'b111 to indicate that it is a configuration access.

Default AXI bridge settings:

The default is DBI_MULTI_FUNCT_BAR_EN set to 0, indicating a single function and single BAR at the ELBI interface.

C.2.8 Shared Slave Interface for DBI Access

The AXI bridge supports an independent slave DBI AXI interface to access the native PCIe core's configuration space and external application-specific register space. This DBI slave interface is an optional method used to access the registers. If the application does not select this option, then SHARED_DBI_ENABLED should be selected. [Table C-6](#) identifies the legal settings for these two parameters.

Table C-6 Legal Parameter Setting

DBI_4SLAVE_POPULATE D	SHARED_DBI_ENABLED	Function Configured
1	0	Slave DBI interface is activated
1	1	Illegal Setting. Default to Slave DBI interface is activated
0	0	PCIe Core's DBI interface is not controlled by the AXI bridge
0	1	Shared DBI mode, (The Slave AXI interface is used to access registers)

The following mechanisms are used to identify to the AXI bridge, the destination of a given AXI request, when shared DBI is configured.

For reads, bit 21 of signal slv_awmisc_info (AXI Slave Read Transaction Associated Miscellaneous Information) identifies the type of access by a master. (Refer to [Table C-9](#) for detailed bit descriptions). When this bit is set, this read is a DBI read transfer. Otherwise, it is an outbound read transfer to a remote PCIe device.

For writes, bit 21 of signal slv_armisc_info (AXI Slave Write Transaction Associated Miscellaneous Information) identifies the type of access by a master. (Refer to [Table C-9](#) for detailed bit descriptions). When this bit is set, this write is a DBI write transfer. Otherwise, it is an outbound write transfer to a remote PCIe device.

If the shared DBI mode is not configured, bit 21 of the miscellaneous information has no effect on the rest of AXI bridge logic.

The address mapping layout identified in [Section C.2.7](#) should always be followed, regardless of whether or not a shared DBI is enabled.



Note A master can use an address range or bits of the slave ID to drive this bit of the misc_info bus. Specific mechanisms for driving this bit are beyond the scope of this document.

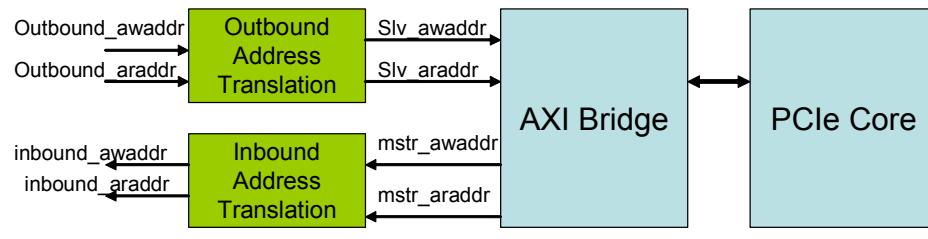
C.2.9 Address Translation When Using the AXI Bridge

The AXI bridge does not provide address translation within the bridge module itself. Normally, for outbound transfers, the address of an AXI master request gets included in a PCIe packet directly as it occurs in AXI. Likewise, the address received from the native PCIe core's TRGT1 interface is presented onto the AXI bridge's master address output, just as it was received in the corresponding PCIe TLP.

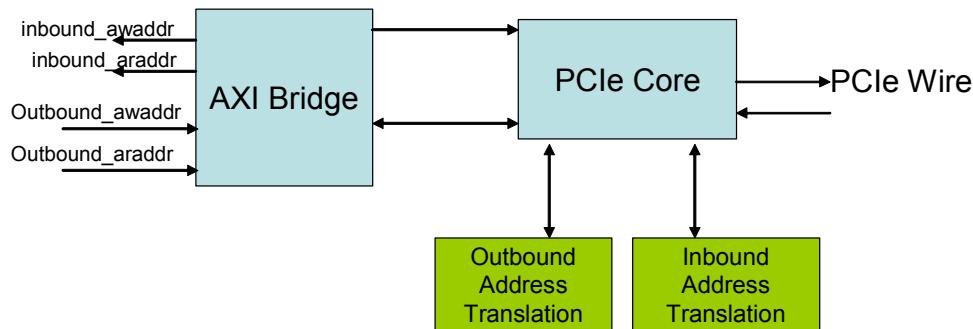
There is an address translation assisting mechanism built into PCIe native core that allows an AXI master's request or an AXI slave's request to be translated. The application is expected to build its desired translations using the native core's address translation assist mechanism.

If the application chooses not to use the native core's address translation assist mechanism, it can always perform its own address translation at the AXI interface. In other words, a desired address translation can be performed through either an application-specific address translation mechanism, through the native core's address translation assist mechanism, or even some combination of the two. [Figure C-6](#) illustrates the two methods of address translation.

Figure C-6 Two Methods of Address Translation



Address Translation Method 1



Address Translation Method 2

In both mechanisms, the application is required to define how many regions are involved for address translation, and how many types of accesses will be supported (such as memory, I/O, configuration or messages, etc.).

C.2.10 Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)

The PCI Express Base 2.0 Specification supports zero-byte transactions as a means of flushing out previous transactions.

C.2.10.1 Inbound Zero-Byte Transfer

Option 1 (CX_FLT_MASK_HANDLE_FLUSH=1) [RECOMMENDED]:

The native PCIe core has a feature to terminate all zero-byte transactions when it is configured to do so. Enable the bit CX_FLT_MASK_HANDLE_FLUSH (default value is 0) in the Filter Mask Register of the native core to enable this feature. If the native PCIe core is configured to support termination of zero-byte transactions, it will generate completions for zero-byte reads and return them automatically to the requester without sending the transaction to the AXI bridge. It will also drop zero-byte writes. Transactions arriving at the PCIe receive queues will be terminated with the ‘flush’ function.

Configuring the native PCIe core to terminate zero-byte transactions is one way of avoiding handling zero-byte reads at the AXI interface. It has the disadvantage of not flushing out any previous transactions remaining in the AXI bridge’s queue buffer. In other words, the flush semantic terminates at the native PCIe core’s receive queues. See [Section 2.3.8.8, “Filtering Rules Not Defined in PCIe Specification” on page 73](#) and [Section Table 4-296, “Filter Mask 2” on page 520](#).

Option 2 (CX_FLT_MASK_HANDLE_FLUSH=0) [NOT SUPPORTED]:

A second possible means of handling zero-byte transactions is to let the AXI bridge receive the inbound zero-byte transaction from the native PCIe core and present it to the AXI interface.

Write: The AXI specification (and the bridge) supports zero byte writes by setting all write strobes (mstr_wstrb) to 0.

Read: The AXI specification, provides no support mechanism for zero-byte reads.

- ❖ The zero-byte read request could be supported using an extra signal that would be part of the mstr_*misc_info busses. This is NOT currently supported.

C.2.10.2 Outbound Zero-Byte Transfer

Write: The AXI specification (and the bridge) supports zero byte writes by setting all write strobes (slv_wstrb) to 0.

- ❖ Therefore, the AXI bridge IS expected to handle outbound zero-byte write transfers.

Read: The AXI specification, provides no support mechanism for zero-byte reads.

- ❖ The zero-byte read request could be supported using an extra signal that would be part of the slv_*misc_info busses. This is NOT currently supported.
- ❖ Therefore, the AXI bridge is NOT expected to handle outbound zero-byte read transfers.

C.2.11 I/O and CFG Transaction Handling

I/O and CFG -type inbound and outbound transfers are fully supported by the AXI bridge.

[Section C.2.9, "Address Translation When Using the AXI Bridge,"](#) also includes information on this type of transfer.

I/O and CFG outbound transfers are signaled to AXI bridge through the slv_awmisc_info or slv_armisc_info busses using the Type field in this bus. Please refer to [Section C.4, "Signal Descriptions,"](#) for the bit location details.

The I/O and CFG outbound write transfers are issued through the bridge's AXI slave write channel. The I/O and CFG outbound read transfers are issued through the bridge's AXI slave read channel.

I/O and CFG inbound transfers are signaled to the AXI master interface by means of the Type field in the mstr_awmisc_info or mstr_armisc_info bus.

The slv_armisc_info and slv_awmisc_info 5-bit Type fields are encoded as follows.

- ❖ 00000: Memory Read or Write Request
- ❖ 00001: Memory Read Request-Locked
- ❖ 00010: IO Read or Write Request
- ❖ 00100: Configuration Read or Write Type 0
- ❖ 00101: Configuration Read or Write Type 1
- ❖ 10***: Message Request
- ❖ 01010: Completion
- ❖ 01011: Completion for Locked Request

Please refer to [Section C.4, "Signal Descriptions,"](#) for additional details.

I/O and CFG inbound write transfers are issued through the AXI master write channel when received from the native PCIe core. I/O and CFG inbound read transfers are issued through the AXI master read channel when received from the native PCIe core. The responses to an inbound master I/O and CFG write or read are expected to be generated by the AXI slave within the application.

Note that there is a special requirement for the master channel. The application slave is required to drive bit [12] in mstr_rmisc_info or mstr_bmisc_info to indicate the type of a response expected. This bit is set to indicate whether the response is to be Posted or Non-Posted.

If a response is for a master-Posted request (such as memory write or message), then the response is a Posted-type response. Therefore, mstr_rmisc_info[12] or mstr_bmisc_info[12] should be deasserted during the response cycle.

If a response is for a master Non-Posted request (configuration read/write, I/O read/write, or memory read), then the response is considered as a Non-Posted response. Therefore, mstr_rmisc_info[12] or mstr_bmisc_info[12] should be asserted to 1'b1 during the response cycle.

If the application does not support a Non-Posted write or read, then bit [12] will be tied to 1'b0 for mstr_bmisc_info and tied to 1'b0 for mstr_rmisc_info. Please refer to [Section C.4, "Signal Descriptions,"](#) for details.

Note: if there is an ATU (address translation unit) attached to the PCIe core's Address Translation Interface and that ATU is converting MemWr TLP's to CfgWr TLP's, then mstr_bmisc_info[12] should be set to 1'b1.

Note: Since all types of reads are non-Posted, then mstr_rmisc_info[12] should always be set to 1'b1 for any type of reads.

C.2.12 Message Transaction Handling

There are many types of PCIe messages defined in the PCI Express Base 2.0 Specification. They are not part of the AXI specification. Therefore, messages of interest must be translated by the AXI bridge or other interface. The message types defined in the PCI Express Base 2.0 Specification include interrupt messages, power management messages, AER messages, vendor messages, and slot messages.

The core provides two methods for message handling. The first message handling method is to have the native PCIe core handle all messages, meaning that the signaling of a particular message is accomplished through the native core's SII interface.

The other message handling method is to allow the native core to pass messages to the AXI bridge, or transmit messages that are generated by the AXI interface. The native core must be configured to pass messages to the AXI bridge in this mode of operation. Inbound messages are handled through the master AXI interface's write channel. Outbound messages are handled through the slave AXI interface's write channel.

There are two ways an application can handle messages at the AXI interface. One is to have the message translated to a memory region. The other is to handle the sideband signals defined in this addendum at the AXI interface (refer to [Section C.2.9](#) for more information).

Message codes and other header information of a message are obtained through the misc_info busses of both inbound and outbound direction. Please refer to [Section C.4, "Signal Descriptions,"](#) for details related to misc_info.

C.2.13 Transaction Order Enforcement Through the AXI Bridge

Order enforcement through the AXI bridge is handled differently for inbound and outbound transactions. Please refer to the order enforcement application note for additional information.

[Figure C-7](#) highlights the AXI bridge internal inbound queuing architecture. [Figure C-8](#) highlights the AXI bridge internal outbound queuing architecture.

Figure C-7 AXI Bridge Inbound Traffic Queuing Architecture

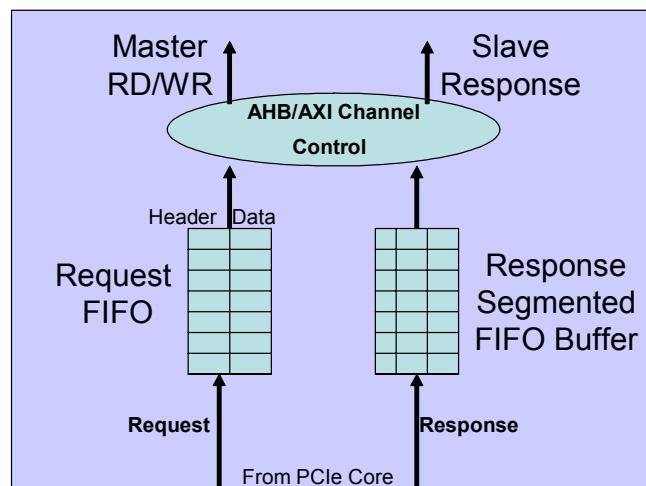
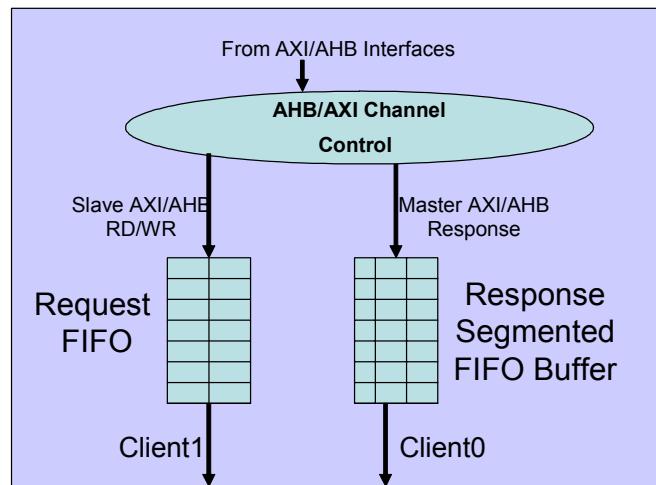


Figure C-8 AXI Bridge Outbound Traffic Queuing Architecture



For inbound transfers:

The AXI bridge master will generate a unique ID for each Non-Posted request and Posted request. The out-of-order servicing of these requests from the AXI interconnect is supported by the AXI bridge master. Each response is identified by the AXI bridge using its response ID. The PCIe interface does support out-of-order completions. The AXI bridge master interface issues a write with a unique ID that is independent from master reads. Since the PCIe Posted transfer expects no response, the AXI bridge drops the write responses. Each PCIe read request can be associated with several AXI master IDs because of the decomposition of the

PCIe read packet within the AXI bridge. Therefore, it is expected (and supported) that the AXI bridge will have a master ID width supporting a larger number of IDs than the maximum number of outstanding read requests configured for the native PCIe core.

For outbound transfers:

The AXI bridge can be configured for either IN_ORDER service or OUT_OF_ORDER service. If the AXI bridge is configured for IN_ORDER service, it will maintain the response order based on its request order. This IN_ORDER service is defined within a particular transfer type. A Non-Posted transfer will be served in its order. A Posted transfer will be served in its order. However, there is no order preservation between Posted and Non-Posted for the current AXI bridge. For example, if a write is issued to the AXI slave interface and is followed by a read and a write, the first write response will be returned before the second write response. However, the second write response may or may not be returned before the read's response. Also, when the AXI bridge is configured for IN_ORDER service, the ID of the AXI slave bus can be identical or unique. The ID bus width can be more or less than the maximum number of outstanding AXI Non-Posted transfers configured. If the AXI bridge is configured for OUT_OF_ORDER service, the bridge will not guarantee any order of response to a set of requests. Therefore, in this case, the application is responsible for generating a unique ID per each outstanding transfer such that it can properly associate responses with the originating requests.



Note It is recommended to have the ID width match the maximum number of outstanding Non-Posted requests allowed (that is, configured) in OUT_OF_ORDER service mode.

For all outbound transfers, the AXI bridge stores the ID from the slave bus and generates a unique internal TAG for each Non-Posted request that the AXI bridge transmits to the native PCIe core. When the response is returned with the internal tag, the AXI bridge performs a lookup for the correct ID that it stored (using the unique internal tag) to return the response with the proper ID.

C.2.14 Memory Read/Write With Data Gaps (“Holes”)

The PCI Express Base 2.0 Specification supports memory reads and writes with “holes” only if the read and write requests are less than or equal to 2 DWORDS of length. The PCIe transaction header first and last byte enable fields are used to signal which bytes are valid. By “hole” we mean that it is possible to have non-contiguous byte enables such as 4'b0101.

AXI writes can generate "holes" in the data by utilizing the AXI write strobes. In the case of reads, AXI sequential read bursts (as specified in the AXI spec) do not support this. A read over the AXI bridge is required to be a sequential read burst. Since a read over the AXI bridge is required to be a sequential read burst, all inbound reads from the PCIe with holes will be "packed" by the AXI bridge, and presented on the AXI bus as a contiguous sequential read.

The AXI bridge outbound read will never contain holes due to AXI protocol requirements for the incremental burst type.

The AXI bridge supports outbound writes with holes for 1 and 2 DWORD-transfers only with the appropriate write-strobes set.

C.2.15 MTU Read Request Size and Page Boundary Support

A PCIe device is programmed to support certain maximum transfer sizes and maximum read request sizes. The native PCIe core's configuration module contains the device's Maximum Transfer Unit (MTU) and maximum read request size information.

The AXI bridge supports sequential burst transfers with a maximum burst length determined by the AXI interconnect. The AXI bridge can support more than the traditional 16-beat maximum burst length, such that bursts up to 4KB may be supported. The maximum AXI transfer length is limited by the data bus width and the maximum AXI burst length, in beats.

The AXI bridge supports mismatches that occur when the AXI maximum transfer length is different than the PCIe MTU and maximum read request size.

The application must define its system requirements such as PCIe MTU and maximum read request size, and the AXI maximum burst length. There are resources allocated within the AXI bridge which are set based on these system parameters.

For example, an outbound read transfer has an associated response buffer that is dependent upon the maximum AXI outbound read request size. An outbound write transfer has an associated outbound transmit buffer that is dependent upon the maximum AXI outbound write request size. An inbound read transfer has an associated response buffer that is dependent upon the remote PCIe device's maximum read request size. An inbound write transfer has a master write buffer that is dependent upon the remote PCIe device's maximum transfer size.

The PCIe defines specifies a page boundary of 4 KB as a requirement. The AXI interconnect typically also has a 4 KB page boundary requirement; however, the AXI bridge can support a non-4K AXI page boundary. In this case, the application must program the AXI bridge to the inbound page boundary required. The inbound page boundaries that the AXI bridge currently supports are 128 bytes, 256 bytes, 512 bytes, 1 KB, 2 KB and 4 KB. The outbound AXI page boundary support is set at 4K bytes based on PCIe's requirement.

C.2.16 Legacy Interrupt Handling and Other Message Handling Over the AXI Bridge

Legacy interrupts can be handled in PCIe, based on either legacy interrupt support through PCIe messages, or through the MSI protocol as a Posted write transfer. The AXI Bridge can handle both of these methods. Refer to “[Message Transaction Handling](#)” if the interrupt message is used to handle legacy interrupts. Furthermore, the bridge can handle vendor messages. How the bridge handles MSI protocol is described in [Section C.2.18, “MSI Handling Over the AXI Bridge.”](#)

C.2.17 PCIe Vendor-Defined Messages

The vendor messages are handled in a manner similar to interrupt messages discussed above. Vendor message can be translated into a memory transfer when address translation is enabled by application. Otherwise the outbound vendor message is created by an AXI master by using the slv_awmisc_info bus. The inbound vendor message is presented by the AXI bridge onto a slave through mstr_awmisc_info. Refer to [Section C.2.9, “Address Translation When Using the AXI Bridge,”](#) and [Section C.4, “Signal Descriptions,”](#) for details.

C.2.18 MSI Handling Over the AXI Bridge

There are two methods to send and receive MSI messages. These are discussed in the following two sections.

C.2.18.1 Using the Regular AXI Interface With a Regular Memory Write Operation

One method is to have the AXI bridge send and receive MSI requests in the same manner as a regular memory write. It is the application's responsibility to form the MSI request based on the native PCIe core's configuration. The native PCIe core CDM block contains the MSI address, enable, etc. The application must obtain the MSI information from the native PCIe core SII interface, or from reading the CDM registers through the DBI interface, and then form the MSI request to present onto the AXI slave interface. This method should be used by an application that preserves the order of Posted transfers requested before an MSI request.

The AXI bridge simply sends and receives MSI requests in the same manner as a memory write. To send an MSI to the PCIe link from the AXI bridge, the application presents the MSI address and data onto the AXI slave write channel. To receive a PCIe MSI packet, the AXI bridge presents the MSI address (like a regular memory write address), and data (like regular memory write data), onto the master write channel.

The native PCIe core outputs `cfg_msi_addr`, which should be used to form a memory write for MSI requests by an AXI master. The native PCIe core outputs `cfg_msi_64`, which should be used to form a 64-bit addressed memory write by an AXI master. The native PCIe core also outputs `cfg_msi_data` as one DWORD of memory write data when an AXI master issues an MSI request. The lower MSB of the data, up to 5 bits, determines the MSI vector number. The application can control the vector number by replacing the lower MSB of the memory write data, up to 5 bits, to support a maximum of 32 MSI vectors. The maximum number of MSI vectors supported is a core configuration parameter.

C.2.18.2 Using the Core's Native Interface (SII)

The second method to send MSI messages is to use the native PCIe core's MSI interface (part of the SII interface group) to signal the native PCIe core to send an MSI request. The native PCIe core has an MSI request formation and transmission capability and does not terminate an MSI request received. This second method is used for applications that do not have restricted ordering. When an MSI message is pending to be sent, it can bypass other previously Posted transfers. An MSI request presented through native PCIe core's MSI interface will be transmitted as soon as Posted credit is available. The MSI request will be received through the native PCIe core's receive interface, just as for a regular memory write. The termination of an MSI request is up to the application. Note, that this interface is synchronous to `core_clk` and not one of the AXI clocks.

C.2.19 Parity Checking Over AXI Bridge

The AXI bridge and native PCIe core together provide parity checking, if enabled, over all address buses and data buses of the bridge. This is done on a per-byte basis. Parity is enabled by means of the configuration Client Data/Address Bus Parity Protection parameter, which is located on the Common Register Configuration page of the native PCIe core configuration settings in coreConsultant. Data parity is also included on a byte-wide basis for all data busses, but although data parity is propagated through the AXI bridge, it is only checked at the bridge interface. Data parity checking is performed by the native PCIe core on outbound transactions and by the application on inbound transactions. The address parity checking also involves all blocks internal to AXI bridge where the address is modified.

The address parity check output bus from the AXI bridge is named `axi_parity_errs`. This bus is optionally present if the parity function has been enabled during configuration.

This bus consists of a concatenation of parity error bits as follows:

- ❖ `axi_parity_errs[0]`: `client1_addr_parerr` – parity error set if any errors detected on the internal slave request module address bus. Synchronous to slave clock.
- ❖ `axi_parity_errs[1]`: `radmx_addr_parerr` – parity error set if any errors detected on the internal receive master request module address bus. Synchronous to core clock.
- ❖ `axi_parity_errs[2]`: `slv_awaddr_parerr` – parity error set if any errors detected on the slave adapter address bus. Synchronous to slave clock.
- ❖ `axi_parity_errs[3]`: `mstr_req_addr_parerr` – parity error set if any errors detected on the internal slave request module address bus. Synchronous to master clock.

Note that these parity errors are synchronized to different clocks, as specified above.

These parity checks are placed inside the AXI bridge because of address modifications, alignment, and page boundary crossing that occurs. For example, if a page boundary is crossed, the decomposer module of the AXI bridge will check the parity at the address bus and perform a parity calculation to the new address that it generates. If there is a parity error detected, it will report parity error onto the proper bit location of the `axi_parity_errs` bus as described above.



Note The AXI bridge or native PCIe core does not perform any parity error correction or take any action on the transfer in which the parity error was detected.

The parity calculation within the AXI bridge is performed by an XOR of every 8 bits with the result placed into the ninth bit. For example, a slave read data bus without parity is defined as `slave_rdata_bus[31:0]` within the AXI bridge. But when parity is enabled on this bus by configuration, the data bus will be presented as `{xor_of_bits31_to_24, slave_rdata_bus[31:24], xor_of_bits23_to_16, slave_rdata_bus[23:16], xor_of_bits15_to_8, slave_rdata_bus[15:8], xor_of_bits7_to_0, slave_rdata_bus[7:0]}`

All AXI bridge address and data buses are defined as above when parity is enabled by the application. All address and data bus inputs are required to have parity bits calculated by the application and presented with the data/address in the same manner as in the above example. All address and data bus outputs are presented with parity bits attached in the same format as in the above example.

The following is a list of all buses that will include parity when enabled during configuration:

- ❖ `mstr_awaddr`
- ❖ `mstr_araddr`,
- ❖ `mstr_wdata`

- ❖ mstr_rdata
- ❖ slv_awaddr
- ❖ slv_araddr,
- ❖ slv_wdata
- ❖ slv_rdata

**Note**

This section describes the parity format and checking for the address and data pipe stages within the AXI bridge. The AXI bridge currently does not support RAM parity checking. Since the AXI bridge can be configured for external RAM, it is expected that the application will add RAM parity checking when configured for external RAM, if so desired.

C.3 Top-Level I/O Diagrams

Figure C-3 shows a block diagram of the DWC PCIe AXI core with respect to signal interfaces. Figure C-9, Figure C-10, and Figure C-11 illustrate the AXI Bridge Module top-level I/O signals. The signal interface group names (grouped according to function) in these figures are cross-referenced to the signal description tables that follow in Section C.4. (For example, click on “AXI Clock and Reset Signals” in Figure C-10 to hyper-link to Table C-7 on page 788 for a description of these signals.)

Figure C-9 AXI Bridge Module Top-Level I/O Diagram (1 of 3)

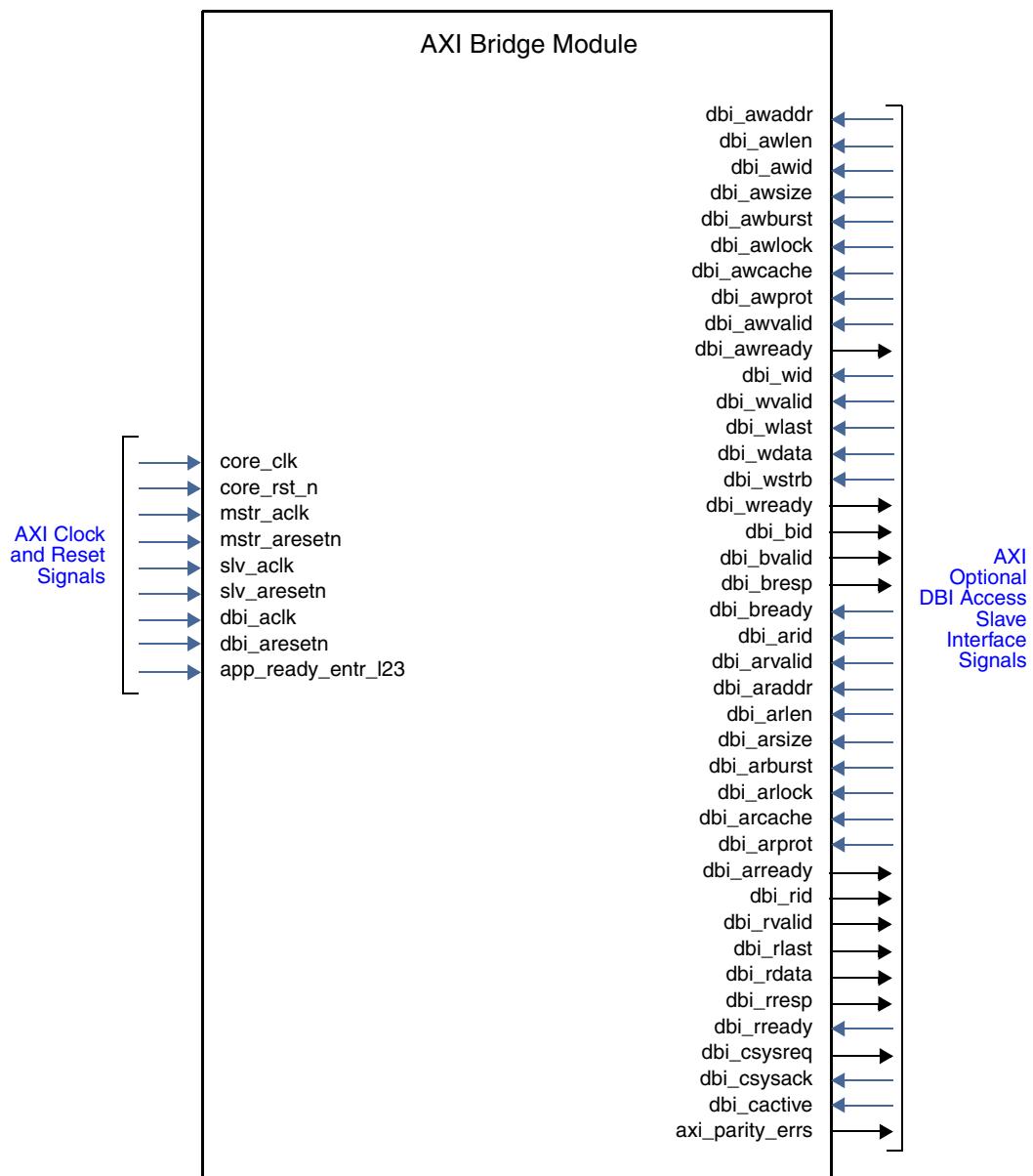


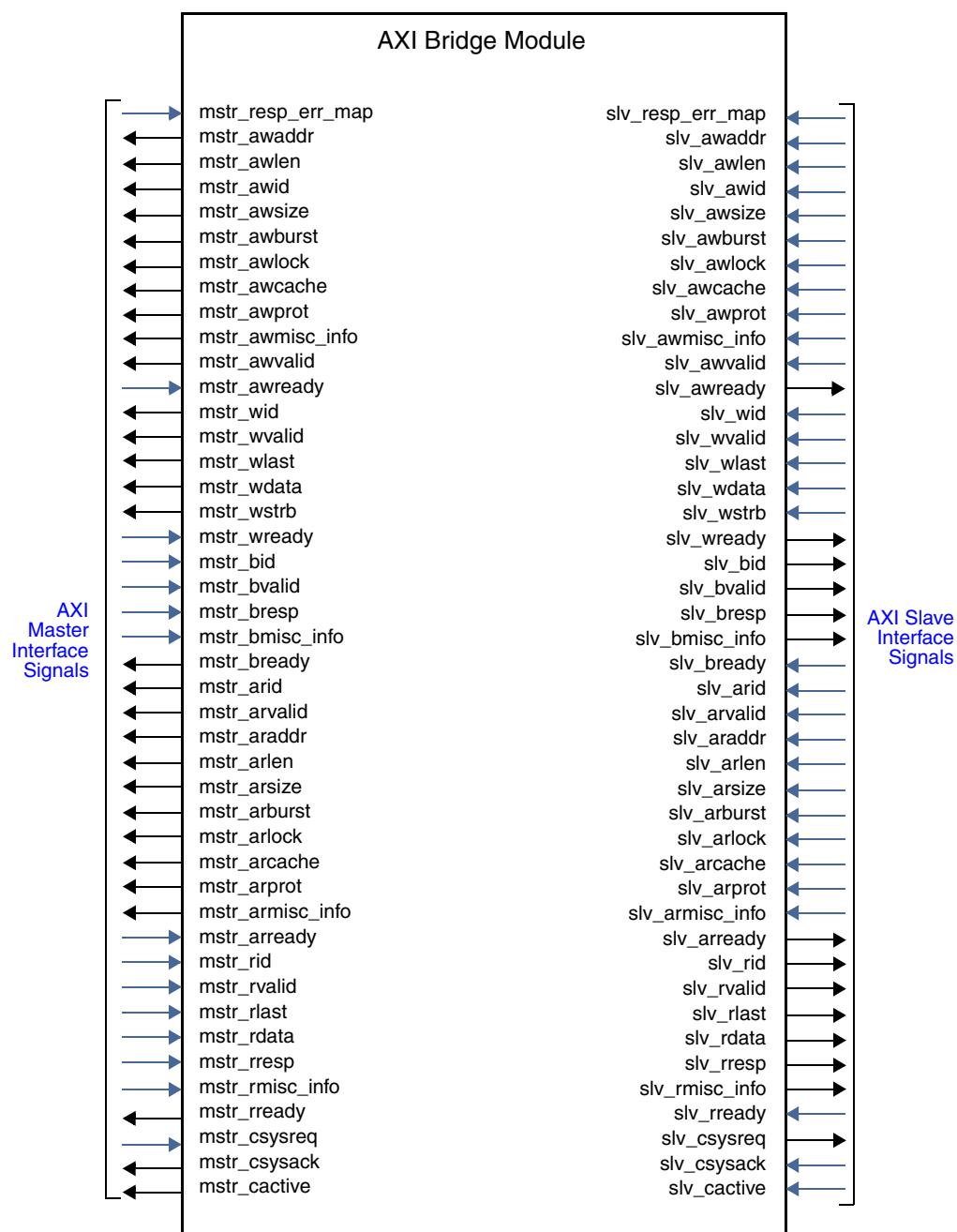
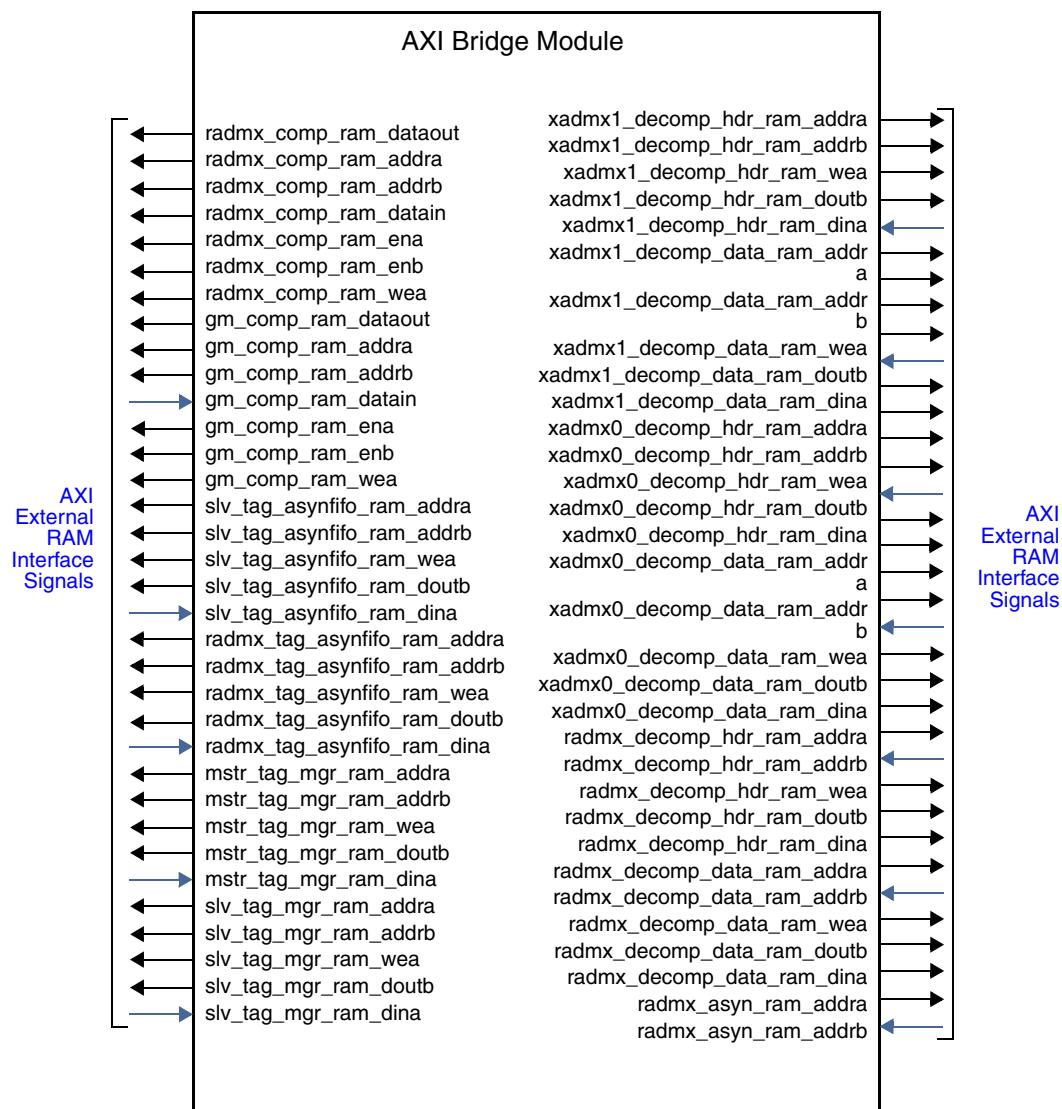
Figure C-10 AXI Bridge Module Top-Level I/O Diagram (2 of 3)

Figure C-11 AXI Bridge Module Top-Level I/O Diagram (3 of 3)

C.4 Signal Descriptions

The following sections describe the AXI Bridge Module top-level interface signals, grouped according to function.



Note Only the top-level AXI signals associated with the AXI Bridge Module itself are described here.

- ❖ [AXI Clock and Reset Signals](#)
- ❖ [AXI Master Interface Signals](#)
- ❖ [AXI Slave Interface Signals](#)
- ❖ [AXI Optional DBI Access Slave Interface Signals](#)
- ❖ [AXI External RAM Interface Signals](#)

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

- ❖ **Registered:** Indicates whether or not the signal is registered directly at the core boundary. 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 core.
- ❖ **Synchronous to:** Indicates which clock the signal is synchronous to, or if the signal is asynchronous.



- All signals with “_n” are active low. For example, signal core_rst_n is an active low signal; that is, it is asserted low.
- Port reset signals are named according to the AXI protocol, and also required to be named as such for coreAssembler. These are also active low. For example, signal slv_aresetn is an active low port reset signal.

C.4.1 AXI Clock and Reset Signals

Table C-7 defines the AXI Bridge Module Clock And Reset signals.

Table C-7 AXI Clock and Reset Signals

Signal	Width (bits)	I/O	Description
core_clk	1	I	<p>Primary Clock</p> <p>Function: The primary clock input to the DM Core. It is assumed that all input signals of the native core are synchronous to this clock, unless otherwise stated. Depending on the DM Core configuration, core_clk is either 62.5 MHz, 125 MHz, 250 MHz, or 500 MHz.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
core_rst_n	1	I	<p>Primary Core Reset Signal</p> <p>Function: The primary reset to the core</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
mstr_aclk	1	I	<p>Optional AXI Master Clock</p> <p>Function: Optional AXI clock for AXI master interface (rather than using core_clk)</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
mstr_aresetn	1	I	<p>Optional AXI Master Reset</p> <p>Function: Optional AXI reset for AXI master interface (rather than using core_rst_n). We recommend that the application reset the master interface when it is asserting a reset to the core.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
slv_aclk	1	I	<p>Optional Slave Clock</p> <p>Function: Optional AXI clock for AXI slave interface (rather than using core_clk)</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
slv_aresetn	1	I	<p>Optional Slave Reset</p> <p>Function: Optional AXI reset for AXI slave interface (rather than using core_rst_n). We recommend that the application reset the slave interface when it is asserting a reset to the core.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>

Table C-7 AXI Clock and Reset Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_aclk	1	I	<p>Optional DBI Clock</p> <p>Function: Optional AXI clock for AXI DBI interface (rather than using core_clk)</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>
dbi_aresetn	1	I	<p>Optional DBI Reset</p> <p>Function: Optional AXI clock for AXI DBI interface (rather than using core_rst_n). We recommend that the application reset the DBI interface when it is asserting a reset to the core.</p> <p>Active State: Low</p> <p>Registered: N/A</p> <p>Synchronous to: N/A</p>

C.4.2 AXI Master Interface Signals

Table C-8 defines the AXI Bridge Module Master Interface signals.

Table C-8 AXI Master Interface Signals

Signal	Width (bits)	I/O	Description
mstr_awaddr	*	O	<p>AXI Master Write Address</p> <p>Function: Specifies the address of the first transfer in a write burst transaction. Associated control signals are used to determine addresses of remaining transfers in the burst.</p> <p>*Width: CC_MSTR__BUS_ADDR_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_awlen	*	O	<p>AXI Master Write Burst Length</p> <p>Function: Specifies the exact number of transfers in a burst; determines the number of data transfers associated with the address.</p> <p>*Width: The pointer width for parameter CC_MSTR_LEN</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_awid	*	O	<p>AXI Master Write ID</p> <p>Function: Indicates ID of the burst. This signal provides a unique ID for each write transfer by default. The application can choose to ignore this signal if an identical ID for each memory write transfer is desired. I/O write transfer responses require this ID. Memory write responses are ignored by the AXI bridge.</p> <p>*Width: The pointer width for parameter CC_MAX_MSTR_TAG</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_awsize	3	O	<p>AXI Master Write Burst Size</p> <p>Function: Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_awburst	2	O	<p>AXI Master Write Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_awlock Note: This signal is not yet supported and all bits are static tied inactive low.	2	O	AXI Master Write Lock Function: Provides additional information about the characteristics of the transfer. Active State: High Registered: No Synchronous to: core_clk or mstr_aclk
mstr_awcache Note: This signal is not yet supported and all bits are static tied inactive low.	4	O	AXI Master Write Cache Function: Indicates bufferable, cacheable, write-through, and write-back attributes of transaction. Active State: High Registered: Yes Synchronous to: core_clk or mstr_aclk
mstr_awprot Note: This signal is not yet supported and all bits are static tied inactive low.	3	O	AXI Master Write Protection Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. Active State: High Registered: Yes Synchronous to: core_clk or mstr_aclk

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_awmisc_info	24	O	<p>AXI Master Write Misc Information</p> <p>Function: AXI Master write transaction associated misc information from the TLP received by the native PCIe core. This is a signal that application can optionally choose. It is not part of standard AXI interface.</p> <ul style="list-style-type: none"> • [2:0]: Transfer's byte offset • [5:3]: Response status suggested by native PCIe core filter • [8:6]: BAR number of Posted TLP • [9]: TLP is an I/O • [10]: TLP is in ROM range • [13:11]: TLP's function number • [15:14]: TLP's attributes • [18:16]: TLP's TC • [23:19]: TLP's Type <p>Notes:</p> <p>Bit 0 to Bit 7 also indicate the message code when the type is message.</p> <p>The Native PCIe core is configured to handle TAG lookup for response of the master request. Therefore it will store the request ID, TC, attributes for its corresponding TAG.</p> <p>Bus number, device number, function number, and register numbers are overlaid to the address bus when type is configuration transactions.</p> <p>Message contents are also overlaid to the address bus when type is a message.</p> <p>Registered: N/A</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_awvalid	1	O	<p>AXI Master Write Address Valid</p> <p>Function: Indicates valid write address and control information are available. Address and control information remain stable until awready signal is high.</p> <ul style="list-style-type: none"> • 0: Address and control information not available • 1: Address and control information available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_awready	1	I	<p>AXI Master Write Address Ready</p> <p>Function: Indicates that the slave is ready to accept address and associated control signals.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_wid	*	O	<p>AXI Master Write ID</p> <p>Function: AXI Master write identification ID. This signal is the ID tag of the write data transfer. The WID value must always match the AWID value of the write transaction.</p> <p>*Width: The pointer width for parameter CC_MAX_MSTR_TAG</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_wvalid	1	O	<p>AXI Master Write Valid</p> <p>Function: Indicates valid write data and strobes are available.</p> <ul style="list-style-type: none"> • 0: Write data and strobes not available • 1: Write data and strobes available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_wlast	1	O	<p>AXI Master Write Last</p> <p>Function: Indicates last transfer in write burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_wdata	*	O	<p>AXI Master Write Data</p> <p>*Width: Width is CC_MSTR_BUS_DATA_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_wstrb	*	O	<p>AXI Master Write Strobes</p> <p>Function: Indicates which byte lanes to update in memory. One write strobe for each eight bits of write data bus. WSTRB[n] is associated with the following byte of MDATA: MDATA[8n+7 : 8n]</p> <p>*Width: CC_MSTR_BUS_DATA_WIDTH /8</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_wready	1	I	<p>AXI Master Write Ready</p> <p>Function: Indicates that slave can accept write data.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clkcore_clk or mstr_aclk</p>
mstr_bid	*	I	<p>AXI Master Write Response Identification Tag</p> <p>Function: The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding. This signal is designed for Non-Posted writes such as a PCIe configuration write or an I/O write, if they are applicable in a particular application. If I/O or configuration writes are not supported in a particular application, this signal can be tied to 0.</p> <p>*Width: The pointer width for parameter CC_MAX_MSTR_TAG</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_bvalid	1	I	<p>AXI Master Write Response Valid</p> <p>Function: I Indicates that valid write response is available.</p> <ul style="list-style-type: none"> • 0: Write response not available • 1: Write response available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_bresp	2	I	<p>AXI Master Write Response</p> <p>Function: Indicates status of write transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_bmisc_info	*	I	<p>AXI Master Write Response Transaction Associated Misc Information (from AXI channel)</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AXI interface.</p> <ul style="list-style-type: none"> • [1:0]: AXI response's attributes • [2]: AXI response with Bad EOT assigned to signal native PCIe core to drop this response • [5:3]: AXI response's TC • [6]: AXI response's BCM • [7]: AXI response's EP • [8]: AXI response's TD • [11:9]: AXI response's Function number • [12]: Indicates that this response is for a Non-Posted transfer. It is used to indicate to the core that the current AXI response should translate into a PCIe response packet (Cpl/CplID). For example, in the case of a MemWr transfer (or a Msg) this bit should be 1'b0, while for CfgWr and IOWr operations it should be 1'b1. <p>Note: if there is an ATU (address translation unit) attached to the PCIe core's Address Translation Interface and that ATU is converting MemWr TLP's to CfgWr TLP's, then bit [12] should be set to 1'b1. See Section C.2.11, “I/O and CFG Transaction Handling.”</p> <p>*Width: The width of this signal is based on MSTR_AXI_MISC_INFO_WD and is 10 by default</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_bready	1	O	<p>AXI Master Response Ready</p> <p>Function: Indicates master can accept response information.</p> <ul style="list-style-type: none"> • 0: Master not ready • 1: Master ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arid	*	O	<p>AXI Master Read Address</p> <p>Function: AXI Master read address identification tag for read address signals. This signal provides a unique ID for each read transfer. If the application desires an identical ID with each read, then the application can ignore the ID field. But it is required that the application return this same ID on its response channel.</p> <p>*Width: The pointer width for parameter CC_MAX_MSTR_TAG</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_arvalid	1	O	<p>AXI Master Read Address Valid</p> <p>Function: When high, indicates read address and control information is valid and remains stable until already is high.</p> <ul style="list-style-type: none"> • 0: Address and control information not valid • 1: Address and control information valid <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_araddr	*	O	<p>AXI Master Read Address</p> <p>Function: Gives initial address of read burst transaction. Only start address of burst is provided, and control signals issued alongside address show how address is calculated for remaining transfers in burst.</p> <p>*Width: CC_MSTR_BUS_ADDR_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arlen	*	O	<p>AXI Master Read Burst Length</p> <p>Function: Gives exact number of transfers in burst; determines number of data transfers associated with address.</p> <p>*Width: the pointer width for parameter CC_MSTR_LEN</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arsize	3	O	<p>AXI Master Read Burst Size</p> <p>Function: Indicates size of each transfer in burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arburst	2	O	<p>AXI Master Read Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arlock	2	O	<p>AXI Master Read Lock</p> <p>Note: This signal is not yet supported and all bits are static tied inactive low.</p> <p>Function: Encoded value indicates whether the transfer is a normal, exclusive, or locked access.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_arcache Note: This signal is not yet supported and all bits are static tied inactive low.	4	O	<p>AXI Master Read Cache</p> <p>Function: Indicates bufferable, cacheable, read-through, and read-back attributes of transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_arprot Note: This signal is not yet supported and all bits are static tied inactive low.	3	O	<p>AXI Master Read Protection</p> <p>Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_armisc_info	24	O	<p>AXI Master Read Transaction Associated Misc Information (from the TLP received by the native PCIe core)</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AXI interface.</p> <ul style="list-style-type: none"> • [2:0]: Transfer's byte offset • [5:3] Response status suggested by native PCIe core filter • [8:6]: BAR number of Posted TLP • [9]: TLP is an I/O • [10]: TLP is in ROM range • [13:11]: TLP's function number • [15:14]: TLP's attributes • [18:16]: TLP's TC • [23:19]: TLP's Type <p>Notes:</p> <ol style="list-style-type: none"> 1. Bit 0 to Bit7 also indicates the message code when the type is message. 2. The native PCIe core is configured to handle TAG lookup for response of the master request. Therefore it will store the request ID and TC, attributes for its corresponding TAG. 3. Bus number, device number, function number, and register numbers are overlaid to the address bus when type is configuration transactions. <ul style="list-style-type: none"> - bus_number = addr[31:24] - dev_number = addr[23:19] - func_number = addr[18:16] - ext_reg_number = addr[11:8] - reg_number = addr[7:2] <p>Registered: N/A</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_arready	1	I	<p>AXI Master Read Address Ready</p> <p>Function: Indicates that slave is ready to accept address and associated control signals.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_rid	*	I	<p>AXI Master Read Identification Tag</p> <p>Function: This signal is required by the AXI bridge to identify a response for a particular master request.</p> <p>*Width: The pointer width for parameter CC_MAX_MSTR_TAG</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_rvalid	1	I	<p>AXI Master Read Valid</p> <p>Function: Indicates that required read data is available and read transfer can complete.</p> <ul style="list-style-type: none"> • 0: Read data not available • 1: Read data available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_rlast	1	I	<p>AXI Master Read Last</p> <p>Function: Indicates last transfer in read burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_rdata	*	I	<p>AXI Master Read Data</p> <p>*Width: CC_MSTR_BUS_ADDR_WIDTH</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_rresp	2	I	<p>AXI Master Read Response</p> <p>Function: Indicates status of read transaction</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_rmisc_info	MSTR_AXI_MISC_INFO_WD = 13	I	<p>AXI Master Read Response Transaction Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of standard AXI interface. This signal is only valid during data phase.</p> <ul style="list-style-type: none"> [1:0]: AXI response's attributes [2]: AXI response with Bad EOT assigned to signal native PCIe core to drop this response [5:3]: AXI response's TC [6]: AXI response's BCM [7]: AXI response's EP [8]: AXI response's TD [11:9]: AXI response's Function number [12]: Indicates that this response is for a Non-Posted transfer. It is used to indicate to the core that the current AXI response should translate into a PCIe response packet (Cpl/CpID). Since this is a read response channel interface, then it should always be set to 1'b1. See Section C.2.11, "I/O and CFG Transaction Handling." <p>Registered: Yes</p> <p>Synchronous to: core_clk or hclk</p>
mstr_rready	1	O	<p>AXI Master Read Ready</p> <p>Function: Indicates master can accept read data and response information.</p> <ul style="list-style-type: none"> 0: Master not ready 1: Master ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_csysreq	1	I	<p>AXI Master System Low-Power Request</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: AXI MASTER System low-power request from system clock controller for peripheral to enter low-power state.</p> <p>Active State: Low</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_csysack	1	O	<p>AXI Master Low-Power Request Acknowledgement</p> <p>Function: Indicates acknowledgement from peripheral of system low-power request.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>

Table C-8 AXI Master Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_cactive Note: AXI Power Modes are not currently supported.	1	O	<p>AXI Master Clock Active</p> <p>Function: Indicates that peripheral requires clock signal:</p> <ul style="list-style-type: none"> • 1: peripheral clock required • 0: peripheral clock not required <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or mstr_aclk</p>
mstr_resp_err_map	2	I	<p>Master Response Error Map</p> <p>Function: This map signal is designed to allow the application to select a report mechanism of master response error received from AXI response channel to CPL status of native PCIe core transmission. When map signal sets to 1, it will set slave or decode errors to UR of PCIe CPL. When map signal sets to 0, it will set slave or decode error to CA of PCIe CPL. Default is set to all 1s.</p> <ul style="list-style-type: none"> • [0]: Slave error • [1]: Decode error <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or mstr_aclk</p>

C.4.3 AXI Slave Interface Signals

Table C-9 defines the AXI Bridge Module Slave Interface signals.

Table C-9 AXI Slave Interface Signals

Signal	Width (bits)	I/O	Description
slv_awaddr	*	I	<p>AXI Slave Write Address</p> <p>Function: Specifies the address of the first transfer in a write burst transaction. Associated control signals are used to determine the addresses of remaining transfers in the burst.</p> <p>*Width: CC_SLV_BUS_ADDR_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awlen	*	I	<p>AXI SLAVE Write Burst Length</p> <p>Function: Specifies the exact number of transfers in a burst; determines number of data transfers associated with the address.</p> <p>*Width: The pointer width for parameter CC_SLV_BURST_LEN</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awid	*	I	<p>AXI Slave Write ID</p> <p>Function: Indicates ID of the write transfer. This signal is an input to the AXI bridge from an AXI master. This signal can contain unique or identical IDs for each AXI write transfer.</p> <p>*Width: CC_SLV_BUS_ID_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awsize	3	I	<p>AXI Slave Write Burst Size</p> <p>Function: Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awburst	2	I	<p>AXI Slave Write Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Note: Only INCR-type bursts are supported.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_awlock	2	I	<p>AXI Slave Write Lock</p> <p>Function: Provides additional information about characteristics of transfer.</p> <p>Note: The core slave does not support Locked/Exclusive transfers</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awcache	4	I	<p>AXI Slave Write Cache</p> <p>Function: Indicates bufferable, cacheable, write-through, and write-back attributes of transaction.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awprot	3	I	<p>AXI Slave Write Protection</p> <p>Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access.</p> <p>Active State: High</p> <p>Registered: N/A</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awmisc_info	22	I	<p>AXI Slave Write Transaction Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AXI interface.</p> <ul style="list-style-type: none"> • [4:0]: AXI transaction's type • [5]: AXI Transaction's BCM • [6]: AXI Transaction's EP • [7]: AXI Transaction's TD • [9:8]: AXI Transaction's ATTR • [12:10]: AXI Transaction's TC • [20:13]: AXI Transaction's MSG code • [21]: AXI transaction is a DBI access. This is for SHARED DBI mode only. <p>Note: The configuration transfer type will have bus number, device number, function number, and register number driven onto address bus[31:0].</p> <ul style="list-style-type: none"> • bus_number = addr[31:24] • dev_number = addr[23:19] • func_number = addr[18:16] • ext_reg_number = addr[11:8] • reg_number = addr[7:2] <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_awvalid	1	I	<p>AXI Slave Write Address Valid</p> <p>Function: Indicates valid write address and control information are available. Address and control information remain stable until awready signal is high.</p> <ul style="list-style-type: none"> • 0: Address and control information not available • 1: Address and control information available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_awready	1	O	<p>AXI Slave Write Address Ready</p> <p>Function: Indicates that the slave is ready to accept the address and associated control signals.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_wid	*	I	<p>AXI Slave Write Identification</p> <p>Function: AXI Slave write identification tag of a write data transfer. It should be the same as slv_awid for each write transfer. Since we support only sequential access where we match each address channel transfer to write data channel transfer in order, this signal is not used internally.</p> <p>*Width: CC_SLV_BUS_ID_WIDTH</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_wvalid	1	I	<p>AXI Slave Write Valid</p> <p>Function: Indicates valid write data and strobes are available.</p> <ul style="list-style-type: none"> • 0: Write data and strobes not available • 1: Write data and strobes available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_wlast	1	I	<p>AXI Slave Write Last</p> <p>Function: Indicates last transfer in write burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_wdata	*	I	<p>AXI Slave Write Data</p> <p>*Width: CC_SLV_BUS_DATA_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_wstrb	*	I	<p>AXI Slave Write Strobes</p> <p>Function: Indicates which byte lanes to update in memory. One write strobe for each eight bits of write data bus. WSTRB[n] is associated with the following byte of MDATA:</p> <p>MDATA[8n+7 : 8n]</p> <p>*Width: CC_SLV_BUS_DATA_WIDTH /8</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_wready	1	O	<p>AXI Slave Write Ready</p> <p>Function: Indicates that slave can accept write data.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_bid	*	O	<p>AXI Slave Write Response Identification Tag</p> <p>Function: This signal indicates the write response ID which correlates to a transfer that has an identical ID</p> <p>*Width: CC_SLV_BUS_ID_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_bvalid	1	O	<p>AXI Slave Write Response Valid</p> <p>Function: Indicates that valid write response is available.</p> <ul style="list-style-type: none"> • 0: write response not available • 1: write response available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_bresp	2	O	<p>AXI Slave Write Response</p> <p>Function: Indicates status of write transaction.</p> <p>Note: EXOKAY response is not supported.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_bmisc_info	11	O	<p>AXI Slave Write Transaction's Response Associated Misc Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AXI interface.</p> <ul style="list-style-type: none"> [0]: AXI response transaction's BCM [1]: AXI response transaction's TD [4:2]: AXI response transaction's TC [6:5]: AXI response transaction's attributes [9:7]: AXI response transaction's status defined as the same location of PCIe completion header status field [10]: Indicates that AXI response is for a non-posted request <p>Registered: Only registered when SLV_CLK_DIFF_ENABLE==1 and RAM is synchronous read port.</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_bready	1	I	<p>AXI Slave Response Ready</p> <p>Function: Indicates SLAVE can accept response information.</p> <ul style="list-style-type: none"> 0: Slave not ready 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arid	*	I	<p>AXI Slave Read Address</p> <p>Function: The ID of a slave read transfer. This signal is an input to the AXI bridge from an AXI master. This signal can contain unique or identical IDs for each AXI write transfer. The bridge is statically configured as either in-order services or out of order services.</p> <p>*Width: CC_SLV_BUS_ID_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arvalid	1	I	<p>AXI Slave Read Address valid</p> <p>Function: When high, indicates read address and control information is valid and remains stable until arready is high.</p> <ul style="list-style-type: none"> 0: Address and control information not valid 1: Address and control information valid <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_araddr	*	I	<p>AXI Slave Read Address</p> <p>Function: Gives initial address of read burst transaction. Only the start address of the burst is provided, and control signals issued alongside the address show how the address is calculated for the remaining transfers in the burst.</p> <p>*Width: CC_SLV_BUS_ADDR_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arlen	*	I	<p>AXI Slave Read Burst Length</p> <p>Function: Gives the exact number of transfers in the burst; determines the number of data transfers associated with the address.</p> <p>*Width: The pointer width for parameter CC_SLV_BURST_LEN</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arsize	3	I	<p>AXI Slave Read Burst Size</p> <p>Function: Indicates size of each transfer in burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arburst	2	I	<p>AXI Slave Read Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Note: Only INCR-type bursts are supported.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arlock	2	I	<p>AXI Slave Read Lock</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Encoded value indicates whether the transfer is a normal, exclusive, or locked access.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_arcache	4	I	<p>AXI Slave Read Cache</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates bufferable, cacheable, read-through, and read-back attributes of transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_arprot	3	I	<p>AXI Slave Read Protection</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_armisc_info	22	I	<p>AXI Slave Read Transaction Associated Miscellaneous Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of the standard AXI interface.</p> <ul style="list-style-type: none"> • [4:0] AXI transaction's type • [5] AXI Transaction's BCM • [6] AXI Transaction's EP • [7] AXI Transaction's TD • [9:8] AXI Transaction's attribute • [12:10] AXI Transaction's TC • [20:13] AXI Transaction's MSG code • [21]: AXI transaction is a DBI access. This is for SHARED DBI mode only. <p>Note: The configuration transfer type will have bus number, device number, function number, and register number driven onto address bus[31:0]</p> <ul style="list-style-type: none"> • bus_number = addr[31:24] • dev_number = addr[23:19] • func_number = addr[18:16] • ext_reg_number = addr[11:8] • reg_number = addr[7:2] <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_already	1	O	<p>AXI Slave Read Address Ready</p> <p>Function: Indicates that the slave is ready to accept address and associated control signals.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_rid	*	O	<p>AXI Slave Read Identification Tag</p> <p>Function: This signal indicates the read response ID which correlates to a request with an identical ID.</p> <p>*Width: CC_SLV_BUS_ID_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_rvalid	1	O	<p>AXI Slave Read Valid</p> <p>Function: Indicates that required read data is available and read transfer can complete.</p> <ul style="list-style-type: none"> • 0: Read data not available • 1: Read data available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_rlast	1	O	<p>AXI Slave Read Last</p> <p>Function: Indicates last transfer in read burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_rdata	*	O	<p>AXI Slave Read Data</p> <p>*Width: CC_SLV_BUS_DATA_WIDTH</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>
slv_rresp	2	O	<p>AXI Slave Read Response</p> <p>Function: Indicates status of read transaction.</p> <p>Note: EXOKAY response is not supported.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_rmisc_info	11	O	<p>AXI Slave Read Transaction's Response Associated Miscellaneous Information</p> <p>Function: This is a signal that the application can optionally choose. It is not part of standard AXI interface.</p> <ul style="list-style-type: none"> [0]: AXI response transaction's BCM [1]: AXI response transaction's TD [4:2]: AXI response transaction's TC [6:5]: AXI response transaction's attributes [9:7]: AXI response transaction's status defined as the same location of PCIe completion header status field [10]: Indicates that AXI response is for a non-posted request <p>Registered: Synchronous to: core_clk or slv_aclk</p>
slv_ready	1	I	<p>AXI Slave Read Ready</p> <p>Function: Indicates slave can accept read data and response information.</p> <ul style="list-style-type: none"> 0: Slave not ready 1: Slave ready <p>Active State: High Registered: No Synchronous to: core_clk or slv_aclk</p>
slv_csysreq	1	I	<p>AXI Slave System (Low-Power) Request</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: AXI Slave System low-power request from system clock controller for peripheral to enter low-power state.</p> <p>Active State: Low Registered: Yes Synchronous to: core_clk or slv_aclk</p>
slv_csysack	1	O	<p>AXI Slave Low-Power Request Acknowledgement</p> <p>Function: Indicates acknowledgement from peripheral of system low-power request.</p> <p>Active State: High Registered: Yes Synchronous to: core_clk or slv_aclk</p>
slv_cactive	O	I	<p>AXI Slave Clock Active</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: Indicates that peripheral requires clock signal:</p> <ul style="list-style-type: none"> 1: Peripheral clock required 0: Peripheral clock not required <p>Active State: High Registered: Yes Synchronous to: core_clk or slv_aclk</p>

Table C-9 AXI Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
slv_resp_err_map	6	I	<p>Slave Response Error Map</p> <p>Function: This map signal is designed to allow the application to select a report mechanism of slave response error received from PCIe completion. Since there are 6 kinds of PCIe completion error that core can report to AXI interface, therefore there are 6 bit mapping to select these errors corresponding to SLVERR or DECERR. Default is 1 which selects SLVERR.</p> <ul style="list-style-type: none"> • [0]: CPL UR • [1]: Not used • [2]: CPL CA • [3]: CPL Poisoned • [4]: CPL Time-out • [5]: CPL Abort (unexpected) <p>Registered: No</p> <p>Synchronous to: CPL Timeout</p>

C.4.4 AXI Optional DBI Access Slave Interface Signals

[Table C-10](#) defines the AXI Bridge Module optional DBI Access Slave Interface signals.

Table C-10 AXI Optional DBI Access Slave Interface Signals

Signal	Width (bits)	I/O	Description
dbi_awaddr	32	I	<p>AXI Slave Write Address</p> <p>Function: Provides the initial address of a write burst transaction. This address addresses the CDM registers.</p> <p>Note: There is a special address decode to generate dbi_cs2 to the core (please refer to the native core data book for its function). The core uses dbi_cs2 to access the Bar Mask registers. There is a parameter (VALID_DBI_ADDR_WD) that can be programmed by the application to control the address decode location of dbi_cs2.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awlen	4	I	<p>AXI Slave Write Burst Length</p> <p>Specifies the exact number of transfers in a burst; determines the number of data transfers associated with the address.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awid	4	I	<p>AXI Slave Write ID</p> <p>Function: Indicates ID of the burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awsize	3	I	<p>AXI Slave Write Burst Size</p> <p>Function: Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awburst	2	I	<p>AXI Slave Write Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_awlock	2	I	<p>AXI Slave Write Lock</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Provides additional information about characteristics of transfer.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awcache	4	I	<p>AXI Slave Write Cache</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates bufferable, cacheable, write-through, and write-back attributes of transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_awprot	3	I	<p>AXI Slave Write Protection</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_awvalid	1	I	<p>AXI Slave Write Address Valid</p> <p>Function: Indicates that valid write address and control information are available. Address and control information remain stable until awready signal is high.</p> <ul style="list-style-type: none"> • 0: Address and control information not available • 1: Address and control information available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_awready	1	O	<p>AXI Slave Write Address Ready</p> <p>Function: Indicates that the slave is ready to accept address and associated control signals.</p> <p>0: Slave not ready</p> <p>1: Slave ready</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_wid	4	I	<p>AXI Slave Write Identification</p> <p>Note: This signal is not used in the current design.</p> <p>Function: AXI slave write identification tag of write data transfer. The AXI master always sets this value to 0 on all bits.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_wvalid	1	I	<p>AXI Slave Write Valid</p> <p>Function: Indicates valid write data and strobes are available.</p> <ul style="list-style-type: none"> • 0: Write data and strobes not available • 1: Write data and strobes available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_wlast	1	I	<p>AXI Slave Write Last</p> <p>Function: Indicates last transfer in write burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_wdata	32	I	<p>AXI Slave Write Data.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_wstrb	4	I	<p>AXI Slave Write Strobes</p> <p>Function: Indicates which byte lanes to update in memory. One write strobe for each eight bits of write data bus. WSTRB[n] is associated with the following byte of MDATA: MDATA[8n+7 : 8n]</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_wready	1	O	<p>AXI Slave Write Ready</p> <p>Function: Indicates that slave can accept write data.</p> <ul style="list-style-type: none"> • 0: Slave not ready • 1: Slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_bid	4	O	<p>AXI Slave Write Response Identification Tag</p> <p>Function: This signal is not actually required to be connected; it is not used and therefore is not connected internal to the component.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_bvalid	1	O	<p>AXI Slave Write Response Valid</p> <p>Function: Indicates that valid write response is available.</p> <ul style="list-style-type: none"> • 0: write response not available • 1: write response available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_bresp	2	O	<p>AXI Slave Write Response</p> <p>Function: Indicates status of write transaction.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_bready	1	I	<p>AXI Slave Response Ready</p> <p>Function: Indicates SLAVE can accept response information.</p> <ul style="list-style-type: none"> • 0: SLAVE not ready • 1: SLAVE ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arid	4	I	<p>AXI Slave Read Address</p> <p>Function: AXI Slave read address identification tag for read address signals. The DW_axi_gm always sets this value to 0 on all bits.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arvalid	1	I	<p>AXI Slave Read Address Valid</p> <p>Function: When high, indicates read address and control information is valid and remains stable until already is high.</p> <ul style="list-style-type: none"> • 0: address and control information not valid • 1: address and control information valid <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_araddr	32	I	<p>AXI Slave Read Address</p> <p>Function: Provides the initial address of a read burst transaction. This address addresses the CDM registers.</p> <p>Note: There is a special address decode to generate dbi_cs2 to the core (please refer to the native core data book for its function). The core uses dbi_cs2 to access the Bar Mask registers. There is a parameter (VALID_DBI_ADDR_WD) that can be programmed by the application to control the address decode location of dbi_cs2.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arlen	4	I	<p>AXI Slave Read Burst Length</p> <p>Function: Gives the exact number of transfers in a burst; determines the number of data transfers associated with the address.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arsize	3	I	<p>AXI Slave Read Burst Size</p> <p>Function: Indicates size of each transfer in burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arburst	2	I	<p>AXI Slave Read Burst</p> <p>Function: Determines the address for each transfer within a burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arlock	2	I	<p>AXI Slave Read Lock</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Encoded value indicates whether the transfer is a normal, exclusive, or locked access.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_arcache	4	I	<p>AXI Slave Read Cache</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates bufferable, cacheable, read-through, and read-back attributes of transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_arprot	3	I	<p>AXI Slave Read Protection</p> <p>Note: This signal is not used in the current design.</p> <p>Function: Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_arready	1	O	<p>AXI Slave Read Address Ready</p> <p>Function: Indicates that the slave is ready to accept the address and associated control signals.</p> <ul style="list-style-type: none"> • 0: slave not ready • 1: slave ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_rid	4	O	<p>AXI Slave Read Identification Tag</p> <p>Function: This signal is not actually required to be connected; it is not used and therefore is not connected internal to the component.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_rvalid	1	O	<p>AXI Slave Read Valid</p> <p>Function: Indicates that required read data is available and read transfer can complete.</p> <ul style="list-style-type: none"> • 0: read data not available • 1: read data available <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_rlast	1	O	<p>AXI Slave Read Last</p> <p>Function: Indicates last transfer in read burst.</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_rdata	32	O	<p>AXI Slave Read Data</p> <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
dbi_rresp	2	O	<p>AXI Slave Read Response</p> <p>Function: Indicates status of read transaction.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_rready	1	I	<p>AXI Slave Read Ready</p> <p>Function: Indicates slave can accept read data and response information.</p> <ul style="list-style-type: none"> • 0: SLAVE not ready • 1: SLAVE ready <p>Active State: High</p> <p>Registered: No</p> <p>Synchronous to: dbi_aclk</p>
dbi_csysreq	1	O	<p>AXI Slave System Low-Power Request</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: AXI slave System low-power request from system clock controller for peripheral to enter low-power state.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_csysack	1	I	<p>AXI Slave Low-power Request Acknowledgement</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: Indicates acknowledgement from peripheral of system low-power request.</p> <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>
dbi_cactive	1	I	<p>AXI Slave Clock Active</p> <p>Note: AXI Power Modes are not currently supported.</p> <p>Function: Indicates that peripheral requires clock signal:</p> <ul style="list-style-type: none"> • 1: Peripheral clock required • 0: Peripheral clock not required <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to: dbi_aclk</p>

Table C-10 AXI Optional DBI Access Slave Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
axi parity errs	4	O	<p>AXI Parity Errors</p> <p>Function: Indicates parity errors on address buses based on checks performed in the AXI bridge:</p> <ul style="list-style-type: none"> • [0]: client1_addr_parerr • [1]: radmx_addr_parerr • [2]: slv_awaddr_parerr • [3]: mstr_req_addr_parerr <p>Active State: High</p> <p>Registered: Yes</p> <p>Synchronous to:</p> <ul style="list-style-type: none"> • [1]: core_clk • [0], [2]: slave axi clock • [3]: master clock

C.4.5 AXI External RAM Interface Signals

[Table C-11](#) defines the AXI Bridge Module External RAM Interface signals. Descriptions of the RAMs are provided in [Section C.6.5](#).

Table C-11 AXI External RAM Interface Signals

Signal	Width (bits)	I/O	Description
Composer RAM Ports			
Notes:			
<ul style="list-style-type: none"> RADMX composer RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True GM composer RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True and MASTER_POPULATED=True 			
radmx_comp_ram_dataout	*	O	RAM Read Data *Width: [RADMX_COMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_addrA	*	O	RAM Port A address (write interface) *Width: [(RADMX_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_addrB	*	O	RAM Port B Address (read interface) *Width: [(RADMX_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_datain	*	I	RAM Write Data *Width: [(RADMX_COMPOSER_DATAQ_WD)-1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_enA	1	O	RAM Port A Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_comp_ram_enB	1	O	RAM Port B Enable (read enable) Active State: N/A Registered: No Synchronous to: core_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_comp_ram_wea	1	O	RAM Port A Write Enable Active State: N/A Registered: No Synchronous to: core_clk
gm_comp_ram_dataout	*	O	RAM Read Data *Width: [MSTR_COMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_addrA	*	O	RAM Port A Address (write interface) *Width: [(MSTR_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_addrB	*	O	RAM Port B Address (read interface) *Width: [(MSTR_COMPOSER_DATAQ_PW)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_datain	*	I	RAM Write Data *Width: [(MSTR_COMPOSER_DATAQ_WD)-1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_enA	1	O	RAM Port A Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_enB	1	O	RAM Port B Enable (read enable) Active State: N/A Registered: No Synchronous to: mstr_aclk
gm_comp_ram_weA	1	O	Ram Port A Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
Tag RAM Ports			
Note:			
			<ul style="list-style-type: none"> Slave TAG asynchronous RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True and SLV_CLK_DIFF_ENABLE=True
slv_tag_asynfifo_ram_addrA	*	O	Slave Tag Asynchronous FIFO RAM Port A Address *Width: [CLIENT1_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to:
slv_tag_asynfifo_ram_addrB	*	O	Slave Tag Asynchronous FIFO RAM Port B Address *Width: [CLIENT1_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_asynfifo_ram_weA	1	O	Slave Tag Asynchronous FIFO RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
slv_tag_asynfifo_ram_doutB	*	O	Slave Tag Asynchronous FIFO RAM Port B Data Out *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: core_clk
slv_tag_asynfifo_ram_dinA	*	I	Slave Tag Asynchronous FIFO RAM Port A Address *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
Decomposer RAM Ports			
Note:			
			<ul style="list-style-type: none"> RADMX TAG Asynchronous header and data RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True and MASTER_POPULATED and MSTR_CLK_DIFF_ENABLE=True
radmx_tag_asynfifo_ram_addrA	*	O	RADMX Tag Asynchronous FIFO RAM Port A Address *Width: [RADMX_TS_DATAQ_ADDR_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_tag_asynfifo_ram_addrb	*	O	RADM Tag Asynchronous FIFO RAM Port B Address *Width: [CLIENT1_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
radmx_tag_asynfifo_ram_wea	1	O	RADM Tag Asynchronous FIFO RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_tag_asynfifo_ram_doutb		O	RADM Tag Asynchronous FIFO RAM Port B Data Out *Width: [RADMX_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
radmx_tag_asynfifo_ram_dina	*	I	RADM Tag Asynchronous FIFO RAM Port A Data In *Width: [RADMX_TS_DATAQ_WIDTH -1:0] Active State: N/A Registered: No Synchronous to: core_clk
mstr_tag_mgr_ram_addrb	*	O	Master Tag RAM Port A Address *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
mstr_tag_mgr_ram_wea	*	O	Master Tag RAM Port B Address *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
mstr_tag_mgr_ram_wea	1	O	Master Tag RAM Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
mstr_tag_mgr_ram_doutb	*	O	Master Tag RAM Port B Data Out *Width: [MAX_MSTR_TAG_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
mstr_tag_mgr_ram_dina	*	I	<p>Master Tag RAM Port A Data In</p> <p>*Width: [MAX_MSTR_TAG_PW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: mstr_aclk</p>
Composer RAM Ports			
Note:			
<ul style="list-style-type: none"> XADMX1 header and data RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True 			
slv_tag_mgr_ram_addr	*	O	<p>Slave Tag RAM Port A Address</p> <p>*Width: [SLV_DECOMP_TAG_DP_PW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p>
slv_tag_mgr_ram_addrb	*	O	<p>Slave Tag RAM Port B Address</p> <p>*Width: [SLV_DECOMP_TAG_DP_PW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: slv_aclk</p>
slv_tag_mgr_ram_wea	1	O	<p>Slave Tag RAM Port B Address</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p>
slv_tag_mgr_ram_doutb	*	O	<p>Slave Tag RAM Port B Data Out</p> <p>*Width: [MAX_WIRE_TAG_PW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: slv_aclk</p>
slv_tag_mgr_ram_dina	*	I	<p>Slave Tag RAM Port A Data In</p> <p>*Width: [MAX_WIRE_TAG_PW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: core_clk</p>
xadmx1_decomp_hdr_ram_addr	*	O	<p>XADMX1 Decomposer Header RAM Port A Address</p> <p>*Width: [XADMX_CLIENT1_QUEUE_HPW -1:0]</p> <p>Active State: N/A</p> <p>Registered: No</p> <p>Synchronous to: slv_aclk</p>

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx1_decomp_hdr_ram_addrb	*	O	XADM1 Decomposer Header RAM Port B Address *Width: [XADMX_CLIENT1_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_hdr_ram_wea	1	O	XADM1 Decomposer Header RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_hdr_ram_doutb	*	O	XADM1 Decomposer Header RAM Port B Data Out *Width: [XADMX_CLIENT1_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_hdr_ram_dina	*	I	XADM1 Decomposer Header RAM Port A Data In *Width: [XADMX_CLIENT1_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_a_ddra	*	O	XADM1 Decomposer Data RAM Port A Address *Width: [XADMX_CLIENT1_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_a_ddrb	*	O	XADM1 Decomposer Data RAM Port B Address *Width: [XADMX_CLIENT1_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx1_decomp_data_ram_w_ea	1	O	XADM1 Decomposer Data RAM Write Enable Active State: N/A Registered: No Synchronous to: slv_aclk
xadmx1_decomp_data_ram_d_outb	*	O	XADM1 Decomposer Data RAM Port B Data Out *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx1_decomp_data_ram_di_na	*	I	XADM1 Decomposer Data RAM Port A Data In *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
RADMX Asynchronous RAM Ports			
Note:			
• RADMX asynchronous RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True			
xadmx0_decomp_hdr_ram_addr_a	*	O	XADM0 Decomposer Header RAM Port A Address *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_hdr_ram_addrb	*	O	XADM0 Decomposer Header RAM Port B Address *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_hdr_ram_wea	*	O	XADM0 Decomposer Header RAM Write Enable *Width: [XADMX_CLIENT0_QUEUE_HPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_hdr_ram_doutb	*	O	XADM0 Decomposer Header RAM Port B Data Out *Width: [XADMX_CLIENT0_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_hdr_ram_dina	*	I	XADM0 Decomposer Header RAM Port A Data In *Width: [XADMX_CLIENT0_QUEUE_HWD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
xadmx0_decomp_data_ram_a_ddra	*	O	XADM0 Decomposer Data RAM Port A Address *Width: [XADMX_CLIENT0_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
xadmx0_decomp_data_ram_a ddrb	*	O	XADM0 Decomposer Data RAM Port B Address *Width: [XADMX_CLIENT0_QUEUE_DPW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_data_ram_w ea	1	O	XADM0 Decomposer Data RAM Write Enable Active State: N/A Registered: No Synchronous to: mstr_aclk
xadmx0_decomp_data_ram_d outb	*	O	XADM0 Decomposer Data RAM Port B Data Out *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
xadmx0_decomp_data_ram_di na	*	I	XADM0 Decomposer Data RAM Port A Data In *Width: [XADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_hdr_ram_ addrA	*	O	XADM Decomposer Header RAM Port A Address *Width: [RADMX_DECOMPOSER_HDRQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_hdr_ram_ addrB	*	O	XADM Decomposer Header RAM Port B Address *Width: [RADMX_DECOMPOSER_HDRQ_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_hdr_ram_ wea	1	O	XADM Decomposer Header RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_hdr_ram_ doutb	*	O	XADM Decomposer Header RAM Port B Data Out *Width: [RADMX_DECOMPOSER_HDRQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_decomp_hdr_ram_dina	*	I	XADM Decomposer Header RAM Port A Data In *Width: [RADMX_DECOMPOSER_HDRQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_addr	*	O	XADM Decomposer Data RAM Port A Address *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_addrb	*	O	XADM Decomposer Data RAM Port B Address *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_data_ram_wea	*	O	XADM Decomposer Data RAM Write Enable *Width: [RADMX_DECOMPOSER_DATAQ_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk
radmx_decomp_data_ram_doutb	*	O	XADM Decomposer Data RAM Port B Data Out *Width: [RADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: mstr_clk
radmx_decomp_data_ram_dina	*	I	XADM Decomposer Data RAM Port A Data In *Width: [RADMX_DECOMPOSER_DATAQ_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk
RADMX Asynchronous RAM Ports			
Note:			
RADMX asynchronous RAMs are present when AXI_POPULATED=True and AXI_RAM_EXTERNAL=True and SLV_CLK_DIFF_ENABLE=True			
radmx_asyn_ram_addr	I	O	RADM Asynchronous RAM Port A Address *Width: [FIFO_PW -1:0] Active State: N/A Registered: No Synchronous to: core_clk

Table C-11 AXI External RAM Interface Signals (Continued)

Signal	Width (bits)	I/O	Description
radmx_asyn_ram_addrb	*	O	RADM Asynchronous RAM Port B Address *Width: [FIFO_PW -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
radmx_asyn_ram_wea	1	O	RADM Asynchronous RAM Write Enable Active State: N/A Registered: No Synchronous to: core_clk
radmx_asyn_ram_doutb	*	O	RADM Asynchronous RAM Port B Data Out *Width: [FIFO_WD -1:0] Active State: N/A Registered: No Synchronous to: slv_aclk
radmx_asyn_ram_dina	*	I	RADM Asynchronous RAM Port A Data In *Width: [FIFO_WD -1:0] Active State: N/A Registered: No Synchronous to: core_clk

C.4.6 AXI Bridge SII Interface Addition

The native PCIe core has an SII interface that varies from configuration to configuration. Because the native PCIe core is instantiated together with the AXI bridge core, the top-level SII signals for the AXI bridge will also vary from configuration to configuration. Refer to “[System Information Interface \(SII\)](#)” for details on the PCIe/ AXI bridge product’s top-level I/O signals, other than those signals described within this appendix.

Most of the SII signals are not needed for a basic AXI configuration. Refer to “[System Information Interface \(SII\)](#)” when the application has special requirements such as additional power management control, advanced PCIe feature enabled, MSIX support, etc.

The following lookup error signals are the only ones generated by the AXI bridge to notify the application that there is an overflow that occurred in a lookup table of either inbound or outbound responses. These indicate that there was a violation for the number of outstanding Non-Posted requests issued for the inbound or outbound direction.

- ❖ gm_cmposer_lookup_err
- ❖ radmx_cmposer_lookup_err

The following two signals are for parity error detection. These signals indicate that either the AXI bridge or native PCIe core has detected parity errors at either the address or data bus. It is expected that the application will choose a parity error handing scheme by monitoring these signals. One way to use these signals is to "OR" them to create an interrupt.

- ❖ app_parity_errs
- ❖ axi_parity_errs

C.5 Parameters

The AXI Bridge Module is populated based on the functions you select with the coreConsultant configuration parameters. [Table C-12](#) defines these configuration parameters.

Table C-12 Configuration Parameters

coreConsultant Option Name	Description
AXI Enable	Function: Enables the AXI Bus interface to the native PCIe core Value Range: On, Off Default Value: Off Parameter Name: AXI_POPULATED
Slave DBI Enable	Function: Enables the AXI Slave DBI interface Value Range: On, Off Default Value: On Parameter Name: DBI_4SLAVE_POPULATED
Master Interface Enable	Function: Enables the AXI Master interface Value Range: On, Off Default Value: On Parameter Name: MASTER_POPULATED
Enable Independent Master Clock	Function: Enables an independent clock (other than core_clk) for the AXI Master bus Value Range: On, Off Default Value: On Parameter Name: MSTR_CLK_DIFF_ENABLE
Enable Independent Slave Clock	Function: Enables an independent clock (other than core_clk) for the AXI Slave bus Value Range: On, Off Default Value: On Parameter Name: SLV_CLK_DIFF_ENABLE
Enable Independent AXI DBI Slave Clock	Function: Enables an independent clock (other than core_clk) for the AXI Slave DBI bus Value Range: On, Off Default Value: On Parameter Name: DBISLV_CLK_DIFF_ENABLE
Maximum Master Tags Supported	Function: Specifies the maximum number of transfers supported by the AXI bridge when performing as an AXI master NOTE: A higher value chosen here results in a higher gate count. Value Range: 2, 4, 8, 16, 32, 64, 128, 256 Default Value: 32 Parameter Name: CC_MAX_MSTR_TAG

Table C-12 Configuration Parameters (Continued)

coreConsultant Option Name	Description
Remote Device Max Read Request Size	<p>Function: Specifies the maximum read request size supported by the PCIe remote device when AXI or AXI is enabled. This parameter is used to size the master composer memories inside the AXI adapter.</p> <p>Value Range: 128, 256, 512, 1k, 2k, 4k</p> <p>Default Value: 128</p> <p>Parameter Name: CX_REMOTE_RD_REQ_SIZE</p>
Master Address Width	<p>Function: Specifies the width of the AXI Master Address bus</p> <p>Value Range: 32, 64</p> <p>Default Value: 32</p> <p>Parameter Name: CC_MSTR_BUS_ADDR_WIDTH</p>
Master Data Width	<p>Function: Specifies the width of the AXI Master Data bus</p> <p>Value Range: 32, 64, 128, 256</p> <p>Default Value: 32</p> <p>Parameter Name: CC_MSTR_BUS_DATA_WIDTH</p>
Master Burst Length Width	<p>Function: Specifies the maximum burst length of an AXI transfer at the master interface. This parameter controls the resources that will be assigned inside the AXI bridge for maximum transfer burst length. If undefined length is supported, then this parameter must be set for the maximum of any AXI transfer under undefined length burst. This parameter has the size in burst cycle. Zero represents 1 burst of a data bus width worth of data.</p> <p>Value Range: 16 to 256</p> <p>Default Value: 16</p> <p>Parameter Name: CC_MSTR_BURST_LEN</p>
Master Page Boundary Size	<p>Function: Specifies the page boundary size supported within the AXI interconnect when the AXI bridge is performing as a master. No packet can have an address that crosses this boundary. Packets will be split to conform to this requirement.</p> <p>Value Range: 128, 256, 512, 1k, 2k, 4k</p> <p>Default Value: 4k</p> <p>Parameter Name: CC_MSTR_PAGE_BOUNDARY_PW</p>
Master Request Header FIFO Q Depth	<p>Function: Specifies AXI Master Request Header FIFO Queue size</p> <p>Value Range: 4 to 260</p> <p>Default Value: See 4.</p> <p>Parameter Name: CC_RADMX_DECOMPOSER_HDRQ_DP</p>
Master Request DATA FIFO Q Depth	<p>Function: Specifies AXI Master Request Data FIFO Queue size</p> <p>Value Range: Unlimited integer</p> <p>Default Value: See 34.</p> <p>Parameter Name: CC_RADMX_DECOMPOSER_DATAQ_DP</p>

Table C-12 Configuration Parameters (Continued)

coreConsultant Option Name	Description
Maximum Slave Tags Supported	<p>Function: Specifies the maximum number of transfers supported by the AXI bridge when performing as an AXI slave</p> <p>Note: A higher value chosen here results in higher gate count.</p> <p>Value Range: 32, 64, 128, 256</p> <p>Default Value: 32</p> <p>Parameter Name: CC_MAX_SLV_TAG</p>
Slave Data Width	<p>Function: Specifies the width of the AXI Slave data bus</p> <p>Value Range: 32, 64, 128, 256</p> <p>Default Value: 32</p> <p>Parameter Name: CC_SLV_BUS_DATA_WIDTH</p>
Slave Address Width	<p>Function: Specifies the width of the AXI Slave address bus</p> <p>Value Range: 32, 64</p> <p>Default Value: 32</p> <p>Parameter Name: CC_SLV_BUS_ADDR_WIDTH</p>
Slave Burst Length Width	<p>Function: Specifies the maximum burst length of an AXI transfer at the slave interface. This parameter controls the resources that will be assigned inside the AXI bridge for maximum transfer burst length. If undefined length is supported, then this parameter must be set for the maximum of any AXI transfer under undefined length burst. This parameter has the size in burst cycle. Zero represents 1 burst of a data bus width worth of data.</p> <p>Value Range: 16 and 256</p> <p>Default Value: 16</p> <p>Parameter Name: CC_SLV_BURST_LEN</p>
AXI Slave ID Width	<p>Function: Specifies the width of the AXI Slave ID</p> <p>Value Range: 1 to 8</p> <p>Default Value: 4</p> <p>Parameter Name: CC_SLV_BUS_ID_WIDTH</p>
Slave Request Header FIFO Q Depth	<p>Function: Specifies AXI bridge slave request Header FIFO sizeN</p> <p>Value Range: 4 to 260</p> <p>Default Value: See 4.</p> <p>Parameter Name: CC_XADMX_CLIENT1_QUEUE_HDP</p>
Slave Request DATA FIFO Q Depth	<p>Function: Specifies AXI bridge slave request Data FIFO size</p> <p>Value Range: Unlimited integer</p> <p>Default Value: See 18.</p> <p>Parameter Name: CC_XADMX_CLIENT1_QUEUE_DDP</p>

C.6 Implementation Guidelines

The implementation guidelines presented in this chapter are intended as a guide through a basic implementation of a PCIe IP and AXI bridge instantiation. The reference design demonstrates a basic application where the AXI bridge, the DWC PCIe IP core, and Synopsys PHY are instantiated. The example RTL code is not intended to be part of the completed/validated DWC PCIe IP product offering.

Configuration parameters are discussed in [Section C.6.4 on page 858](#), and RAMs in [Section C.6.5 on page 861](#).

C.6.1 Scope of the Implementation Guidelines

A structural chip-level reference design is provided to facilitate building your PCIe device(s). The structural reference design contains the instantiation and interconnections of the DWC PCIe AXI bridge, IP core, PHY, and additional reference design modules. These other reference design modules demonstrate how to build a PCIe device in a quick and straight-forward fashion. The chip-level reference design provides a basic example of how to drive and monitor the DWC PCIe bridge IP core inputs and outputs.

The example chip top RTL (`chip_top.v`) is located under `<config*>/src/customer/generic/`). The chip top RTL illustrates an example hierarchy for a PCIe device with the PCIe bridge IP core, PHY, and AXI bridge instantiated. There are several example RTL modules other than the IP core that are also provided to support the PCIe bridge IP core integration into a PCIe device. We have also made a number of basic assumptions which are applicable to most applications, and are necessary in order for us to configure and interconnect the AXI bridge, PCIe core and PHY.

These assumptions are as follows:

1. The inbound and outbound AXI transfers are all memory transfers within AMBA memory space. The application intends to use the address translation feature offered by the PCIe core to map AMBA memory regions onto PCIe configuration, IO, message and memory regions. In other words, the application does not intend to use our sideband AXI signals for PCIe Type and Format translations.
2. The application supports both MSI and legacy interrupts via the PCIe core MSI and legacy interrupt interfaces. This means that an MSI and legacy interrupt controller is directly connected to the PCIe core's MSI and legacy interrupt interfaces. The application is responsible to generate the `app_intr_req` and `app_intr_clr` signals. Note, that this assumption requires that no order relation exists between the application AXI transfers and the interrupt event. If ordering is required, the application can modify the outbound address translation module to get MSI requests to properly map into an AMBA memory region.
3. All outbound messages are supported through the address translation provided in the reference design. The PCIe core Vendor Message interface is not activated. All inbound messages are handled by the PCIe core except for Vendor Messages, which are handled by the inbound address translation logic. Inbound Vendor Messages will be translated into AXI memory writes.
4. The Slave DBI interface is activated in this reference design. If an application desires to have a shared DBI slave interface, it will just need to configure the AXI bridge setting for Shared DBI.
5. The External Local Bus Interface (ELBI) is used to design customer-specific registers and reference design-related registers. If the application has its own register access mechanism, it will need to modify the `elbi_driver.v` to accommodate that. This reference design has been set up to use the PCIe core ELBI interface to access extended application registers and registers designed for the reference design.

Since most signals of the PCIe bridge IP are heavily dependent on application-specific functionality, the example RTL modules are there to provide comments on key issues of the core interfaces, and a typical example of how to use them. We expect the customer to modify any or all of the example modules, as needed.

With the above assumptions in mind, this reference design comprises the following major modules:

- ❖ **axi_master_driver.v**: A generic AXI master driver to drive the master response AXI bridge interface
- ❖ **axi_master_sideband_driver.v**: A master Sideband driver to drive the AXI bridge master response interface sideband signals
- ❖ **axi_slave_sideband_driver.v**: A slave Sideband driver to drive the AXI bridge slave interface sideband signal
- ❖ **axi_slave_driver.v**: A generic AXI slave driver to drive the AXI bridge slave interface
- ❖ **ob_addr_translation_driver.v**: An outbound address translation driver to accomplish the outbound address translation from AMBA memory space to PCIe memory space or PCIe transfer type, such as IO, configuration and message type
- ❖ **ib_addr_translation_driver.v**: An Inbound address translation driver to accomplish the address translation from PCIe memory space, type and format to AMBA memory space
- ❖ **clk_RST.v**: A clock and reset module to drive the clocks and resets that are required by PCIe PHY, DWC PCIe core, and AXI bridge.
- ❖ **pm_ctrl_driver.v**: An extended power management control module to drive the extended power management functionalities the application desires.

All inputs and outputs of the PCIe core and AXI bridge are set or monitored through the reference design based on the above assumptions and modules. This will greatly ease the IP integration effort. Customers need only to concentrate on the connections of the generic (standard) AXI master and slave drivers in this reference design, and application-specific functionalities. The PCIe related functions are completely handled and the interface is simplified to just the SerDes IO pins at this chip-top design.

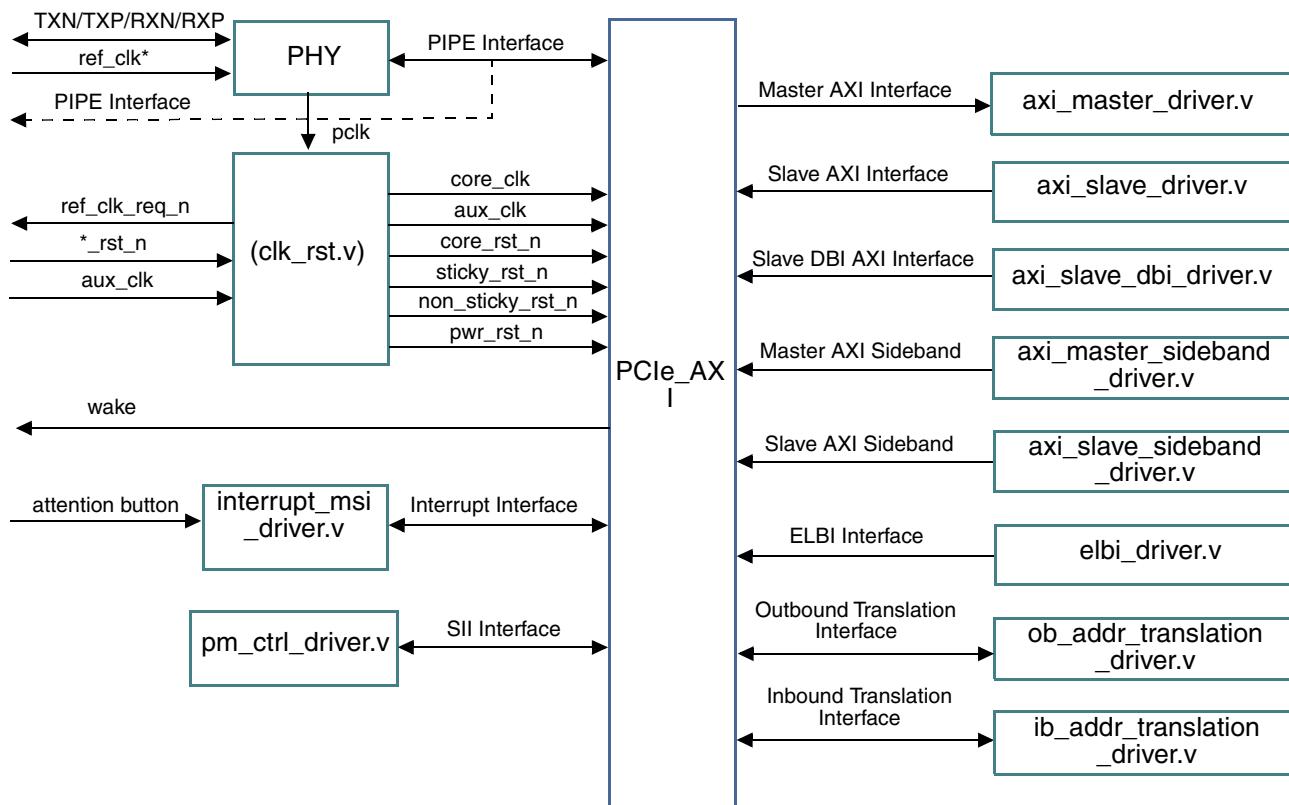
C.6.2 Detailed Descriptions of the Structural Chip-Level Reference Design

The following sections provide an overview of the reference design and then a brief description of each of the major components. The reference code itself is intended to provided additional details on the requirements of each interface and its associated functionality.

C.6.2.1 Chip Top Block Diagram

The architecture is designed to have example modules instantiated together with the IP core, as shown in Figure C-12.

Figure C-12 chip_top.v Architecture



Module functions are as follows:

- ❖ **PCIE_AXI_Bridge**: The configured AXI bridge IP core. It contains the PCIe core and AXI bridge.
- ❖ **PHY**: This is a module for either the SNPS PHY model or another Pipe PHY model configured by the user.
- ❖ **Clock and Reset Control (clk_RST.v)**: Provides an example of how to drive the clocks and resets expected by the core, PHY and AXI bridge.
- ❖ **MSI and Legacy Interrupt Control (interrupt_msi_driver.v)**: Provides an example of how to initiate a legacy interrupt or an MSI request.
- ❖ **PM MISC Control (pm_ctrl.v)**: Provides an example of how to drive the power management inputs that are designed to enable the application logic to have more control over the low-power states of the core.

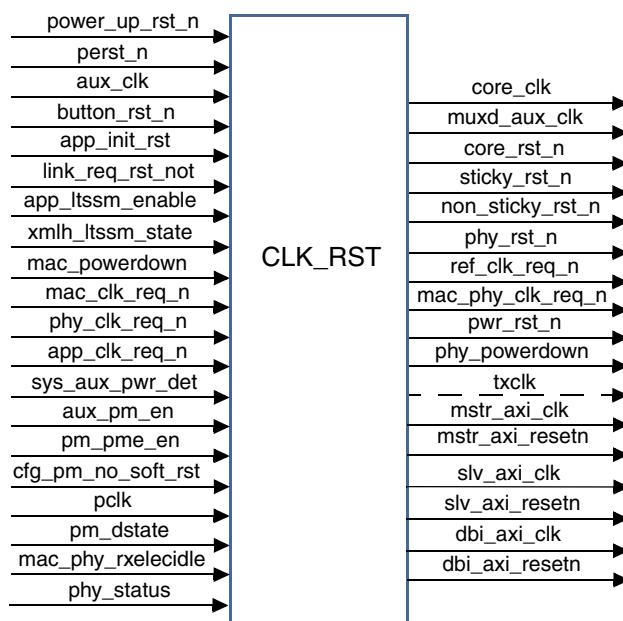
- ❖ **Master AXI Interface Driver (`axi_master_driver.v`):** Enables customers to build a response driver to the DWC PCIe AXI bridge master interface. This driver is connected directly to the response portion of the DWC PCIe AXI bridge master interface. (This module acts like an AMBA slave to the DWC PCIe AXI bridge.)
- ❖ **Slave AXI Interface (`axi_slave_driver.v`):** Enables customers to build an AMBA master. This driver is connected directly to the DWC PCIe AXI bridge slave interface. (This module acts like an AMBA master to the DWC PCIe AXI bridge.)
- ❖ **Master AXI Sideband Interface (`axi_master_sideband_driver.v`):** Provides a driving capability for the DWC PCIe AXI bridge master interface sideband signals to accommodate the extra information associated with a master response. This driver is connected directly to the DWC PCIe AXI bridge master interface sideband signals.
- ❖ **Slave AXI Sideband Interface (`axi_slave_sideband_driver.v`):** Provides a driving capability for the DWC PCIe AXI bridge slave interface sideband signals to accommodate the extra information associated with a master request. This driver is connected directly to the DWC PCIe AXI bridge slave interface sideband signals.
- ❖ **Application Registers (`elbi_driver.v`):** Provides a reference design to drive the ELBI interface to access the registers required for the reference design. Note, that the application can modify this module to replace these registers with their own application-specific registers.
- ❖ **Outbound Address Translation (`ob_addr_translation_driver.v`):** Provides the address translation on outbound transfers. It maps the AMBA memory region to a PCIe memory region, PCIe Configuration request, IO request, or Message.
- ❖ **Inbound Address Translation (`ib_addr_translation_driver.v`):** Provides the address translation on inbound transfers. It maps an incoming PCIe transfer onto the desired AMBA memory region.

C.6.2.2 Clock and Reset Control Module

The Clock and Reset Control module (`clk_RST.v`) is an example RTL module driving the core's clock and reset inputs for a typical application. This module primarily covers the following functions:

- ❖ The core's clock and reset control
- ❖ AUX clock control
- ❖ The PHY's reset
- ❖ The AXI interface clock and reset
- ❖ Generation of the external reference clock request signal

[Figure C-13](#) illustrates the I/O requirement of this reference module.

Figure C-13 Clock and Reset Controller I/O

There are several core output signals that affect the clock and reset circuitry of a PCIe device due to the power state or link status of the IP core.

The inputs to the clock and reset module are from four sources:

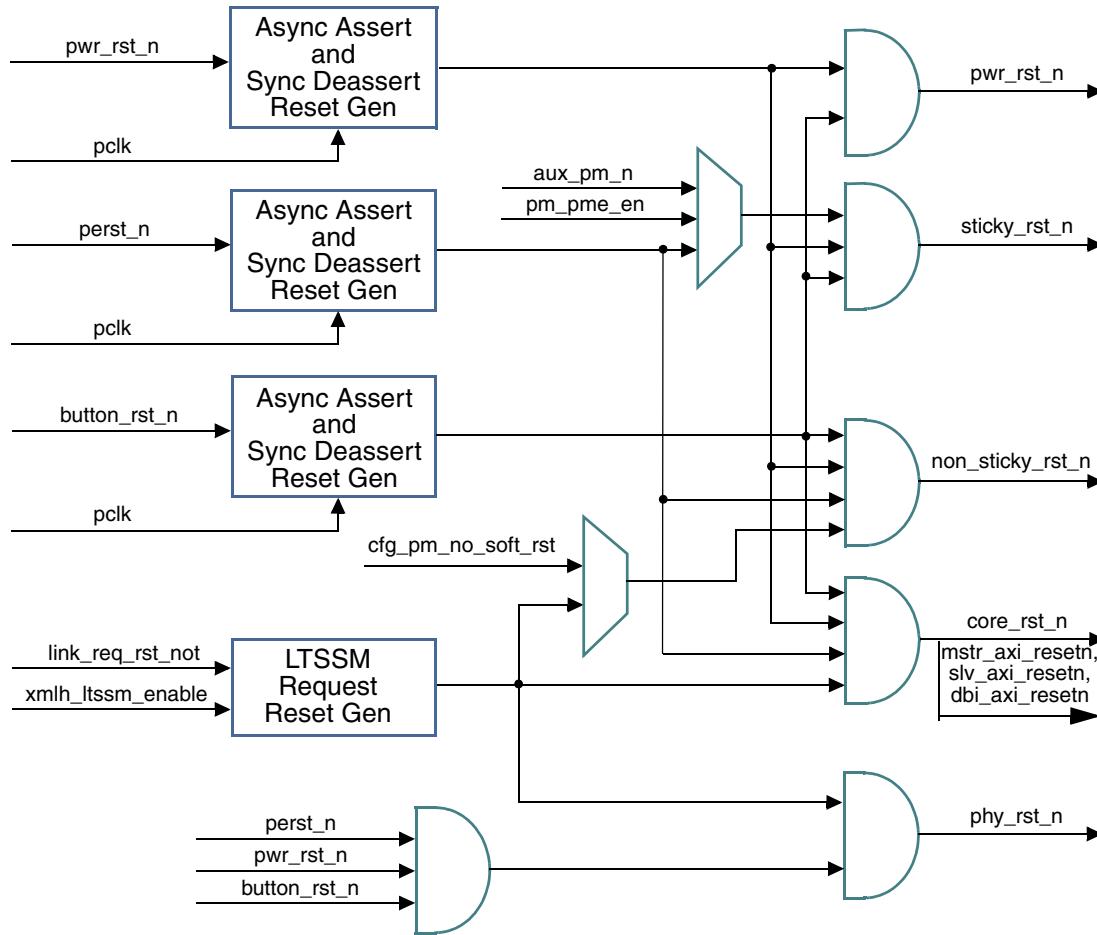
- ❖ Chip inputs: power_up_rst_n, perst_n, aux_clk, button_rst_n, and sys_aux_pwer_det.
- ❖ Inputs from the IP core: link_req_rst_not, xmlh_ltssm_state, mac_powerdown, mac_clk_req_n, aux_pm_en, pm_pme_en, cfg_pm_no_soft_rst, mac_phy_rxecidle, and pm_dstate.
- ❖ Inputs from the application's internal logic within the chip: app_clk_req_n, app_ltssm_enable, and app_init_rst.
- ❖ Inputs from the PHY module: pclk, phy_status, and phy_clk_req_n.

The outputs are driven to three destinations:

- ❖ To the IP core: pwr_rst_n, core_clk, core_rst_n, sticky_rst_n, non_sticky_rst_n, and muxd_aux_clk.
- ❖ To the I/O pins of the chip: ref_clk_req_n and txclk.
- ❖ To the PHY module: phy_rst_n, phy_powerdown, and mac_phy_clk_req_n.
- ❖ To AXI bridge and AXI application logic: mstr_axi_clk, mstr_axi_resetn, slv_axi_clk, slv_axi_resetn, dbi_axi_clk, dbi_axi_resetn.

The Clock and Reset Control module generates all clock and reset signals required by the core IP. Reset control architecture is illustrated in [Figure C-14](#). Clock control architecture is illustrated in [Figure C-15](#).

Figure C-14 Reset Control

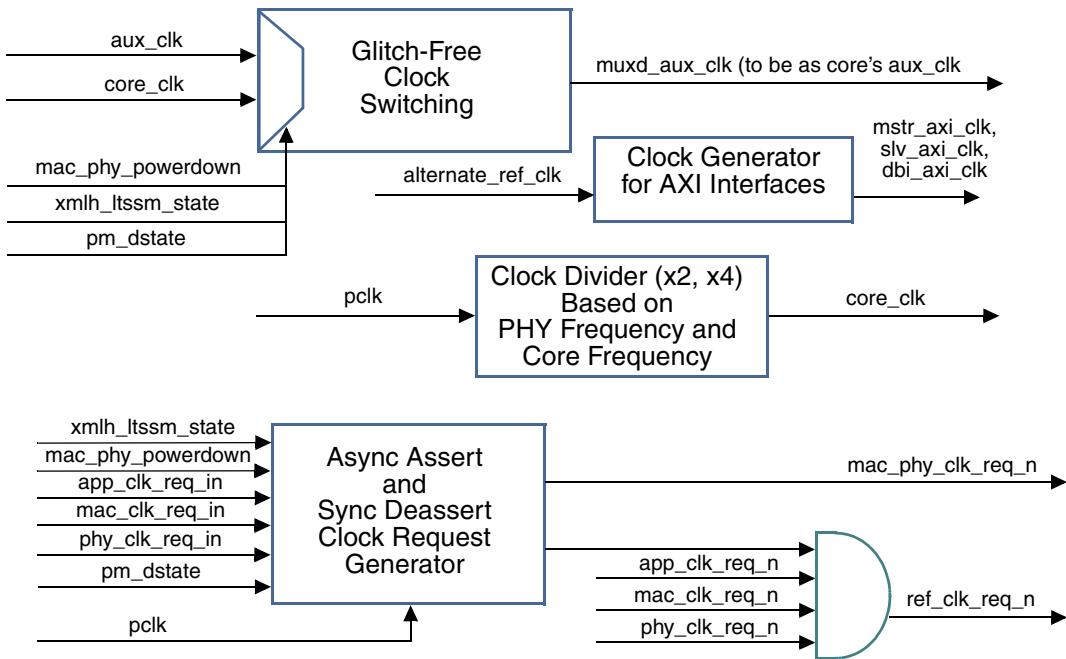


The PCIe core reset inputs can be asynchronously asserted, but must be synchronously deasserted with respect to the core_clk input. The reset implementation should be compliant with DFT methodology. It is recommended that the application logic and the core be reset simultaneously. There are two reset domains in the core logic. In general, core_rst_n resets the logic domain powered by main power; pwr_rst_n resets the logic domain powered by aux power. The core_rst_n signal should be asserted any time pwr_rst_n is asserted.

The sticky and non-sticky resets are routed to the CDM.

- ❖ sticky_rst_n: Resets registers marked as sticky registers in the PCI Express Base 2.0 Specification.
- ❖ non_sticky_rst_n: Resets the rest of the configuration registers not marked as sticky.

The phy_rst_n signal is used to reset the PIPE compliant PHY and is allowed to be completely asynchronous.

Figure C-15 Clock Control

- The Clock Control diagram shows the major features of an example clock design to support a typical PCIe application. The aux_clk input is designed to support a low-speed clock that may be used to minimize power when operating in a low-power mode. If the AXI clock is running in sync with the core clock, then all AXI clocks should be sourced from the core clock. Otherwise, the application should generate the AXI clock for AXI master and slave, based on an additional reference clock.
- The bulk of the core logic runs on the core_clk domain. The core_clk output is derived from the pclk output of the PHY. This clock may need to be divided (as shown) in cases where the core is running at a fraction of the pclk rate.
- A small amount of logic within the core needs a clock that is always available, even in low-power states. This logic is placed on the muxd_aux_clk domain. This clock is equivalent to core_clk during normal operation, but may be switched over to a low-speed auxiliary clock when core_clk is removed. This switching is performed with glitch-free clock switching logic in the reference design.
- The ref_clk_req_n output signal can be used to indicate to the system when a reference clock is required. By setting this signal to 1'b1, the device can indicate to the system that refclk may be removed. This can be used to further reduce power. The Synopsys PCIe PHY provides additional support for the ref_clk_req_n feature, and the example design includes this connectivity as well.
- Please refer to clk_RST.v for more details.

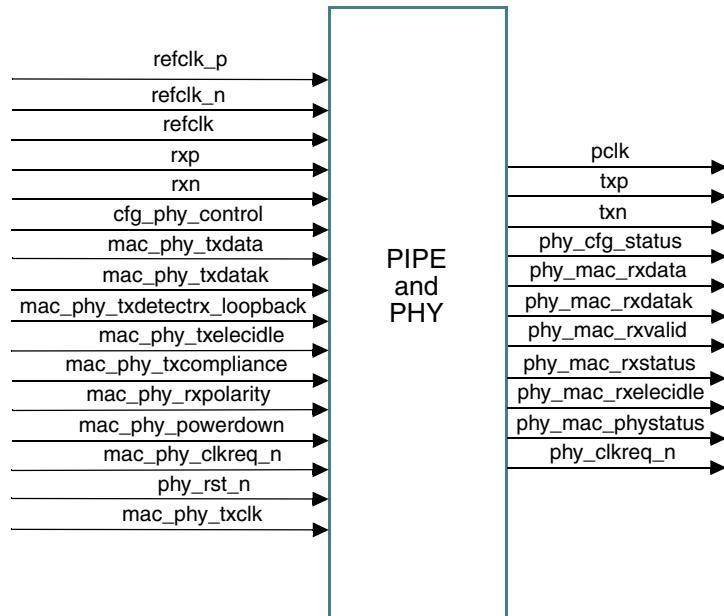
C.6.2.3 PIPE PHY Module

A standard PIPE PHY module for PHY implementation has been introduced. Both a generic PHY model for simulation-only purposes, and a Synopsys PHY model are available for your chip implementation. The PIPE PHY module is illustrated in [Figure C-16](#).

The example design instantiates this standard PIPE PHY module. To use the generic PHY or the Synopsys PHY, all that is required is selecting the appropriate PHY during the coreConsultant configuration. To select an alternate PHY, the intent is to make a wrapper (an example wrapper is provided) to map the desired PHYs I/O to the standard PHY interface. This should be primarily signal renaming if the PHY provides a

standard PIPE interface. A few additional parameters control optional features, such as selecting a differential clock. These can be determined from the example files provided. When configuring the core with the intent to use a custom PHY, select "Custom" from the PHY pull-down and follow the instructions provided in the <workspace>/src/Phy/custom/README.txt file to integrate your PHY model with the core.

Figure C-16 PIPE PHY Module



refclk_p and refclk_n:

Reference clock differential pair from device pins.

refclk:

Reference clock (non differential) from device pins. Only one of these reference clocks will typically be used by the PHY.

rxp and rxn:

Differential receive pair inputs from device pins.

txp and txn:

Differential transmit pair outputs to device pins.

cfg_phy_control:

Input to the PHY from the IP core to allow some external register control over the PHY internals. This is an optional interface and may be ignored.

phy_cfg_status:

Output from PHY to the core to indicate some of the internal status of the PHY. This is an optional interface and may be ignored.

mac_phy_clkreq_n:

Input to the PHY from the IP core and application logic to indicate that the MAC and application logic are ready to get the core clock removed. Supported by the Synopsys PHY.

phy_clkreq_n:

Output from the PHY to indicate that it is ready for the reference clock to be removed. Supported by the Synopsys PHY.

phy_RST_n:

Input from the clock reset control module to indicate that it desires to reset the PHY.

pclk:

Output from the PHY to supply the PIPE clock.

mac_phy_*:	Inputs to the PHY from the core. These inputs are standard PIPE signals. Please refer to the PIPE specification for information about these signals.
phy_mac_*:	Outputs from the PHY to the core. These outputs are standard PIPE signals. Please refer to the PIPE specification for information about these signals.
mac_phy_txclk:	This signal is present if your PHY is a PXPIPE PHY, and provides a source-synchronous transmit clock for data moving from the MAC to the PHY.

C.6.2.4 Master and Slave AXI Driver Modules

The Master and Slave AXI Driver modules (`axi_master_driver.v` and `axi_slave_driver.v`) are designed as a place holder for customers to connect an AXI master, an AXI slave, and optionally, an AXI DBI master to the DWC PCIe AXI bridge.

Applications may have an AXI interconnect module loaded into the reference design chip top to replace these modules. Or, an application may follow the hierarchy that the chip top offers to design master and slave drivers within these modules (`axi_master_driver.v` and `axi_slave_driver.v`)

The design for these modules should contain an AXI master, a DBI AXI master, and an AXI slave. These interfaces at the module are all standard AXI interfaces, as specified in the AMBA AXI specification.

These modules are empty aside from the module instantiation. It is fully expected that customers modify and replace these modules based on their applications.

C.6.2.5 Master Sideband AXI Driver Module

The Master Sideband AXI Driver module (`axi_master_sideband_driver.v`) drives the sideband signals needed to form a response to a master request issued from the DWC PCIe AXI bridge.

This reference design achieves two basic functions. One is to drive the response channel sideband signals such as `mstr_bmisc_info`, `mstr_rmsic_info`, and `mstr_resp_err_map`. The other is to take the master DWC PCIe AXI bridge ID signals and extend them into the application desired AXI ID signals.

The response channel sideband signals are controlled via registers. These registers allow the application to issue the requests with special PCIe header information. The `mstr_bmisc_info` signal is controlled by `mstr_bctrl_reg`. The `mstr_rmsic_info` signal is controlled by `mstr_rcctrl_reg`, and the `mstr_resp_err_map` signal is controlled by `mstr_ctrl_reg`. Additional information is provided in the ELBI Driver module section.

The reason for the extension of the ID is the assumption that the traffic class (TC of PCIe transaction), the PCIe requester function number, and the response for a non-posted request are all desired to be part of the extended ID, in order to identify the AXI responses. The function number and TC being part of an extended ID is only applicable to DWC PCIe IP core multiple function and multiple VC configurations, respectively. The 1-bit extended ID is used to identify that the response belongs to a non-posted request, and is an example to show how to drive the non-posted response bit of the sideband signals. This 1-bit extension only applies to customer configurations where the inbound and outbound translation modules are not loaded. Refer to the reference design `axi_master_sideband_driver.v` for detailed comments.

Note, that with a single function and single TC supported, and address translation performed where the AXI interface only issues memory RD/WR transfers, there will be no need to extend the ID. The chip-top has the capability to automatically detect the multiple function and TC enable modes.

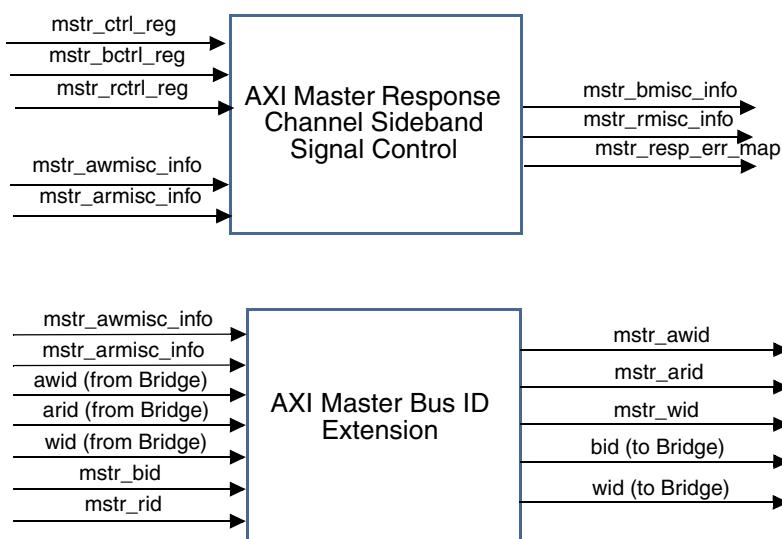
The outputs of this module are identified as follows:

- ❖ output [MSTR_RESP_MISC_INFO_WD -1:0] `mstr_bmisc_info`: AXI interface write response channel sideband signal output
- ❖ output [MSTR_RESP_MISC_INFO_WD -1:0] `mstr_rmsic_info`: AXI interface read response channel sideband signal output

- ❖ output [RESP_ERRBUS_WD -1:0] mstr_resp_err_map: Output to control the error response mapping
- ❖ output [BRIDGE_MSTR_BUS_ID_WD-1:0] bridge_mstr_rid: AXI Bridge output ID for read response channel
- ❖ output [BRIDGE_MSTR_BUS_ID_WD-1:0] ridge_mstr_bid: AXI Bridge output ID for write response channel
- ❖ output [MSTR_BUS_ID_WD-1:0] mstr_awid: Customer AXI interconnect ID for write command channel
- ❖ output [MSTR_BUS_ID_WD-1:0] mstr_arid: Customer AXI interconnect ID for read command channel
- ❖ output [MSTR_US_ID_WD-1:0] mstr_wid: Customer AXI interconnect ID for write data channel

AXI Master Sideband Control architecture is shown in [Figure C-17](#).

Figure C-17 AXI Master Sideband Control



C.6.2.6 Slave Sideband AXI Driver Module

The Slave Sideband AXI Driver module (`axi_slave_sideband_driver.v`) drives the sideband signals required for an AXI master portion of an application to drive the DWC PCIe AXI bridge.

This reference design achieves two basic functions. One is to drive sideband signals (such as `slv_awmisc_info`, `slv_armsic_info`, and `slv_resp_err_map`) required by the DWC PCIe bridge. The other is to extend the AXI ID bus based on multiple functions and multiple TCs.

The DWC PCIe bridge slave channel sideband signals can be controlled via registers. These registers enable the application to issue the requests with special PCIe header information. The `slv_awmisc_info` signal is controlled by `slv_wctrl_reg`, `slv_armisc_info` by `slv_rctrl_reg`, and `slv_resp_err_map` by `slv_ctrl_reg`.

The reason for the extension of the ID is the assumption that an AXI master can issue the traffic class (TC of PCIe transaction) and the function number onto the PCIe wire via its AXI ID. The need for function number and TC, being part of an extended ID, is only applicable to multiple function and multiple DWC PCIe IP core VC configurations, respectively. Please refer to the reference design `axi_slave_sideband_driver.v` for detailed comments.

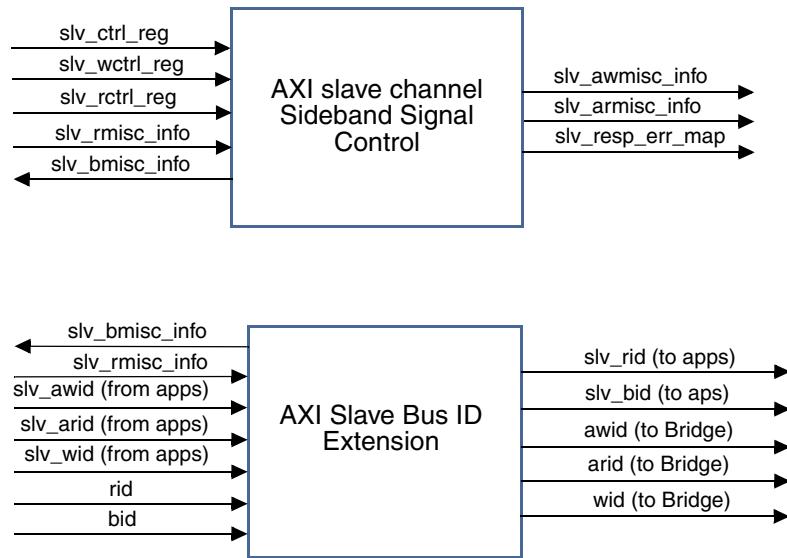
Note, that with a single function and a single TC supported, there will be no need to extend the AXI master ID signal with extended ID width. The chip top has the capability to automatically detect the multiple function and TC enable modes.

The outputs of this module are described as follows:

- ❖ output [SLV_MISC_INFO_WD -1:0] `slv_awmisc_info`: sideband signal of AXI slave write channel
- ❖ output [SLV_MISC_INFO_WD -1:0] `slv_armisc_info`: sideband signal of AXI slave read channel
- ❖ output [RESP_ERRBUS_WD -1:0] `slv_resp_err_map`: Error response map controlled by application
- ❖ output [BRIDGE_SLV_BUS_ID_WD -1:0] `bridge_slv_awid`: Write channel ID signal output to DWC AXI PCIe bridge
- ❖ output [BRIDGE_SLV_BUS_ID_WD -1:0] `bridge_slv_arid`: Read channel ID signal
- ❖ output [BRIDGE_SLV_BUS_ID_WD -1:0] `bridge_slv_wid`: Write data channel ID signal
- ❖ output [SLV_BUS_ID_WD -1:0] `slv_rid`: Read response channel ID signal
- ❖ output [SLV_BUS_ID_WD -1:0] `slv_bid`: Write response channel ID signal

Slave Sideband AXI Driver architecture is shown in [Figure C-18](#).

Figure C-18 AXI Slave Sideband Control



C.6.2.7 Inbound Address Translation Driver Module

The Inbound Address Translation Driver module translates an inbound PCIe transaction onto an AXI master memory transfer.

The reference design can perform the following address translation:

1. An inbound memory RD/WR that falls into BAR0 of a configured range will be translated as an AXI memory RD/WR transfer. There are two AMBA memory regions defined in this module. A PCIe memory BAR0 region can be mapped into two (or more) AMBA memory regions if an application has this requirement.
2. An inbound IO RD/WR that falls in BAR1 of a configured PCIe range will be translated as an AXI memory RD/WR transfer in a defined AMBA memory region.
3. An inbound vendor message request will be translated as an AXI memory write transfer in a defined AMBA memory region.
4. An inbound memory RD/WR that falls in an extended ROM BAR will be translated as an AXI memory RD/WR transfer at a defined AMBA memory region.

There are a total of five AMBA memory regions defined in this module. These are identified as follows:

Starting Address

- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim0_mem_addr_start: First internal memory region (where BAR0 of PCIe transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim1_mem_addr_start: Second internal memory region (where BAR0 of PCIe transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_io_addr_start: Internal IO region (where BAR0 of PCIe IO transaction will map to)
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_msg_addr_start: Internal message region
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_rom_addr_start: Internal ROM region (where ROM BAR of PCIe transaction will map to)

Address Limit

- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim0_mem_addr_limit: First internal memory region
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim1_mem_addr_limit: Second internal memory region
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_io_addr_limit: Internal IO region
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_msg_addr_limit: Internal message region
- ❖ input [`MASTER_BUS_ADDR_WIDTH -1:0] pim_rom_addr_limit: Internal ROM region

Note that all of these regions are controlled via the registers defined in the ELBI driver module. Please refer to "[ELBI Driver Module](#)" for more information.

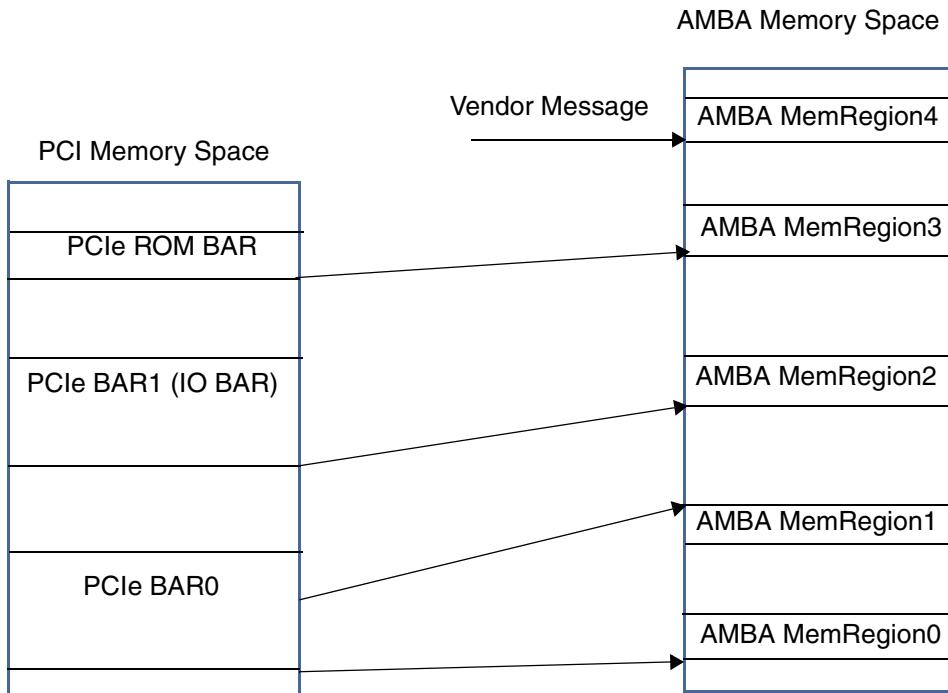
The outputs of this module are described as follows:

- ❖ output [63:0] rtranslated_addr_in_d: The translated address based on the regions defined above
- ❖ output [1:0] rtranslated_err_in_d: Error indicates the status of the translation. 00 -- Normal, 01 -- UR, 10 -- CA. 11-- Undefined (not allowed)

When an address matches a BAR, but fails to match an AMBA memory region, an error response will be sent to the PCIe core as a response.

Inbound Address Translation Driver architecture is shown in [Figure C-19](#).

Figure C-19 Inbound Address Translation



C.6.2.8 Outbound Address Translation Driver Module

The Outbound Address Translation Driver module translates an AXI slave memory transfer into a PCIe transaction. The reference design can perform the following address translation:

1. Two AMBA memory regions are translated onto two different PCIe memory regions. This demonstrates a multiple-function device with two independent functions requiring two memory regions.
2. One AMBA memory region is defined to support an outbound AXI transfer translated into an outbound PCIe IO transaction.
3. Two AMBA memory regions are defined to support an outbound AXI transfer translated into an outbound PCIe configuration transaction. Each of these AMBA memory regions can take care of the PCIe configuration type0 and type1 translation.
4. One AMBA memory region is defined to support an outbound AXI transfer translated into an outbound PCIe message transaction.

There are total of six AMBA memory regions defined in this module. These are identified as follows:

Starting Address

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] pom0_mem_addr_start: PCIe out memory space (where the address of the internal request is in region0)
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] pom1_mem_addr_start: starting address of the PCIe out memory space (where the address of the internal request is in region1)
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in0_mem_addr_start: Internal memory region 0

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in1_mem_addr_start: Internal memory region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_io_addr_start: Internal IO region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg0_addr_start: Internal CFG0 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg1_addr_start: Internal CFG1 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_msg_addr_start: Internal MSG region

Address Limit

- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in0_mem_addr_limit: Internal memory region 0
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in1_mem_addr_limit: Internal memory region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_io_addr_limit: Internal IO region 1
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg0_addr_limit: Internal CFG0 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_cfg1_addr_limit: Internal CFG1 region
- ❖ input [`SLAVE_BUS_ADDR_WIDTH -1:0] in_msg_addr_limit: Internal MSG region

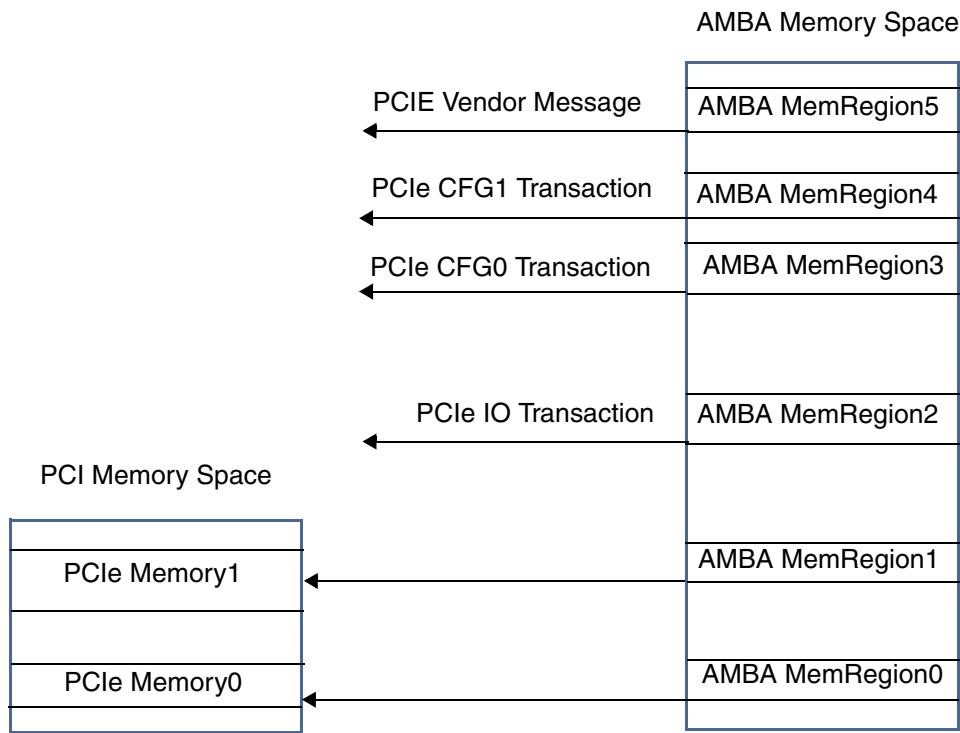
Note that all of these regions are controlled via the registers defined in the ELBI driver module. Please refer to “[ELBI Driver Module](#)” for more information.

The outputs of this module are identified as follows:

- ❖ output [63:0] xtranslated_addr_in_d: The translated address of a PCIe transaction
- ❖ output [4:0] xtranslated_type_in_d: The translated type of a PCIe transaction
- ❖ output [7:0] xtranslated_func_in_d: The translated function number of a PCIe transaction
- ❖ output [7:0] xtranslated_msgcode_in_d: The translated message code of a PCIe transaction
- ❖ output xtranslate_enable: The enable to the translation. When no matching is detected, the enable is off.

Inbound Address Translation Driver architecture is shown in [Figure C-20](#).

Figure C-20 Outbound Address Translation



C.6.2.9 ELBI Driver Module

The ELBI Driver module accesses all registers that required to support the reference design. It can be modified by the customer to support application-specific registers (such modifications are beyond the scope of this document).

All registers within this module can be accessed by an application AXI master via the slave DBI interface of the AXI bridge (by a common slave interface, if this option is utilized).

Registers designed to support the reference design are identified as follows:

Table C-13 Inbound AMBA memory region0 start address: Address Offset 0 of DBI transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region0 for inbound address translation

Table C-14 Inbound AMBA memory region0 limit: Address Offset 0x08 of DBI transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region0 for inbound address translation

Table C-15 Inbound AMBA memory region1 start address: Address Offset 0x10 of DBI transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region1 for inbound address translation

Table C-16 Inbound AMBA memory region1 limit: Address Offset 0x18 of DBI transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region1 for inbound address translation

Table C-17 Inbound AMBA memory region2 start address: Address Offset 0x20 of DBI transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region2 for inbound address translation of IO transaction

Table C-18 Inbound AMBA memory region2 limit: Address Offset 0x28 of DBI transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region2 for inbound address translation of IO transaction

Table C-19 Inbound AMBA memory region3 start address: Address Offset 0x030 of DBI transfer

Bits	Default	Descriptions
63:0	0	The starting address of an AMBA memory region3 for inbound address translation of PCIe Expansion ROM address space

Table C-20 Inbound AMBA memory region3 limit: Address Offset 0x038 of DBI transfer

Bits	Default	Descriptions
63:0	0	The limit of an AMBA memory region3 for inbound address translation of PCIe Expansion ROM address space

Table C-21 Starting address of Outbound PCIe memory transaction for AMBA memory region 0: Address Offset 0x50 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of PCIe memory RD/WR transaction translated from an AMBA outbound memory region0

Table C-22 Starting address of Outbound PCIe memory transaction for AMBA memory region1: Address Offset 0x60 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of PCIe memory RD/WR transaction translated from an AMBA outbound memory region1

Table C-23 Outbound AMBA memory region0 start address: Address Offset 0x70 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region0

Table C-24 Outbound AMBA memory region0 limit: Address Offset 0x78 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region0

Table C-25 Outbound AMBA memory region1 start address: Address Offset 0x80 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region1

Table C-26 Outbound AMBA memory region1 limit: Address Offset 0x88 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region1

Table C-27 Outbound AMBA memory region2 start address: Address Offset 0x90 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region2 for the translation of outbound IO transfer

Table C-28 Outbound AMBA memory region2 limit: Address Offset 0x98 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region2 for the translation of outbound IO transfer

Table C-29 Outbound AMBA memory region3 start address: Address Offset 0xA0 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region3 for the translation of outbound CFG0 transfer

Table C-30 Outbound AMBA memory region3 limit: Address Offset 0xA8 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region3 for the translation of outbound CFG0 transfer

Table C-31 Outbound AMBA memory region4 start address: Address Offset 0xB0 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region4 for the translation of outbound CFG1 transfer

Table C-32 Outbound AMBA memory region4 limit: Address Offset 0xB8 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region4 for the translation of outbound CFG1 transfer

Table C-33 Outbound AMBA memory region5 start address: Address Offset 0xC0 of DBI transfer

Bits	Default	Descriptions
63:0	1	The starting address of an AMBA outbound memory region5 for the translation of outbound message transfer

Table C-34 Outbound AMBA memory region5 limit: Address Offset 0xC8 of DBI transfer

Bits	Default	Descriptions
63:0	1	The limit of an AMBA outbound memory region5 for the translation of outbound message transfer

Table C-35 Master Control Register: Address Offset 0xD0 of DBI transfer

Bits	Default	Descriptions
31:0	0	<p>The register is designed to control the error mapping sideband signals of the master AXI interface. Only three of these bits are valid. Others are reserved.</p> <ul style="list-style-type: none"> Bit0: 1'b0 indicates that UR maps to response error, 1'b1 set for decode error Bit1: 1'b0 indicates that CA maps to response error, 1'b1 set for decode error Bit2: 1'b0 indicates that CRS maps to response error, 1'b1 set for decode error

Table C-36 Master Write Response Channel Control Register: address Offset 0xD4 of DBI transfer

Bits	Default	Descriptions
1:0	0	This 2-bit field controls the attribute field of the completion header of a non-posted write
2	0	This bit requests a PCIe transaction nullification for a completion of a non-posted write.
5:3	0	This 3-bit field controls the TC field of the completion header of a non-posted write
6	0	This bit controls the BCM bit of the completion header of a non-posted write.
7	0	This bit controls the EP bit (poison bit) of the completion header of a non-posted write
8	0	This bit controls the TD bit (ECRC bit) of the completion header of a non-posted write
11:9	0	This 3-bit field controls the completer function number of the completion header of a non-posted write.
12	0	This bit indicates that this transfer is a response to a non-posted request

Table C-37 Master Read Response Channel Control Register: address Offset 0xD8 of DBI transfer

Bits	Default	Descriptions
1:0	0	This 2-bit field controls the attribute field of the completion header of a non-posted read
2	0	This bit requests a PCIe transaction nullification for a completion of a non-posted read.
5:3	0	This 3-bit field controls the TC field of the completion header of a non-posted read
6	0	This bit controls the BCM bit of the completion header of a non-posted write.
7	0	This bit controls the EP bit (poison bit) of the completion header of a non-posted read
8	0	This bit controls the TD bit (ECRC bit) of the completion header of a non-posted read
11:9	0	This 3-bit field controls the completer function number of the completion header of a non-posted read.
12	0	This bit indicates that this transfer is a response to a non-posted request

Note that the following tables contain the sideband control registers. These registers control AXI transfers with a particular header information programmed into these registers. If address translation is enabled by a customer using our inbound and outbound reference designs, the controls from these registers for type field of the header are not valid.

Table C-38 Slave Control Register: Address Offset 0xE0 of DBI transfer

Bits	Default	Descriptions
31:0	0	<p>This register is designed to control the error mapping sideband signals of the master AXI interface. Only three of these bits are valid. Others are reserved.</p> <ul style="list-style-type: none"> Bit0: 1'b0 sets for response error maps to UR, 1'b1 set for decode error Bit1: 1'b0 set for response error maps to CA1'b1 set for decode error Bit2: 1'b0 set for response error maps to CRS, 1'b1 set for decode error

Table C-39 Slave Write Command Channel Control Register: address Offset 0xE4 of DBI transfer

Bits	Default	Descriptions
4:0	0	This 5-bit field controls the type field of the PCIe write request header
5	0	This bit controls the BCM field of the PCIe write request header
6	0	This bit controls the EP field of the PCIe write request header
7	0	This bit controls the TD field of the PCIe write request header
9:8	0	This 2-bit field controls the attribute field of the PCIe write request header
12:10	0	This 3-bit field controls the TC field of the PCIe write request header
20:13	0	This 8-bit field controls the message code field of the PCIe write request header
21	0	<p>This bit is only valid when shared DBI mode of AXI bridge is configured. It indicates whether or not the current AXI transfer is a DBI transfer or an outbound PCIe transfer.</p> <p>Set this bit to 1 to indicate a DBI access.</p>

Table C-40 Slave Read Command Channel Control Register: address Offset 0xE8 of DBI transfer

Bits	Default	Descriptions
4:0	0	This 5-bit field controls the type field of the PCIe read request header
5	0	This bit controls the BCM field of the PCIe read request header
6	0	This bit controls the EP field of the PCIe read request header
7	0	This bit controls the TD field of the PCIe read request header
9:8	0	This 2-bit field controls the attribute field of the PCIe read request header
12:10	0	This 3-bit field controls the TC field of the PCIe read request header
20:13	0	This 8-bit field controls the message code field of the PCIe read request header

Table C-40 Slave Read Command Channel Control Register: address Offset 0xE8 of DBI transfer

Bits	Default	Descriptions
21	0	This bit is only valid when shared DBI mode of AXI bridge is configured. It indicates whether or not the current AXI transfer is a DBI transfer or an outbound PCIe transfer. Set this bit to 1 to indicate a DBI access.

Table C-41 Global Translation Control Register: address Offset 0xFC of DBI transfer

Bits	Default	Descriptions
0	0	This bit enables the translation that is designed for inbound and outbound traffic. 1'b1 indicates that all inbound and outbound memory translations for address and type designed in this reference example are enabled.

C.6.2.10 Legacy Interrupt and MSI Interface Driver Module

The DWC PCIe core supports three interrupt mechanisms. These are mutually exclusive, and only one of these interrupt mechanisms should be enabled per-function at any given time.

This module may be used for legacy interrupt reference design and for MSI interrupt reference design.

There are two sources of interrupt requests supported in this example:

- ❖ From the application logic when a status is done, or other reasons for generation of an interrupt.
- ❖ From an external button being pushed.

You can add or remove the interrupt sources to/from this reference design.

When one of these sources is asserted, the reference design will either generate a legacy interrupt request on sys_int, or the MSI request to the core's MSI interface.

The input interrupt source signals to this modules are:

- ❖ input [NF-1:0] attention_button: A request to PCIe module to generate interrupt based on external button pushed.
- ❖ input [NF-1:0] app_intr_req: A request from application logic on chip to request PCIe module to generate interrupt
- ❖ input [NF-1:0] app_intr_clr: A clear interrupt from application logic on chip to indicate that the interrupt has been served, and application is ready to clear the interrupt. Normally, the application gets the clear status from a driver status register

These signals are wired at the chip-top. The customer is required to review them and provide the required driving source behind them. The application will need to evaluate their interrupt strategy and get the source of an interrupt identify.

Please refer to interrupt_msi_driver.v for more information about driving the MSI interface and legacy interrupt sys_int.

C.6.2.11 Power Management Control Interface Driver Module

The Power Management Control Interface Driver module may be used by an application that has special requirements for the lower power states, beyond the typical power management functionalities designed into the IP core. Most power management controls are handled inside the core. There are some controls provided to the application that can improve or support further power management functionality other than those specified by the PCIe spec. This module contains the descriptions for these special power management functionalities.

There are four outputs generated by this reference module:

- ❖ app_req_entr_l1
- ❖ app_req_exit_l1
- ❖ app_ready_entr_l23
- ❖ apps_pm_xmt_pme

app_req_entr_l1 controls entering the lower power L1 state. This is for an ASPM-enabled device only. The core has a timeout mechanism implemented to automatically enter L1. The PCI Express Base 2.0 Specification allows for optional L1 entry schemes (for instance, a way of detecting the packet finished from all request sources of the application logic), using this signal to accelerate a request to enter L1.

app_req_exit_l1: Exits L1 when a packet is pending for a transmission. Applications are expected to make a decision whether or not to remove the reference clock during L1. This decision should be affected by the clock lock and recovery latency of the PHY. Please refer to the clock and reset example for additional details about reference clock removal. If the clock will be removed during L1, the application is expected to issue an app_clk_req_n wake up signal (note that there is no core clock or application clock before waking up). After the clock is restored, the application may assert app_req_l1_exit or simply present a PM PME event transmit request or any other packet transmit request to transition the core from the L1 to the L0 state.

app_ready_entr_l23: When a device has been programmed into the D3 state and a turnoff handshake has been performed between PCIe devices, the device is expected to enter the L23 state and the main clock and power will go off. If the application has any other reason that requires the presence of the main clock and power, it should deassert this ready signal so that the core will delay entering L23 (the application cannot delay it indefinitely without violating the PM protocol).

apps_pm_xmt_pme: The application is responsible for generating the conditions for a power management event. This is typically tied to a wake-up event. When the PCIe device is in lower power, such as the L1, L2/L3 state, this signal can be used to help wake up the device. An event detected by the application, or device addition/removal are the typical sources of the wake up. When a wake-up is decided upon, it is possible that the core is in a no-clock state. In that case, a wake-up sequence must follow after the application asserts the apps_pm_xmt_pme signal.

When the core is in the L1 state, the application should perform two steps:

1. Assert app_clk_req_n.
2. When the clock is up, then assert apps_pm_xmt_pme for one cycle.

When the core is in the L2/L3 state, the application should:

1. Assert apps_pm_xmt_pme. This causes the core to start beacon while aux clock and aux power is supplied.
2. Also assert app_clk_req_n. and then wait for the PHY clock to be restored. A PM PME message will be transmitted from the core to notify the PME event.

C.6.3 Considerations During Integration of the DWC PCIe AXI IP

A chip_top.v is present in the *src/customer/generic/* directory. We strongly recommended that you review the top-level RTL code and understand all of the modules provided as a reference design.

Here is a basic summary of the modules to that you should pay special attention to:

- ❖ Clock and Reset module: Most of the reference design modules are synthesizable. The Clock And Reset module is not. It requires additional work on the part of the chip designer to perform the necessary clock generation, and to define the clock and reset trees required by the IP, and by the rest of the chip. The clock and reset module is a place holder for the customer. This module is heavily commented so that customers may find where the clock generation logic should be placed.
- ❖ The AXI master, slave and DBI standard interface drivers also require additional customer attention, as these modules are where the other application AMBA IP blocks or interface logic to the DWC PCIe AXI bridge should be connected.
- ❖ Interrupt source generation is required from the application logic. Please refer to the above interrupt section for details on the interrupt source signals.
- ❖ PCIe Power management is handled within the PCIe core unless there is a special requirement from the application. Please refer to the Power Management module ([Section C.6.2.11](#)) for information to control the power states with respect to special requirements.

Most of the other modules should have connections made correctly to the interfaces that they are intended to service, and only the internal logic should require modification. These modifications may also need additional inputs and outputs to be added to the reference designs, but the designs are intended to represent a "complete" interconnection.

C.6.4 Configuration Parameters

C.6.4.1 Queue Depths

In general, when selecting the queue depths, the major trade-off to be made is between area and performance. Larger queue sizes will have a larger area impact but may also reduce the occurrence of back pressure, depending on system settings. In addition, synthesis QOR may be negatively impacted by selecting larger queue sizes.

C.6.4.1.1 Master Request Header Queue Depth

This queue depth, set via coreConsultant, specifies the number of master requests that the application requires the AXI bridge to enqueue in the inbound direction. There is a receive queue in the native PCIe core for all inbound requests therefore it is not necessary to have a large queue depth for this master request header queue. This queue depth is set to a default of 4 (recommended value) which means that there can be a total of 4 outstanding inbound read or write requests enqueued in the AXI bridge.

There are pipe stages within the AXI adapter and internal generic interface adapter modules. This queue depth is used to minimize the start/stop pipe delay within the AXI bridge. Increasing this queue depth (during configuration) can improve the overall performance of the AXI interconnect based on the pipe stages within the AXI interconnect. The disadvantage of increasing this depth is an increase in gate count and RAM size. Under normal operation, the default value should be adequate.

C.6.4.1.2 Master Request Data Queue Depth

This queue depth is sized based upon the header queue depth discussed in the previous section. This queue depth is also dependent upon the native PCIe core's data bus width. For example, when the data bus is set to 32 bits, and assuming 1 inbound write for every 4 inbound requests, and the MTU (max transfer size) is 128 bytes, the calculated default depth is $MTU/4 + 2 = 34$.

Increasing this queue buffer depth parameter can improve the overall performance of the AXI interface based on the pipe stages within the AXI interconnect. The disadvantage of increasing this depth is larger gate counts and RAM sizes. It is important for the application to have some understanding of the patterns of inbound requests, such as the average number of reads and writes, how they are interleaved, and typical write request sizes.

The suggested default value should be adequate under normal operation. The default value is set to store one CC_MAX_MSTR_MTU worth of data +2.

C.6.4.1.3 Slave Request Header Queue Depth

This queue depth specifies the number of outbound requests that can be enqueued in the AXI bridge. Since there is no transmit queue in native PCIe core, this queue depth can impact the performance of the overall outbound transitions depended upon the Posted and Non-Posted credit availability of the remote PCIe device. If the device is balanced with the AXI interface bandwidth and native PCIe core's bandwidth, and the credits are available from the PCIe link, then this queue is not required to be very large. Otherwise, increasing this queue buffer depth parameter can improve the overall performance. In some applications however, it can cause more blocking issues in a credit-limited system since both Posted and Non-Posted requests are enqueued into this same buffer.

The suggested default value should be adequate under normal operation. The default value is set to CC_MAX_SLV_TAG.

C.6.4.1.4 Slave Request Data Queue Depth

This data queue depth is related to the header queue depth discussed in the previous section. It is the data queue depth based on headers in the header queue.

This depth is dependent on the number of read or write requests that are enqueued and the maximum transfer size of write requests.

The suggested default value should be adequate under normal operation. The default value is set to store one CC_MAX_SLV_MTU worth of data +2.

C.6.4.2 Maximum Master and Slave Tags

The consideration when setting the parameter for maximum master and/or slave tags, is trading off area versus the maximum number of queued requests allowed. Typically, this is determined by the system software programming model. The default value should work well for most systems, but we recommend that you consult your software provider.

C.6.4.3 Remote Device Maximum Read Request Size

This configuration parameter sets the size of the buffer inside the AXI bridge composer module. It should be set to a size large enough to support the maximum read transfer size expected by the PCIe remote device for outbound read transfers. If this configuration parameter is not set large enough, it will get a partial completion returned to the AXI read channel and in general will not function properly. If it is set larger than necessary, gate area will be wasted.

C.6.4.4 Master Page Boundary Size

This parameter specifies an address page boundary of either 128, 256, 512, 1K, 2K or 4K. The default value is 4K. No inbound or outbound transactions can have an address that crosses this boundary. Typically, 4K would be the recommended setting for an outbound transfer. 1K would be the recommended setting for an inbound transfer. The AXI bridge will ensure that the inbound request will not cross the selected page boundary when driven onto the AXI master interface.

C.6.4.5 Address and Data Widths

The parameter settings for address and data width are usually based on host system interconnect requirements. Internal data conversions match AXI data bus widths with internal datapath sizes (determined by lanes and symbols per lane). Wider data bus width for a given clock frequency supports greater bandwidth on the AXI bus but may not result in overall system bandwidth improvement. This is because if the AXI BW is not balanced against the PCIe link bandwidth in queues filling up faster and hence result in more back pressure in the inbound or outbound directions.

C.6.5 RAMs

This section describes the RAMs that may be used in the AXI Bridge, dependent upon configuration settings in coreConsultant. Some of RAMs may be required for one application, but not others. All RAM options are described in the following sections.

There are two sets of RAMs. One is for the PCIe native core, the other for the AXI bridge. The descriptions for the AXI bridge RAMs are as follows.

RAM size can be printed from coreConsultant after an AXI bridge is configured. Most RAMs are sized based on the application's implied requirements. Few RAMs are sized directly by the application via coreConsultant.

C.6.5.1 XADMX0 Decomposer Header and Data RAMs

XADMX0 decomposer header and data RAMs are used to implement FIFOs to buffer AXI bridge inbound responses when the maximum PCIe inbound transfer and AXI transfer sizes may be different. Since the header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AXI master interface clock is running at the same frequency as the base core clock. The mstr_axi_clk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.2 XADMX1 Decomposer Header and DATA RAMs

XADMX1 decomposer header and data RAMs are used to implement FIFOs to buffer AXI bridge outbound responses when the maximum PCIe outbound transfer and AXI bus burst sizes may be different. Since the header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AXI master interface clock is running at the same frequency as the base core clock. The mstr_axi_clk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.3 Slave TAG Asynchronous FIFO RAM

Slave TAG asynchronous FIFO RAM is used to cross a TAG management interface clock domain between the outbound decomposer and outbound composer. It has two clock inputs that can be tied together if the AXI slave interface clock is running at the same frequency as the core clock. The slv_axi_clk is the slave interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.4 Slave TAG Manager RAM

Slave TAG manager RAM is designed as an asynchronous FIFO for outbound transfer TAG control/recycle. It has two clock inputs that can be tied together if the AXI slave interface clock is running at the same frequency as the core clock. The slv_axi_clk is the slave interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.5 RADMX Asynchronous RAM

RADMX asynchronous RAM is used to cross a clock domain between the outbound composer and the AXI response channel. It has two clock inputs which can be tied together if the AXI slave interface clock running at the same frequency as core clock. The slv_axi_clk is the slave interface clock. The parameters used in the

instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.6 RADMX Decomposer Header and Data RAMs

RADMX decomposer header and data RAMs are used to implement FIFOs to buffer AXI bridge inbound requests when the PCIe transfer size may be different from the AXI bus burst size. Since header and data are separate, two RAMs are required to implement the FIFOs. Each RAM has two clock inputs that can be tied together if the AXI master interface clock is running at the same frequency as the base core clock. The mstr_axi_clk is the master interface clock.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.7 Master TAG Manager RAM

Master TAG Manager RAM is designed as an asynchronous FIFO for inbound transaction TAG control/recycle. It has two clock inputs that can be tied together if the AXI slave interface clock is running at the same frequency as core clock. The mstr_axi_clk is the master interface clock. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.8 GM Composition RAM

GM Composition RAM is designed as a segmented-buffer queue for inbound response composition. It runs under the master AXI clock domain. The parameters used in the instantiation of this RAM define its depth, width, and pointer width. They are located in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.9 RADMX Composition RAM

RADMX composition RAM is designed as a segmented-buffer queue for outbound response composition. It is running under the PCIe base core clock domain.

The parameters used in the instantiation of this RAM define its depth, width, and pointer width. The values of these parameters can be found in the AXI bridge's CC constant file. Please refer to this file for detailed sizing information.

C.6.5.10 RADMX Asynchronous TAG FIFO RAM

This RAM is designed for an AXI configuration where AXI clock is different from core clock. TAG information will need to be synchronized when crossing the clock domain.

C.6.6 The Suggested Integration Method for Application Logic

Refer to the “Building and Verifying Your Core” chapter in the applicable DesignWare core’s user manual for instructions to configure, synthesize, and simulate the core. This section gives you an idea of how to build a chip with the IP core instantiated.

The recommended application integration flow is as follows:

Step 1. Read the implementation guidelines presented in this addendum very carefully.

Step 2. Build a core with your desired features.

Step 3. Review the structured chip level reference design (i.e., chip_top.v, clk_rst.v, etc.)

Step 4. Integrate the application logic into the chip_top.v.

The main considerations for Steps 1 through 3 are:

1. Decide on the number of client interfaces and the type of traffic going through them. There are template (.v) files to help you to integrate the application logic required to drive the client interfaces. The client interfaces are where the application logic can send traffic to the PCIe link.
2. Decide on how you want received inbound traffic handled. Two interfaces (ELBI and Target1) are available within the IP core. Two template (.v) files are available to integrate the required application logic.
3. You should review the clk_rst.v RTL code carefully and read all comments in the RTL file. There is logic (such as clock switching circuitry, etc.) in this file that you will need to replace.
4. You should review the strategy for interrupt implementation in the application and implement the respective interrupt source logic. Signals app_intr_req and app_intr_clr (at chip_top.v) should be driven by customer logic. You should also review the interrupt reference design module.
5. There are a few signals that you will use to control the IP core’s power state transitions. You should provide the driving source logic or tie these signals. There is a power management template (.v) in the reference design that you should review as well.

Complete Step 3 for other miscellaneous functionalities.

C.6.7 VTB Test Bench Integration Method (EP Core only)

Not available for AXI, only AHB.

C.6.8 How to Proceed

Once you have created and configured the RTL core, you can start adding your application interface code. Refer to [Section C.6, “Implementation Guidelines.”](#) This chapter describes the different application interface blocks and what you need to do to start adding your own application code.

D

App Note: FGPA Synthesis for PCle Cores

This application note provides an example flow for implementing a Synopsys PCI Express Core in an FPGA. The example flow is based on the flow that Synopsys has used for its PCI Express Cores demonstration and validation platform.

D.1 Introduction

[Figure D-1](#) shows a block diagram of the FPGA used in the demonstration board that Synopsys uses for its PCI Express Cores. The FPGA demonstration board functions as a PCI Express based Ethernet add-in card, using the DesignWare EP Core (PCI Express Endpoint) as the port logic for the PCI Express interface.

The PIPE interface from the EP Core is double registered and can use either of the following PHY implementations:

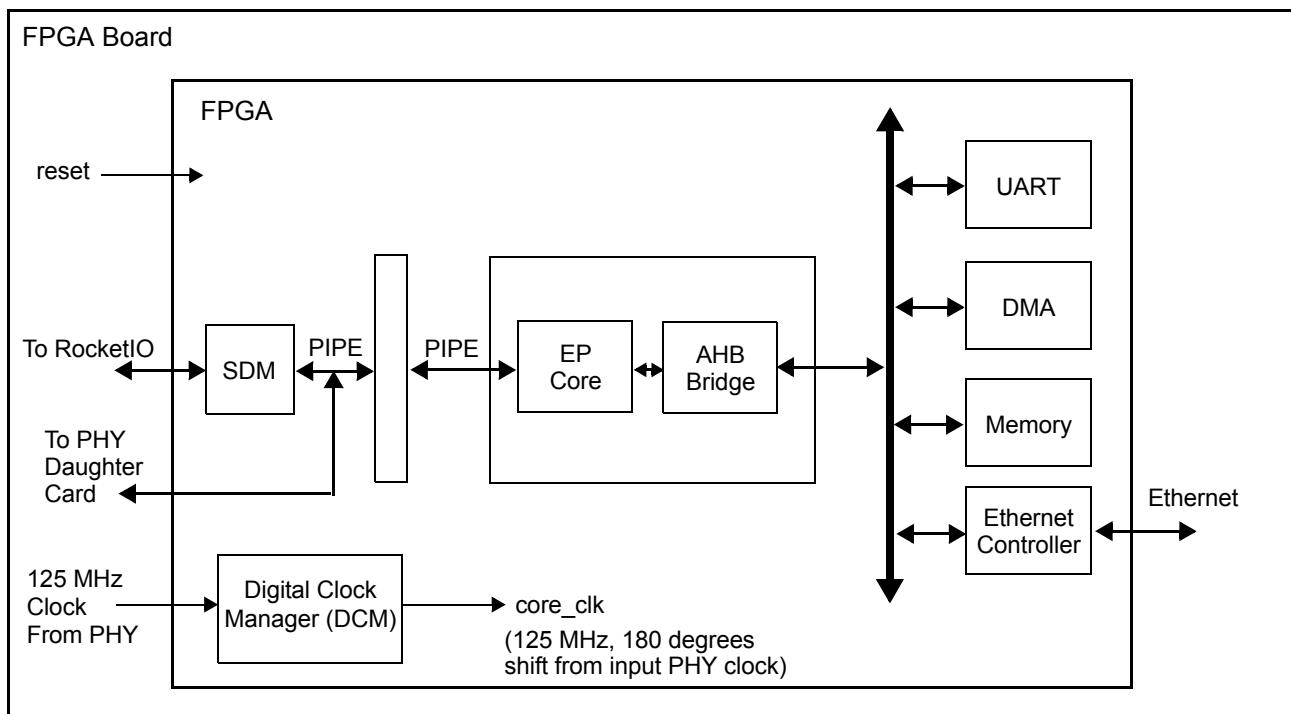
- ❖ Connection to Xilinx RocketIO transceiver through a SerDes Dependent Module (SDM). In this case, the reset input is directly from the motherboard.
- ❖ Connection directly to a PIPE-compliant PHY on an attached PHY daughter card, like Synopsys has done on the demonstration board. In this case, the reset signal from the motherboard goes to the PHY daughter card and from the PHY daughter card to the FPGA.



Example code for the SDM can be provided upon request.

The EP Core configuration used in the FPGA demonstration board is:

- ❖ Number of Lanes: x4
- ❖ Symbols per clock: 2
- ❖ Frequency: 125 MHz
- ❖ Datapath width to application: 64 bits

Figure D-1 Block Diagram of Synopsys PCI Express Demonstration FPGA

D.2 Requirements for FPGA Implementation

In order to improve timing performance the following items are required:

- ❖ FPGA Device:
 - ◆ Please see [Table D-2](#) for a summary of supported FPGA vendors, device families, and speedgrades.
- ❖ DesignWare PCI Express Core configuration
 - ◆ Requirements:
 - ✧ FPGA = 1
 - ✧ CX TECHNOLOGY = 0
 - ✧ CX_RAM_AT_TOP_IF = 0
 - ✧ CX_RAM_TYPE = 2 (FPGA RAM).

If you are using the SR-IOV feature (EP only):

- ❖ The maximum number (total of all VFs in all PFs) of Virtual Functions (VFs) supported is reduced from 256 to 16.

If you are using the Xilinx RocketIO transceiver:

- ❖ PHY_TYPE = 1 (Xilinx V2P) or 2 (Xilinx V4).
- ❖ The following limitations exist:
 - ◆ The PCI Express beacon signal is not supported.
 - ◆ Power management is not supported.
 - ◆ Tx Electrical Idle is not supported.



Note When using RocketIO, please request the Example SDM code which is the glue logic between the Xilinx provided Rocket I/O and the core's PIPE interface. This SDM block is provided as example code and does not include support from Synopsys.

D.3 Xilinx FPGA Flow

Synopsys provides Xilinx-specific scripts and constraint files for use with Xilinx-provided synthesis and place and route tools as examples. You may need to modify them to meet the needs of your design.

The following example targets a Xilinx Virtex2-Pro component, using XST to synthesize the netlist. The “-xst on” option causes xst_build.pl to invoke Xilinx XST for synthesis before running the Xilinx backend tools:

```
% <workspace>/auxiliary/xst_build.pl -product <ep|dm|rc|sw> -config  
<workspace> -xst on -device xc2vp70-7-ff1517
```

D.4 Altera FPGA Flow

Synopsys provides Altera-specific scripts and constraint files for use with Altera-provided synthesis and place and route tools as examples. You may need to modify them to meet the needs of your design.

The following example targets an Altera Stratix II GX component, using Quartus to synthesize the netlist. The “-quartus on” option causes quartus_build.pl to utilize Quartus for synthesis before running the Quartus backend tools:

```
% <workspace>/auxiliary/quartus_build.pl -p <ep|dm|rc|sw> -config <workspace> -quartus  
on -device EP2SGX30DF780C3
```

When targeting the Stratix II GX, Quartus 6.0 requires the TalkBack feature of the synthesis engine to be turned on:

1. Create a .altera.quartus directory in your home directory.
2. Create a file quartus2.ini that contains the following:

[Talkback 6.0]

TALKBACK_ENABLED = on

TALKBACK_IE_ENABLED = on

TALKBACK_SAVE_XML_FILES = off

Please consult your Altera Quartus documentation for further information and implications about turning on the TalkBack feature. The latest version of the Altera Quartus tools do not require TalkBack to be enabled for Stratix II GX.

D.5 Troubleshooting

Limited visibility and control of signals in FPGA pose the biggest challenge in FPGA debugging. Some suggestions for successful implementation and debugging are:

- ❖ Ensure correct pin connections.
 - ❖ Ensure that the clock speed is correct (clock source is on board).
 - ❖ Ensure that the post-synthesis simulations have been successful.
 - ❖ If you encounter a problem, reproduce the problem in simulation if possible.
 - ❖ Use the debug signals provided on the Core. The following signals are useful for debugging:
 - ◆ pm_curnt_state[2:0]
 - ◆ xmlh_ltssm_state[4:0]
 - ◆ cxpl_debug_info[63:0] (This information is also available in Debug Registers 0 and 1.)
- If necessary, make other important signals visible.

D.6 Example DWC_pcie_ep FPGA Synthesis Summary Regression Results

Example DWC_pcie_ep FPGA synthesis summary regression results (core_clk frequency) are shown in [Table D-2](#). The PCI Express core FPGA timing results will be impacted by the logic that will be added to complete the design. The information provided here is intended to be used only a guideline. It is suggested that the complete design is synthesized and placed and routed before determining the final speed grade selection. An “X” in the table indicates timing was met. Configurations are defined in [Table D-1](#).

Table D-1 Configurations

Feature	std	std_mtu256	std_mq_mtu256	std_sg_mtu256
Multiple Functions	1	1	1	1
Multiple VCs	1	1	1	1
Queing Modes	Single	Single	Multi	Segmented
RAM at Top Option	No	No	No	No
Maximum Packet Sizes	128	256	256	256
Port Logic Registers	No	No	No	No
End-to-End CRC Support	No	No	No	No
Lane-to-Lane Deskew	Limit 5	Limit 5	Limit 5	Limit 5
Data Alignment	No	No	No	No
Max. No. of Tags	4	4	4	4

Table D-2 Example DWC_pcie_ep FPGA Synthesis Summary Regression Results

2sx1	xc2vp30_6	xc2vp30_7	xc4vfx60_11	xc4vfx60_12				
std	X	X	X	X				
std_mtu_256	X	X	X	X				
std_mq_mtu256				X				
std_sg_mtu256		X	X	X				
2sx4	xc2vp30_6	xc2vp30_7	xc2vp70_6	xc2vp70_7	xc4vfx60_11	xc4vfx60_12		
std	X	X	X	X	X	X		
std_mtu_256				X	X	X		
std_mq_mtu256				X	X	X		
std_sg_mtu256	X	X		X	X	X		
2sx8	xc2vp100_6	xc2vp100_7	xc2vp30_6	xc2vp30_7	xc2vp70_6	xc2vp70_7	xc4vfx60_11	xc4vfx60_12
std				X		X	X	X
std_mtu_256						X	X	X
std_sg_mtu256						X		X
std_sg_mtu256						X	X	X

E

App Note: Interrupts (Legacy, MSI, MSI-X)

This application note describes the application interrupt resources available to an Endpoint in a PCI Express system. Topic covered are:

- ❖ Legacy interrupts, Message Signaled Interrupts (MSI), and MSI-X from a system viewpoint, including Switch and Root Port handling of those interrupts
- ❖ Your application logic choices and issues for each type of interrupt
- ❖ DesignWare PCI Express IP selection, synthesis configuration, and run-time initialization for each type of interrupt.
- ❖ Additional MSI-X application logic notes and FAQ

For IP-specific details, refer to the user/reference manuals included with the version of the DesignWare Express Core that you are implementing in your design.

E.1 Introduction

The application logic in a PCI Express Endpoint may use one of three methods to signal an interrupt:

- ❖ PCI legacy interrupt
- ❖ MSI
- ❖ MSI-X

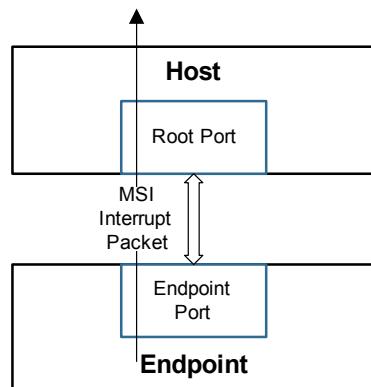
E.1.1 PCI Legacy Interrupts

PCI includes up to four interrupt wires, referred to as INTA, INTB, INTC, and INTD. These wires are shared by all the PCI devices in the system. PCI Express emulates this capability by providing Assert_INTx and Deassert_INTx Message packets sent through the PCI Express serial Link.

E.1.2 Message Signaled Interrupts (MSI)

A PCI Express Endpoint may signal an MSI by sending a standard PCI Express Posted Write packet towards the Root Port. The packet must contain a specific address and one of up to 32 data values.

The varying data values, and the address value provide more detailed identification of interrupt events than legacy interrupts.

Figure E-1 MSI Packet Generation by Endpoint

E.1.3 MSI-X

An MSI-X interrupt is identical to an MSI, except that an Endpoint may use one of up to 2048 address and data pairs in the MSI-X Posted Write packet. Endpoints with MSI-X capability also include application logic to mask and hold pending interrupts, as well as a memory table for the address and data pairs.

The large number of address values available to each Endpoint allows MSI-X Messages to be routed to different interrupt consumers in a system, as compared to the single address available to MSI packets.

E.2 Application Logic for PCI Express Interrupts

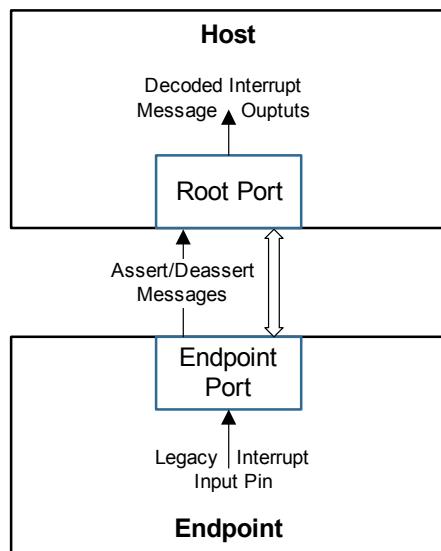
E.2.1 PCI Legacy Interrupt Application Logic

The DesignWare PCI Express Endpoint (EP) Core provides an input pin so that application logic can assert or deassert a legacy interrupt, as shown in [Figure E-2](#). The EP Core automatically maps assertion/deassertion edges on the input pin to PCI Express Assert or Deassert messages. Refer to [System Information Interface \(SII\)](#) for additional information.

Root Ports decode received Assert/Deassert messages into pulses. Refer to [System Information Interface \(SII\)](#) for additional information.

Note that a single-function Endpoint always uses virtual interrupt A (INTA).

Figure E-2 Legacy Interrupt Message Passing



E.2.1.1 Legacy Interrupt FAQ

1. Are legacy interrupts handled differently in a multi-function Endpoint?

In a multi-function Endpoint, each function has its own interrupt input pin. Each function's Interrupt Pin register determines which legacy interrupt Message the function uses (INTA, INTB, INTC, or INTD).

2. Is there a delay between the time a virtual interrupt input pin is asserted or deasserted and the corresponding change in the Root Port decoded Message output pins?

Unlike a real interrupt wire, the virtual interrupt is sharing the serial PCI Express Link. Therefore, transmission of the virtual interrupt Message could be delayed by the maximum length PCI Express data packet your Endpoint can transmit.

3. How are legacy interrupts handled in a PCI Express system that includes Switches?

The legacy interrupt messages are decoded by Switches and used to build virtual interrupt wires after each Switch Port. The state of the virtual wires is transmitted toward the Root Port by additional interrupt messages.

Note that switch ports may remap legacy interrupts as they pass through.

4. What is the typical usage model for legacy interrupts?

The legacy input pins on Endpoint core can queue a limited number of transitions to be converted into interrupt messages. The normal usage would be:

- a. Endpoint application logic asserts the interrupt input.
- b. The EP Core sends an Assert Message.
- c. Host software responds to message and clears the condition causing the interrupt.
- d. Endpoint application deasserts the interrupt input.
- e. The EP Core sends a Deassert Message.

5. What ordering considerations are there for legacy interrupts?

You may wish to guarantee that a legacy interrupt Message is sent after a data packet. In that case, do not assert the interrupt input pin until after the data packet's header is accepted by the core's transmit client interface.

In a complex PCI Express system, which include Switches and/or multiple Virtual Channels, you cannot guarantee that the interrupt Message will arrive after a data packet, unless the data packet uses the same Traffic Class as the legacy interrupt packet (Traffic Class 0).

6. Does the application need to deassert the virtual interrupt inputs if system software has disabled interrupts or if the PCI Express link has been placed in a low power state?

The EP Core does automatically send a Deassert Interrupt Message when software disables interrupts. However, the application must eventually deassert the virtual interrupt before it can send a new interrupt because the EP Core requires a rising edge on the virtual interrupt signal to generate a new Assert Interrupt Message.

The EP Core does not automatically send a Deassert Interrupt Message when the power state changes.

E.2.2 MSI Application Logic

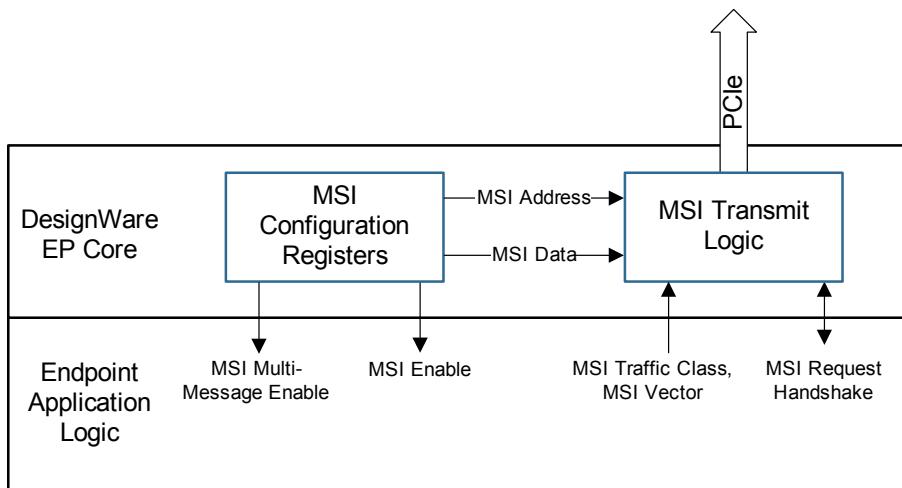
E.2.2.1 MSI Endpoint Application Logic

The DesignWare EP Core automatically builds an MSI packet for you (if MSI is enabled) whenever requested by your application logic. A simple handshake is required.

The EP Core informs your application logic whether MSI interrupts are enabled, and how many MSI data vectors have been allocated by system software.

Before performing the handshake, application logic must assert Traffic Class, and MSI vector information on the EP Core input pins, as shown in [Figure E-3](#).

The EP Core inserts the Traffic Class into the MSI packet. It also merges the MSI vector number into the MSI packet data field as described in the *DesignWare Cores PCIe Express Reference Manual*

Figure E-3 MSI Message Passing

E.2.2.2 MSI Root Port Application Logic

When an Endpoint sends an MSI Message, the Message packet is routed by address in the same way as any other PCIe Posted Write packet. In most systems, these packets will be routed to the Root Port and sent to Root Port application logic. The Root Port application logic recognizes the Posted Write packet by its MSI address and handles the interrupt in hardware or software. The data field is also available in the packet to further define the interrupt.

E.2.2.3 MSI FAQ

1. What is the purpose of the Traffic Class field in the MSI packet?

By setting the MSI packet's Traffic Class to the same value as an earlier data packet, you can guarantee that the MSI packet will always arrive at its destination after the data packet.

2. How many MSI vectors can I use?

You may request up to 32 vectors (data values) in the MSI configuration registers. However, system software may grant any number less than requested. Your application logic must map the requested vectors into the granted set.

3. Can I use legacy interrupts *and* MSI?

You may configure your Endpoint core and application logic to support both. However, you may only use one of these capabilities at a time. When host software clears the MSI Enable bit, you may only use legacy interrupts. When host software sets the MSI Enable bit, you may only use MSI.

4. How does a multi-function device handle MSI?

Each function in a multi-function device has its own configuration space, and therefore, its own MSI output controls. A multi-function core adds a function number to the MSI inputs shown in [Figure E-3](#).

E.2.3 MSI-X Application Logic

E.2.3.1 MSI-X Endpoint Application Logic

The address and data fields for MSI-X packets are defined in an MSI-X table located in the Endpoint's application logic, as shown in [Figure E-4](#). Each entry in the table corresponds to an MSI-X packet that the application may send.

Each MSI-X table entry also includes a mask bit for that interrupt. An additional table, the pending bit array (PBA), includes a pending bit for each MSI-X table entry.

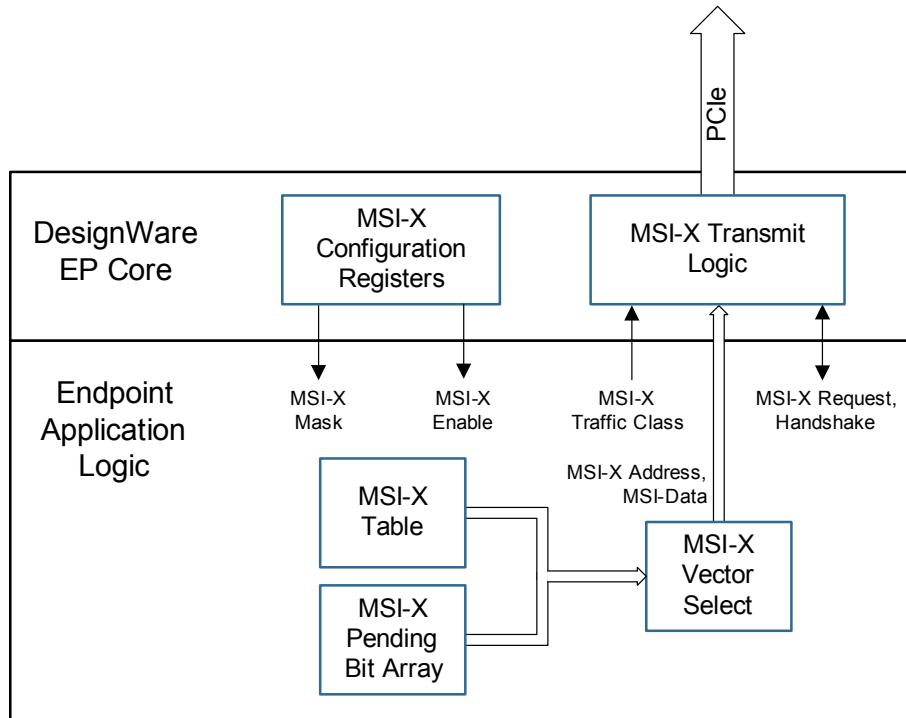
Host software sets and clears mask bits. If an interrupt's mask bit is set when the application wishes to send that MSI-X packet, the application must, instead, set the corresponding entry in the PBA.

Application logic monitors pending bits. If the corresponding mask bit is cleared, then the application sends the corresponding MSI-X packet, and clears the pending bit.

In addition, the MSI-X configuration registers in the EP Core include:

- ❖ Pointers to the function's MSI-X table and PBA
- ❖ Length of the function's MSI-X table
- ❖ An enable bit for MSI-X for that function
- ❖ A global mask for MSI-X

Figure E-4 MSI-X Message Passing



E.2.3.2 MSI-X Root Port and Other Application Logic

Just as an MSI packet, an MSI-X packet may be routed by address to Root Port interrupt logic. However, in more complex systems, MSI-X packets could be routed to different devices, including other Endpoints, based on the multiple address/data pairs available. In any case, the target of the MSI-X packet handles the interrupt just as a Root Port would.

E.2.3.3 MSI-X FAQ

1. How are the entries in the MSI-X table created?

See “[DesignWare IP Software Initialization](#)” on page [880](#).

2. What is the difference between MSI-X Enable and MSI-X Mask in the PCI Express configuration registers?

MSI-X Enable allows the Endpoint to send MSI-X Messages. It also disables legacy interrupts.

MSI-X Mask temporarily stops the Endpoint from sending any MSI-X Messages. Instead of sending the MSI-X Message, the Endpoint application logic must set the corresponding bit in the PBA. When the mask is deasserted, the application logic must send the MSI-X Message and clear the pending bit.

3. How are MSI-X interrupts handled in multi-function devices?

Each function has its own configuration registers, and therefore, its own control outputs from the core. A function number is added to the MSI-X request inputs. In addition each function has its own MSI-X table and PBA.

E.3 Interrupt Selection, Configuration, and Initialization

E.3.1 DesignWare IP Selection

All DesignWare PCI Express Root, Endpoint, and Dual Mode cores support the interrupt mechanisms described in this application note.

E.3.2 DesignWare IP Configuration

At synthesis time, you may choose to support MSI and/or MSI-X interrupts. Legacy interrupts are always supported.

At synthesis time, you may also choose initial values (reset values) for the MSI and/or MSI-X configuration registers. These are the register fields that appear as read-only to host software. The values that you may specify include:

- ❖ **MSI:** Number of MSI Messages requested, 32 or 64-bit MSI addressing
- ❖ **MSI-X:** Table size, BAR indicators, table offsets

For legacy interrupts, a single-function Endpoint always uses INTA. However in a multi-function Endpoint, you may choose which virtual interrupt is used for each function by setting the initial value of the Interrupt Pin configuration register.

E.3.3 DesignWare IP Hardware Initialization

After hardware reset, but before the PCI Express Link is initialized, you may choose to overwrite the hardware reset values referred to above. Application logic uses the core's DBI to write to the configuration registers.

E.3.4 DesignWare IP Software Initialization

After PCI Express communication is established, host software reads the read-only values described above for all devices in the system, then updates the writable configuration register fields as required. Some of the values related to interrupts are:

- ❖ **Legacy interrupt:** Interrupt Disable
- ❖ **MSI:** MSI Enable, number of MSI messages granted, MSI Address, MSI Data
- ❖ **MSI-X:** MSI Mask, MSI-X Enable

In addition, for MSI-X, host software writes the address and data pair values to the MSI-X table located in application logic, before enabling MSI or MSI-X.

E.4 Additional MSI-X Implementation Notes

E.4.1 Large MSI-X Tables

The MSI-X specification is written such that large MSI-X tables and PBAs may be implemented in compiled memories. However, if the host clears the function's global MSI-X mask, a search of the entire 2048 entry PBA might be necessary.

Also, in a large memory array like this, there would be no "memory" of the order in which the pending bits were set. This is not an MSI-X requirement, but might be required by some systems.

E.4.2 Application Control of Pending Bits

Pending bits are intended to "remember" requested, but masked, interrupts. Normally, a pending bit is only cleared when its MSI-X interrupt is unmasked. However, in some systems, application logic might be expected to also clear pending bits when the condition that caused the bit to be set is cleared.

E.5 Miscellaneous Interrupt FAQ

E.5.1 MSI, MSI-X, and Legacy Interrupts

1. Should I support all three interrupt mechanisms in my application logic?

Support of legacy interrupts and/or MSI may be required for backward compatibility. If host software enables MSI or MSI-X, legacy interrupts are automatically disabled. Functionality is undefined if both MSI and MSI-X are enabled.

2. Do the DesignWare PCI Express Cores support optional MSI per-vector masking?

No.

E.5.2 MSI-X Tables and Pending Bit Array

1. How does host software find the MSI-X tables?

Logically, the tables are in application memory space and are accessed from the host just like any application memory. Entries in the MSI-X configuration space indicate which BAR(s) point to the tables, and indicate the address offset of each table.

2. Can MSI-X Tables and PBAs be located in the same memory?

Yes, as long as the offsets are different. *PCI Local Bus Specification*, Revision 3.0 recommends certain spacing between tables for ease of system access mapping.

3. Can different functions share tables or memories?

Each function that implements MSI-X must have its own MSI-X table and PBA. However, tables may be mapped to different locations in the same physical memories.

4. Since MSI-X tables may be in compiled RAMs, there may be some unused bits. Are there any requirements on these bits?

Unlike other PCI Express register reserved bits, these unused bits may be read only or read/write. Host software is expected to preserve the value of these bits in case of possible future use.

5. What are the reset requirements for table bits.

The reset requirements are defined in *PCI Local Bus Specification*, Revision 3.0. Note that the requirements apply “after reset”, which means the host software’s view of the tables. Therefore, application hardware will need to initialize these bits before communication with the host begins.

E.5.3 MSI-X Ordering

1. Are there any PCI Express ordering rules that relate to MSI-X packets?

The normal PCI Express ordering rules for Posted Writes apply to MSI-X packets.

2. Will all MSI-X table entries be programmed by host software to unique values?

Not necessarily. Host software may choose to duplicate values.

3. Will the host process the interrupts indicated by the MSI-X packets in the same order that they are sent?

If the received packets are queued up, then they may be processed in order.

Also see the MSI question above concerning Traffic Classes, which may cause reordering of packets.

E.5.4 MSI-X and Switch/Root Ports

1. Are MSI-X packets used with DesignWare Switch and Root Ports?

Upstream Switch Ports can send MSI-X packets; Root Ports and downstream Switch Ports cannot.

2. How do I send an MSI-X packet toward the host from a Root Port or from a downstream facing Switch Port?

The DesignWare RC and SW Cores do not support this capability.

F

App Note: Order Enforcement Using the PCIe Core and AXI/AHB Bridge Modules

F.1 PCIe Ordering Rule Overview

This application note is written for applications that require certain ordering rules within PCIe traffic. There are many ordering rules according to the PCIe specification. The following is a general description from the PCIe 1.1. base spec.

Table 2-23 in the PCIe 1.1 base specification defines the ordering requirements for PCI Express transactions. The ordering rules defined in the spec apply within a single traffic class (TC). There is no ordering requirement among transactions with different TC levels.

Root Complexes that support peer-to-peer operation and Switches must enforce these transaction ordering rules for all forwarded traffic. These forwarding devices should not forward traffic from one virtual channel to another.

Basic ordering rules are summarized as follows:

1. Posted is permitted to pass posted transaction only if the relaxed order bit is set.
2. Non-posted is permitted to pass or to be blocked by non-posted.
3. Completion with different ID is permitted to pass or be blocked by other completions.
4. Posted must be allowed to pass all non-posted.
5. Posted is permitted to be blocked by completions.
6. Non-posted can never pass posted.
7. Non-posted is permitted to pass or be blocked by completions.
8. Completions can pass posted only if the relaxed order bit is set.
9. Completions must be allowed to pass all non-posted.

In general, ordering rules are used based on a specific device's functionality. Root Complexes with peer-to-peer support and switch devices must enforce the above rules. Rules #4 and #9 are a must. Therefore, posted and completion transactions must be allowed to pass non-posted. But for endpoint devices, it is device-specific. For example, it is possible that an endpoint device is allowed to have completions passing posted and non-posted requests. Because most of the above rules are considered as "permitted or blocked", it is legal to have "permitted" or "blocked" be determined by the architecture of the PCIe core and AHB/AXI bridge. The DWC PCIe core and AHB/AXI bridge IP designed such that the order rules may be enforced strictly or less strictly, based on the requirements of the application.

F.2 DWC PCIe Core Inbound Order Enforcement

There are three receive queue architectures within the DWC PCIe core. These queue structures will determine different order enforcement based on different receive queue architectures. If the core is configured with all transactions in bypass mode, then order enforcement should be performed by the application logic. The core will not be responsible for implementing PCIe order enforcement. Since it is an all bypass configuration, all transactions are presented onto the application interface in the order that they arrived.

Below are descriptions for different queueing architectures that are implemented in the DWC PCIe IP core. Three different queue modes are configurable and are designed to suit different applications.

Single queue:

In Single queue mode, a single header and data queue are populated in the DWC PCIe core for all transactions received. Completions can be bypassed if desired, based on the application. Single queue mode implements strong ordering (in order delivery). All transactions of the same or different traffic class are enqueued and dequeued in the order the transactions were received. For example, if a non-posted transaction is followed by a posted transaction, then the application interface (the DWC PCIe core's target1 interface) will have a non-posted transaction being presented, followed by a posted transaction. Even if the non-posted transaction is blocked by the application (halt), the posted transaction will not be allowed to bypass.

This queue mode has followed all ordering rules above except for rule #4 and rule #9 if completion is not in bypass. What this queuing mode implements is that posted transactions will be blocked by another posted, non-posted will be blocked by another non-posted, etc. That is, we have selected "block" as the default of the above ordering rules. According to the spec., these are permitted except for rule #4, and possibly rule #9, if the completion is not in bypass. Because of these violations, this queue mode is not likely appropriate for a Root Complex with peer-to-peer support or for switch designs. This is designed for endpoint devices that have no order rule requirement at the transaction forwarding between the DWC PCIe core receive queue and the core application interface. The advantage of this queue mode is that it is simple, resulting in an area and power advantage.

Multiple queue:

In Multiple queue mode, a header and data queue per type of transaction are populated in the DWC PCIe core. All transactions are enqueued in-order. All posted are enqueued into posted header/data queue. All non-posted are enqueued into a header/data non-posted queue and all completions are enqueued into a header/data completion queue. In multiple VC cases, each VC will have its own set of queues.

This queue mode has the same ordering as for single-queue mode. The user has further flexibility in setting which TLPs are bypassed and which TLPs are added to the ordering queue, but these additional modes offer little practical value in the normal case. Since it shares the same limitations as single-queue mode, this queue mode is not likely appropriate for a Root Complex with peer-to-peer support or for switch designs. This queue has some gate count advantage over the full order support provided by the segmented buffer; however, this advantage is generally minor, especially for multi-VC systems.

Segmented buffer queue:

A single header/data queue are populated in the DWC PCIe core for this queue mode. Transactions of different types are queued into a segmented buffer; each TLP type into a separate segment in the buffer. The transaction's order information is also queued into the segmented buffer. All posted requests are enqueued into the posted segment, all non-posted requests are queued into the non-posted segment and all

completions are enqueued into the completion segment. Each traffic class also has its own dedicated per-type segments.

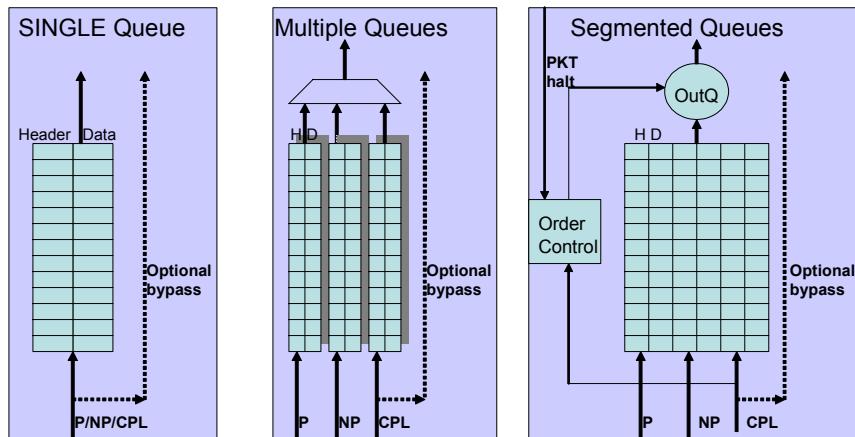
All PCIe order enforcement rules are strictly followed. All transactions are normally served in the order of reception, if the application has no blocking (halt) of the transactions received. If the application does have blocking conditions, then the order rules are enforced for transactions being read out of the queue. For example, if a non-posted request is received followed by a posted request, and the application could not take any more non-posted requests, then the posted will be allowed to pass the non-posted request and the posted transaction will be presented onto the application interface. In the second example, a posted request is received followed by a non-posted request, and the application can not take any more posted requests, then the non-posted transaction will be blocked even if the application can take the next non-posted transaction. Only when there are no previously received posted requests will the non-posted transaction be presented onto the application interface.

Since the order information is stored along with each transaction, full PCIe order enforcement can be accomplished through this mode of operation. This is the difference from the multiple queue mode above along with the single queue architecture.

Segmented buffer queue mode is designed for Switches and Root Complexes with peer-to-peer support. The primary advantage of this mode is the full support of PCIe ordering rules. In addition, this mode offers a reduced RAM requirement when compared with multi-queue mode, especially for multi-VC designs.

[Figure F-1](#) represents the various queue architectures of the DWC PCIe IP core inbound receiving traffic. There are three different queue modes. As described above, the ordering rules are implemented differently, based on different queue architecture.

Figure F-1 DWC PCIe Core Inbound Traffic Queue Architectures



F.3 DWC PCIe Core Outbound Order Enforcement

The DWC PCIe Core has a basic transmit architecture such that all transactions are presented outbound onto the PCIe wire based on the order the application presents them onto the DWC PCIe core transmit interfaces.

There are up to three application outbound transmit interfaces (client0, client1 and optionally, client2). All client interfaces are served default as round robin arbitration if credit is available, regardless of the type of transaction. For example, if a posted transaction is presented onto client1 followed by a completion transaction, and if credits permit, then the posted transaction will be transmitted onto the wire before the completion. If the credit is not available, then the completion can pass the posted and be sent onto the wire.

It is the responsibility of the application to make appropriate use of the three client interfaces. An application that has three independent threads of traffic can use the three client interfaces such that the DWC PCIe core will do the arbitration for the outbound transmission. For example, if the application has one source of outbound read transactions and one source of outbound write transactions, and they are independent sources, then client0 and client1 interfaces of the DWC PCIe core should be used to perform the outbound transmission of the transactions. If the read and write are related, then one single interface client0 should be used and it is up to the application to maintain the desired order.

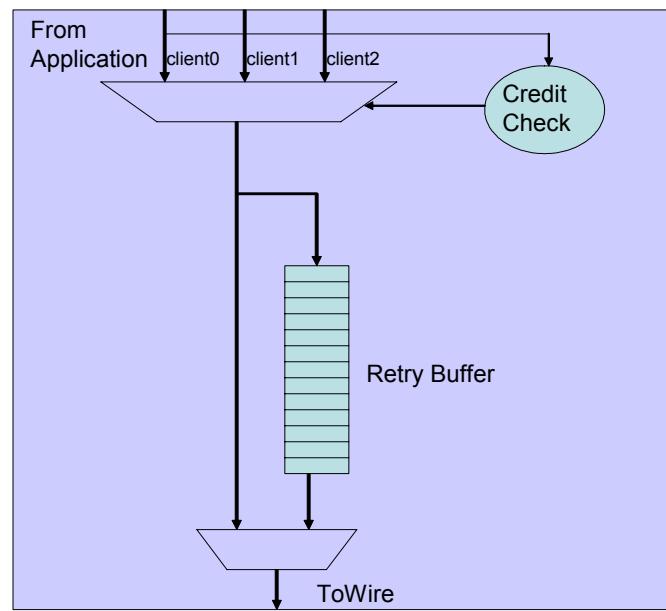
Most applications have independent threads of inbound reads such that its completion is not related to the outbound reads and writes. With these, an independent application interface such as client2 should be used for outbound completions.

In general, there is no transmit queuing in the DWC PCIe core. Therefore, there is no order enforcement among outbound transactions within the transmit queue. The DWC PCIe core has a cut-through transmit architecture where all outbound transaction order rules are enforced by the application. The DWC PCIe core transmits TLPs in the order they are accepted by the core.

Endpoint devices can use two or three client interfaces for posted, non-posted and completion outbound traffic. The two or three client interfaces are determined by whether there is a dependency between the posted and non-posted traffic.

Root Complex and Switch devices will likely use one client interface where the order enforcement is performed by the application logic, where the outbound transfer queue is near to the transmit interface. The credit information is an output of the DWC PCIe core, where it can be used as an input for taking outbound transactions out of the application's queue.

[Figure F-2](#) shows the outbound traffic queue architecture of the DWC PCIe core. As shown in the figure, there is no transmit queue implemented in the core. There is just a retry buffer for replaying PCIe traffic as required by the PCIe base specification. Therefore, outbound TLPs will pass directly from the client interface through the core and to the wire in the same order they were accepted from the application.

Figure F-2 DWC PCIe Core Outbound Traffic Queue Architectures

F.4 DWC PCIe AHB/AXI Bridge Inbound Order Enforcement

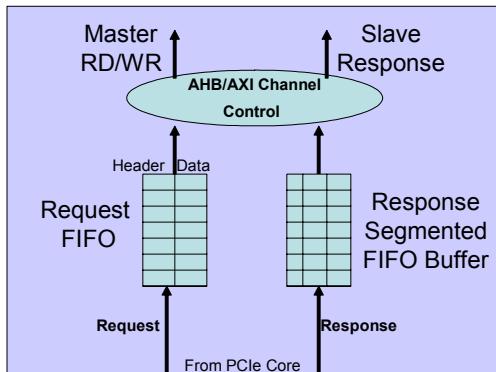
The DWC PCIe AHB/AXI bridge has a FIFO queue structure for inbound requests and a completion queue structure for outbound reads. These two queues are independently structured.

The AHB/AXI bridge has independent AHB/AXI channels for outbound and inbound transfers. Therefore, a read response can be presented on the read outbound AHB/AXI channel while a write request is being presented onto the write inbound AHB/AXI channel. Since the response of an outbound read and inbound write request can be presented onto the application logic of AHB/AXI simultaneously, then the PCIe traffic order rules do not get completely enforced by most AHB/AXI bridge applications.

The PCIe order rules are followed by the DWC PCIe core as described above. The order rules, in typical AHB/AXI bridge configurations, apply only to the PCIe core inbound queue and not after the inbound or outbound traffic has reached the AHB/AXI bridge. This means, for example, if a posted followed by a completion reaches the AHB/AXI bridge, then it is possible that the completion will reach the AHB/AXI slave response channel before the posted reaches the AHB/AXI master request channel. This depends on the readiness of AHB/AXI master request channel (if wait states are asserted from the AXI channel).

[Figure F-3](#) shows the queue architecture in the DWC PCIe AHB/AXI bridge. There are two major queues, one for slave responses which is a segmented buffer segmented on the number of outstanding reads, the other one for master request which is in the FIFO queue where requests are served in a first-come, first-served manner.

Figure F-3 DWC PCIe AHB/AXI Bridge IP Inbound Traffic Queue Architectures



There are many customer-configurable parameters for both the PCIe native core and the AXI Bridge. Order rule #8 states that completions should not pass posted if relaxed ordering is not set. As the figure above shows, once a completion has been taken out of the receive queues of the DWC PCIe core, it can pass posted. To enforce rule #8 for applications that require it, the DWC PCIe AHB/AXI bridge has a feature that may be employed. This feature will prevent completions or non-posted requests from passing posted requests by requiring that posted transactions all complete through the bridge to an AHB/AXI interface before a completion or non-posted can be taken out of the receive queue. This ensures that previous posted transactions are never passed by a completion or non-posted request. This feature is turned on when the DWC PCIe core is configured to have a receive queue in the segmented buffer mode and completion is in stored-forward mode.

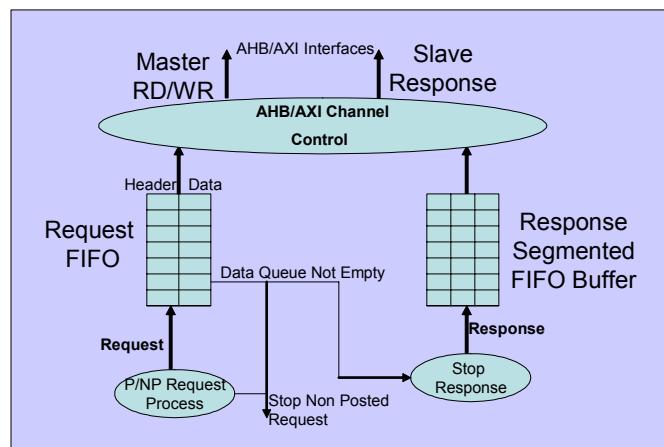
The limitation of the above feature is that it gives the posted transactions priority such that it can always pass through the bridge when it is available, while the non-posted and completions are blocked.

[Figure F-4](#) shows the feature that was implemented to support rule #8.



Rule #4 (posted transactions must be allowed to pass non-posted) is not satisfied even with this implementation, because there is a single FIFO queue for both posted and non-posted. If an application desires to have this rule enforced, then it is required that the application setup the FIFO depth to be one entry such that there is only one transaction that can be served at a time. There is a performance penalty. Please keep in mind, even if the bridge does not block a posted transaction, there will still be a blocking or violation of rule #4 if the AHB/AXI interface has asserted wait states which block a non-posted transaction (read).

Figure F-4 DWC PCIe AHB/AXI Bridge IP Inbound Traffic Order Enforcement

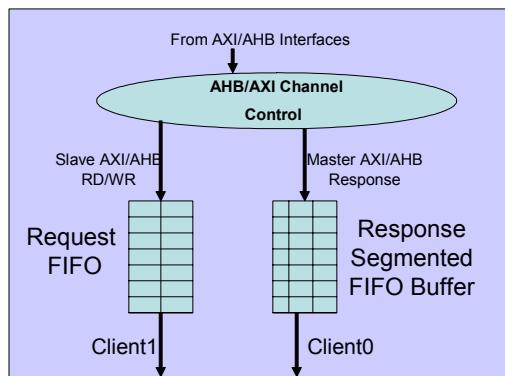


F.5 DWC PCIe AHB/AXI Bridge Outbound Order Enforcement

The DWC PCIe AHB/AXI bridge has a FIFO queue structure for outbound requests and a completion queue structured per inbound reads (see [Figure F-5](#)). These two queues are independently structured. Therefore, there is no order to be maintained once outbound requests or inbound responses have entered into the queue.

This default design architecture is not intended to maintain the special order rules such as rule #4, rule #8 and rule #9. It is designed to serve outbound traffic as a FIFO (first-come, first -served). For applications that must maintain the outbound traffic special ordering rules, please contact Synopsys support.

Figure F-5 DWC PCIe AHB/AXI Bridge IP Outbound Traffic Queue Architectures



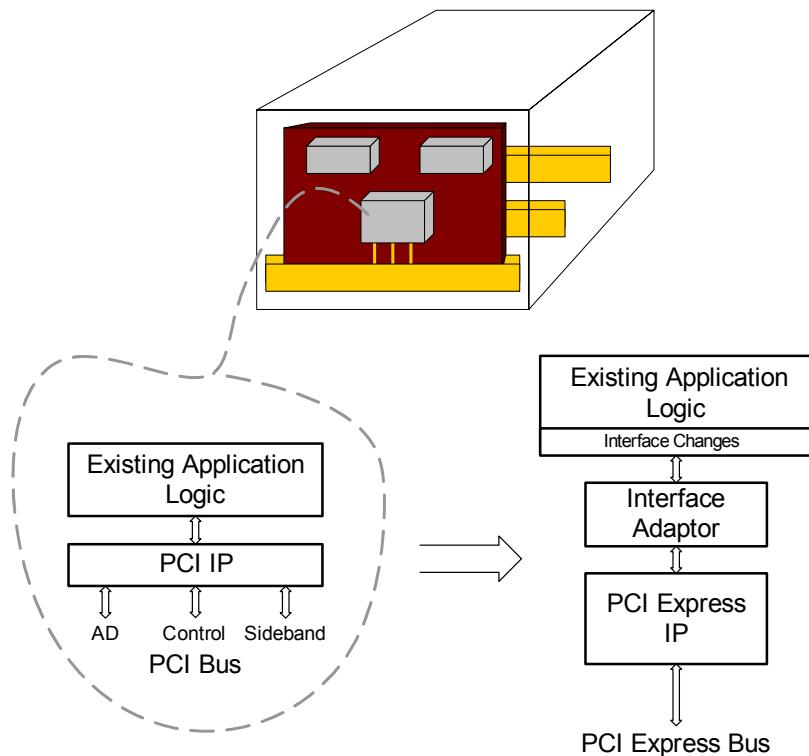
G

App Note: Application Interfaces for PCI and PCIe

PCI slots in personal computers and other applications are being replaced with low-pin-count, high-speed PCI Express connectors. For that reason, you may wish to modify existing PCI plug-in cards to be compatible with PCI Express. There are two basic approaches:

- ❖ Redesign your application logic to take full advantage of PCI Express throughput and other features, or
- ❖ Add an interface adaptor to your application logic to use available PCI Express IP to achieve adequate throughput and latency, with minimal design time, as illustrated in [Figure G-1](#)

Figure G-1 Converting PCI to PCI Express



This application note will help you understand the differences between the PCI and PCI Express IP application interfaces, so that you can decide how to build PCI Express into your product. The emphasis is on the application interfaces, and *not* on the protocols.

It is assumed you are very familiar with PCI. Some basic understanding of PCI Express is helpful, but not required.

Topics covered are:

- ❖ A brief comparison of PCI and PCI Express
- ❖ A comparison of PCI and PCI Express IP application interfaces
- ❖ A checklist of features required in an adaptor
- ❖ Summary

G.1 PCI to PCI Express Comparison

PCI Express was designed as a logical, but not physical replacement for PCI Express, as you will see in the following comparison.

G.1.1 PCI and PCIe Physical and Link Layers are Not Compatible

At the physical and link levels, PCI and PCI Express are not compatible. [Table G-1](#) compares the pins on the simplest PCI (32 bit, 33 MHz) with the simplest PCI Express Endpoint (single Lane, 2.5 GHz). JTAG and serial management pins are not included.

Table G-1 PCI and PCI Express Physical Characteristics

Feature	PCI	PCI Express
Basic signals	49 (including 32-bit shared address/data)	7
64-bit extension signals	39	0
Interrupt and other controls	7	1
Data transport	Address/Data with optional parity	Serial packets with CRC and hardware retry
Data encoding	None	8b10b differential
Bus style	Multi-drop, 33 MHz, parallel	Point-to-point, 2.5 GHz in each direction, serial

G.1.2 PCI and PCIe Bus Transactions are Similar

In contrast to the physical layer, PCI Express transactions are derived from PCI. PCI Express also emulates PCI sideband pins using Messages sent over the serial Link. [Table G-2](#) compares PCI transactions and PCI Express transactions.

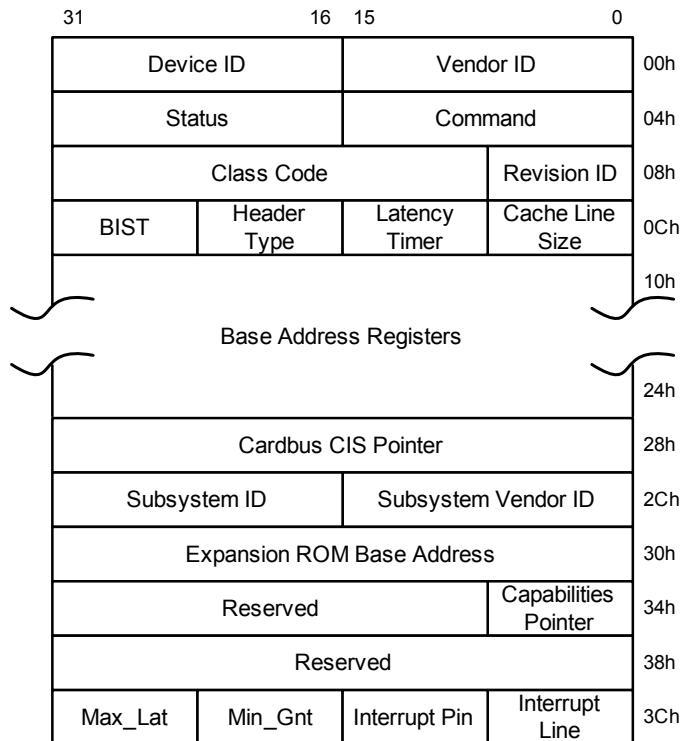
Table G-2 PCI and PCI Express Transaction Comparison

Transaction	PCI	PCI Express
Memory Write	Memory Write, Memory Write and Invalidate	Memory Write
Memory Read	Memory Read, Memory Read Line, Memory Read Multiple	Memory Read
I/O access	I/O Read, I/O Write	I/O Read, I/O Write
64-bit address	Dual Address Cycle	64-bit address in packets
Other	Interrupt Acknowledge, Special Cycle	
Interrupts	INTA, INTB, INTC, and INTD pins	INTA, INTB, INTC, and INTD Messages
Power Control	PME and WAKE pins	PME Message, beacon, optional WAKE pin

G.1.3 PCIe and PCI Base Configuration Registers are the Same

PCI Express Endpoints always include a full set of PCI compatible configuration registers shown in [Figure G-2](#).

Figure G-2 PCI Configuration Registers



G.1.4 Other Differences Between PCI and PCI Express

[Table G-3](#) summarizes several additional differences between PCI and PCI Express.

Table G-3 Additional PCI and PCI Express Differences

Item	PCI	PCI Express
Independent transmit and receive Links	PCI shares a single physical bus between Read and Write transactions	A PCI Express Link always has independent transmit and receive wires
Interrupting transactions	A PCI transaction may be interrupted at almost any time and resumed later	PCI Express transactions are in packets which must be completed without interruption
Transaction length	Limited by Latency Timer in configuration register	Limited by Max Payload Size and Max Read Request size in configuration registers
Address crossing 4-KB boundary	No limit	Address may not cross 4-KB boundary during transaction

Table G-3 Additional PCI and PCI Express Differences (Continued)

Item	PCI	PCI Express
Read completion boundary	Read data may be returned in multiple responses of any size.	Read data may be returned in multiple packets, but packets (except for the last) must end at 64 or 128-byte boundaries.
Optional, advanced features		PCI Express includes optional advanced features such as quality of service (VC/TC), advanced error reporting, end-to-end data integrity (ECRC), multiple split Read transactions, and choice of throughput (Lanes)

G.2 A Comparison of PCI Express and PC IP Application Interfaces

For comparison purposes, the following IP cores are used:

- ❖ Synopsys DesignWare Native PCI Controller
- ❖ Synopsys DesignWare PCI Express Endpoint (EP) Core

G.2.1 Transmit Interfaces

[Figure G-3](#) compares the PCI and PCIe IP transmit application interfaces. [Table G-4](#) summarizes the differences between the two.

Figure G-3 Comparison of Application Transmit Interfaces

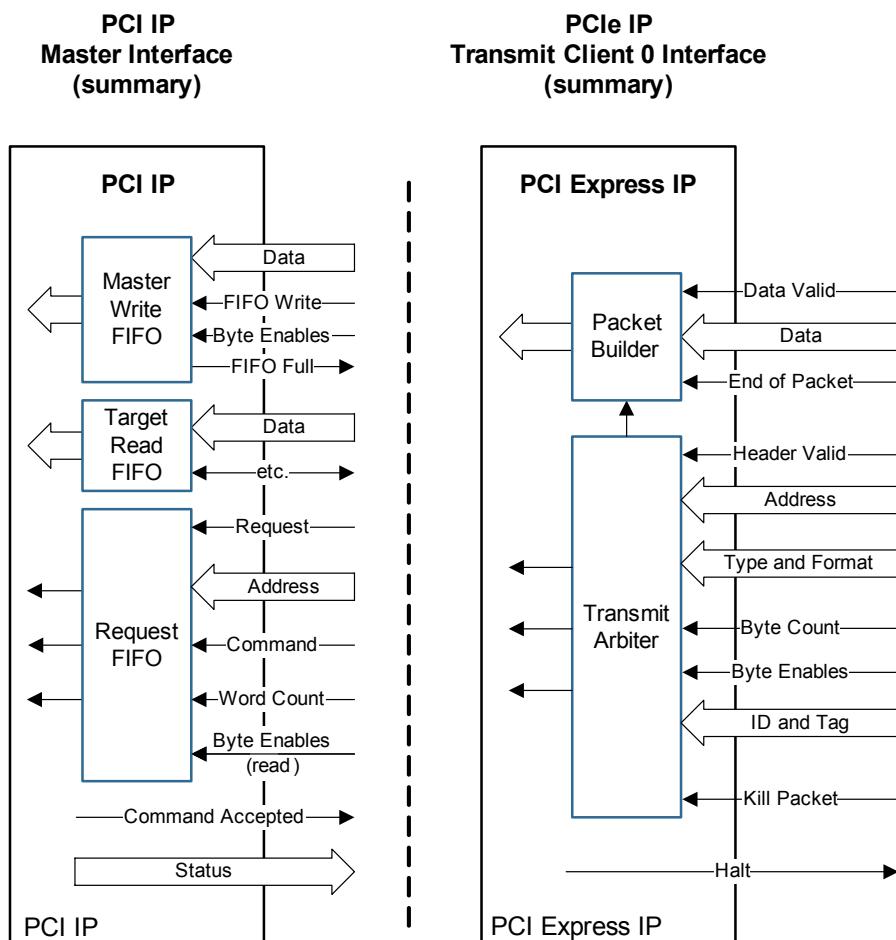


Table G-4 Transmit Application Interface Comparison

Topic	PCI Transmit (Master I/F)	PCIe Transmit Interface (Client 0)
Data Inputs	Two data inputs: one for returning read data, the other for write data	Common data input for transmit Requests and Completions (returned read data). ¹
Input FIFO	FIFOs for data inputs	No FIFOs for data inputs
Byte enables for Read/Write	Separate byte enable inputs for Read and Write	One byte enable input
Byte enables per data phase	Byte enables apply to each data phase of a transaction. No restrictions (in protocol) for non-contiguous byte enables.	Byte enables only apply to first and last DWORD. Protocol only allows non-contiguous byte enables for 1 or some 2 DWORD transactions. Additional restrictions exist for other transactions.
When byte enables must be available	Byte enables are sent during the corresponding data phase.	Byte enables are sent in the packet header. Therefore, they must be available before any data is sent.
Data and command timing	Data and command inputs may be applied in any order (command first, data first, or command and data at the same time). The controller aligns the command and data.	Data and command (header) must be applied at the same time.
Input data pause	Application may pause data entry during a transfer, because protocol allows pauses and resumption.	Application must continuously supply data during a transfer, because protocol requires continuous transfer during a packet.
Kill transaction	If a transaction is ended, all previously sent data is valid.	If a PCIe transaction is killed, the entire packet is cancelled.
ID and Tag (for Read responses)	Not required by protocol.	Required by PCIe protocol. Allows intermixing of Read responses for different Read requests.

1. The PCI Express IP can be configured to have one, two, or three independent transmit interfaces (clients). Each of the transmit interfaces has its own data and request input. [Figure G-3](#) shows the simplest case with one application transmit interface.

G.2.2 Target Receive Interfaces

Figure G-4 compares the PCI and PCIe IP target receive interfaces, which handle target Write and target Read requests. Table G-4 summarizes the differences between the two.

Figure G-4 Comparison of Application Receive Interfaces

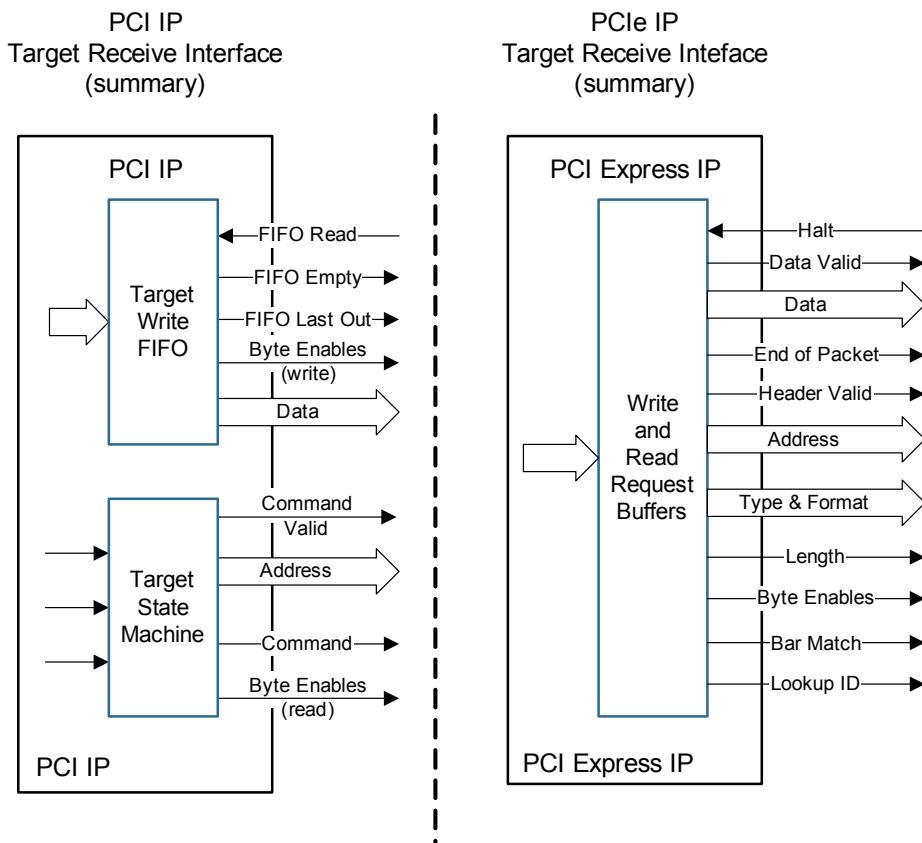


Table G-5 Application Receive Interface Comparison

Topic	PCI Target Interface	PCIe Target Interface
FIFO/buffer	Cut-through	Bypass, cut-through, or store-and-forward ¹
Byte enables for Read/Write	Separate	Common
Byte enables per data phase	Apply to each data phase	Protocol requires that they only apply to first and last DWORDs of the transactions
Error reporting	May occur at any time during a transaction; earlier part of transaction is valid	An error cancels the entire transaction (packet) ²

1. PCIe IP receive buffer options:

- a. Bypass: No FIFO storage; application cannot stop transfer.
- b. Cut-through: Transaction is available immediately to application, but may be stored in FIFO if application halts transfer.
- c. Store-and-forward: Transaction is fully buffered in FIFO, validated, then made available to application.

2. PCIe IP error handling:

- a. Bypass and cut-through: The core signals the error to the application, which must cancel any previous portion of transaction.
- b. Store-and-forward: A bad packet will never be seen by the application.

G.2.3 Read Response Interfaces

Figure G-5 compares the PCI and PCIe IP Read response interfaces. Table G-6 summarizes the differences between the two.

Figure G-5 Comparison of Application Read Response Interfaces

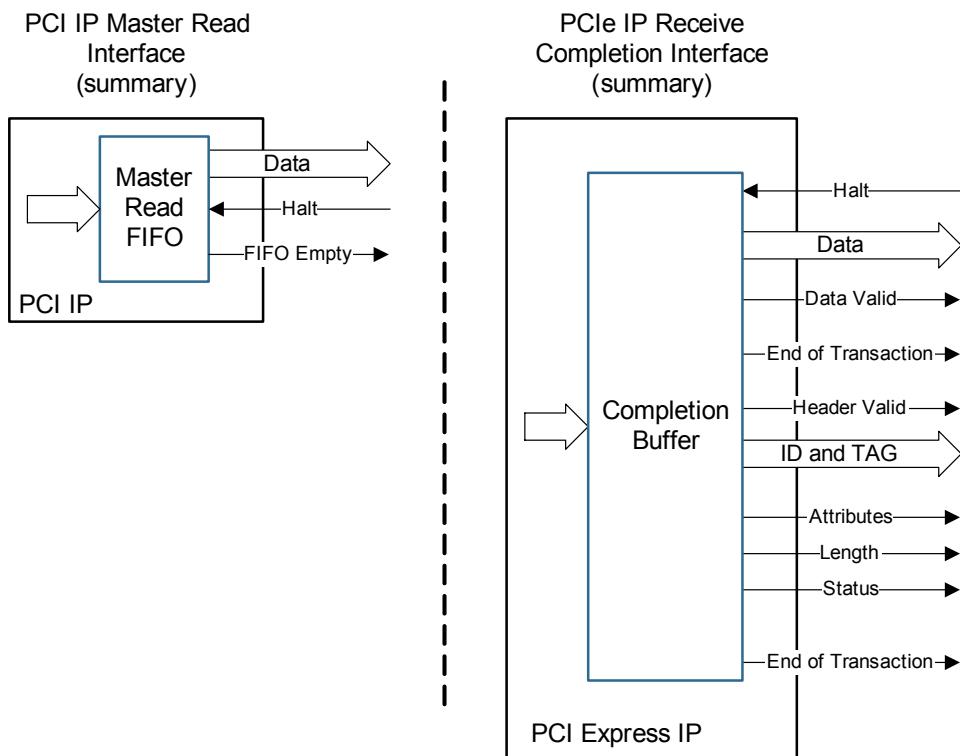


Table G-6 Read Response Interface Comparison

Topic	PCI Target Interface	PCIe Target Interface
Identify original Read request (ID and Tag)	Not required since PCI device will only have one outstanding Read request	You must return this to the PCIe IP when you deliver the Read response (Completion)
FIFO/buffer	Cut-through	Bypass, cut-through, or store-and-forward
Byte enables per data phase	Apply to each data phase	Protocol requires that they only apply to first and last DWORDs of the transactions. Byte enables are generated at the master interface where the request is made.
Error reporting	May occur at any time during a transaction; earlier part of transaction is valid.	An error cancels the entire transaction (packet).

G.2.4 Read-Only Configuration Initialization

As mentioned earlier, both the PCI and PCIe IP provide identical base configuration registers. The fields in these registers appear as either read-only or read-write to host computer software. You can configure the read-only fields at synthesis time. In addition, for more flexibility, you can overwrite the read-only fields from your application logic, just after reset is released.



The PCI Express protocol also provides additional configuration registers. These registers may be initialized at synthesis time with the DesignWare coreConsultant tool, or you may initialize them in the same way described here for the base registers.

Both the PCI and PCIe IP provide an interface for read-only register initialization:

- ❖ The PCI IP includes a serial EEPROM interface for this purpose.
- ❖ The PCIe IP provides a simple register interface (DBI).

G.3 Adaptor and Redesign Checklist

Whether you choose to design an adaptor, or to redesign your application logic, you should consider the following primary differences for PCI Express as compared to PCI:

- ❖ You must provide data continuously during a transmit transaction.
- ❖ Byte enables only apply to the first and last byte of a transaction; there are additional restrictions for enabling of non-contiguous bytes.
- ❖ All byte enable information must be available before you start the transaction.
- ❖ You must supply address and the first data values simultaneously to the PCIe IP.
- ❖ If you cancel a transaction in progress, the entire transaction is cancelled, not just the data after the cancellation.
- ❖ If you receive a Read request, the PCIe IP will deliver an [ID, Tag] identifier with the request. You must return this identifier with the Read response.
- ❖ You must observe the packet length, Read request limit, Read completion boundary, and 4-KB address requirements for PCI Express.
- ❖ Remember that the PCI Express IP application interface probably operates at a different clock frequency than your existing PCI interface.

G.4 Summary

Depending on your requirements, you may choose to adapt existing PCI application logic to use PCI Express IP. When making this choice, you must trade off design and debug time versus complexity of adaptor logic, efficiency of data transfer, and access to advanced PCI Express features.

H

App Note: PCI Express Port Bifurcation

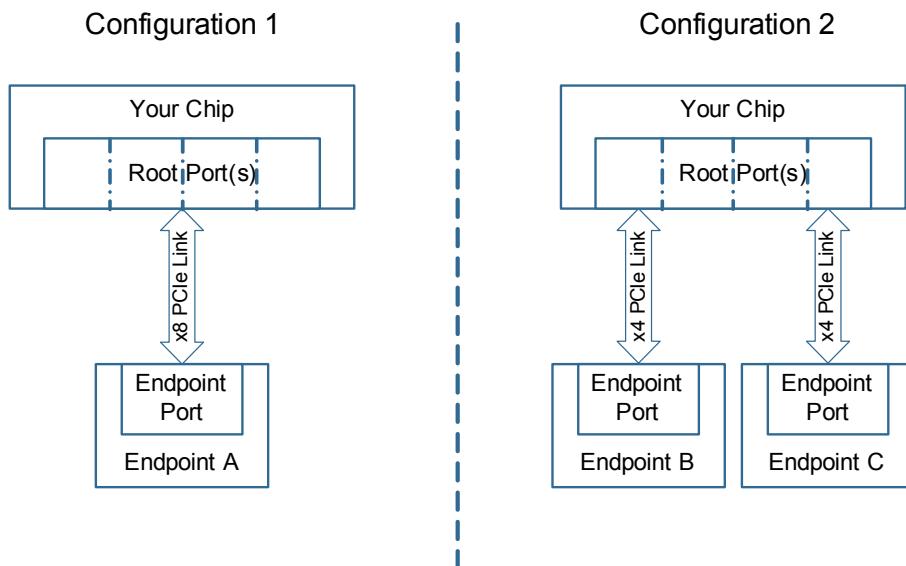
For each PCI Express Port on a chip you must choose the number of pins required, based on the Port's maximum Link width (from 1 to 32 Lanes). You may be able to save pins by allowing flexible use of these Ports. For example, suppose there are enough pins on your PCI Express Port for a Link width of eight Lanes, but you wish to use those pins in more than one configuration, such as shown in [Figure H-1](#) where the x8 Port can be configured as:

- ❖ One single x8 Link, or
- ❖ Two x4 Links

This technique is known as Port bifurcation or Port splitting, and is the subject of this application note, which presents:

- ❖ A brief review of PCI Express components and Links
- ❖ A number of PCI Express Port design techniques for bifurcation
- ❖ A detailed bifurcation example using DesignWare PCI Express IP

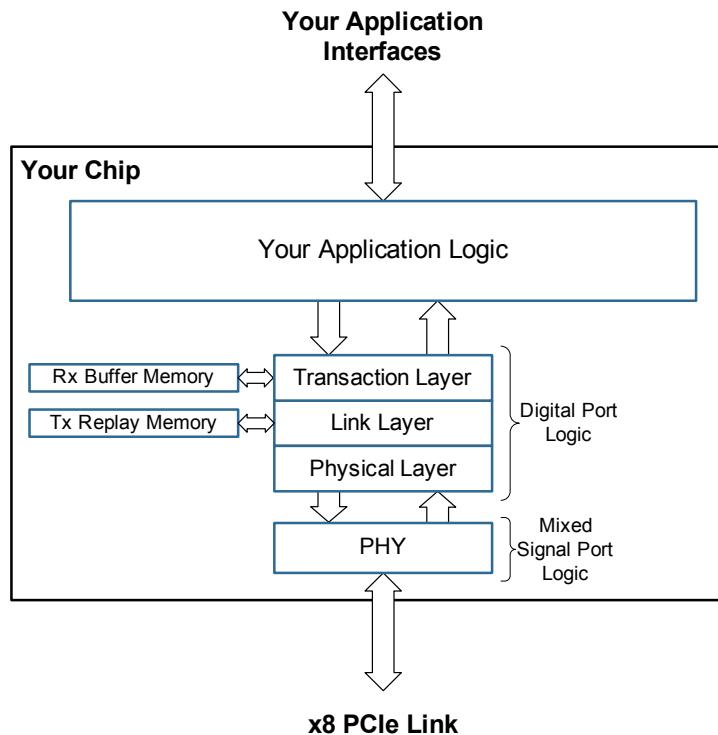
Figure H-1 Bifurcation Example: Flexible use of your Chip Pins



H.1 PCI Express Port

As a very brief review of PCI Express Ports before discussing bifurcation, consider the PCI Express Port example shown in [Figure H-2](#).

Figure H-2 PCI Express Port



Some of the tasks performed by the blocks in [Figure H-2](#) are:

- ❖ **PHY:**
 - ◆ Conversion between differential low voltage external signaling and internal logic levels
 - ◆ Serialization/deserialization; 8b/10b encode/decode
 - ◆ Detection of components on the other side of the Link; detection of active and idle states
 - ◆ Generation of master clock for transmit data and digital IP; transfer of received data from receive clock domain to transmit clock domain
- ❖ **Physical Layer:**
 - ◆ Negotiation with opposite side of Link with respect to data rate, Link width, data and Lane reversal
 - ◆ Data scrambling/descrambling; skip character insertion/deletion (for clock difference compensation)
 - ◆ Data distribution to/from active lanes; packet framing/deframing
- ❖ **Link Layer:**
 - ◆ Reliable data transmission (Ack/Nak, packet storage for replay)

- ◆ Reception/transmission of Data Link Layer Packets (DLLPs) such as Ack, Nak, and Flow Control
- ◆ CRC checking/generation
- ❖ Transaction Layer
 - ◆ Data packet assembly/disassembly; packet format checks
 - ◆ Management of receive buffers; transmission of buffer space information (credits)
 - ◆ Gating of packet transmission based on remote buffer space (credits)
 - ◆ Power down state machines
 - ◆ Configuration registers for Link control (for example, maximum payload size)

H.2 Bifurcation Methods

H.2.1 Bifurcation Approaches

Given the mixed-signal and digital logic described earlier, there are three basic approaches to Port bifurcation:

- ❖ Lane splitting
- ❖ Multi-threading and selective duplication
- ❖ Multiple instantiation

Lane splitting slices logic dynamically according to need. This works well for a modularly designed PHY (mixed-signal logic), but is generally not applicable to the digital Port logic.

To use multi-threading and selective duplication, we observe that dividing a x8 Link into two x4 Links halves the data rate in each x4 section. So, we can potentially maintain our data path and clock rate and, on consecutive clock cycles, alternate between the two x4 links. The disadvantages of this approach include:

- ❖ The need to maintain duplicate states for the two links
- ❖ A severe verification burden on the Port designer due to the increased state logic
- ❖ A complex application interface, which may have to handle interleaved packet data

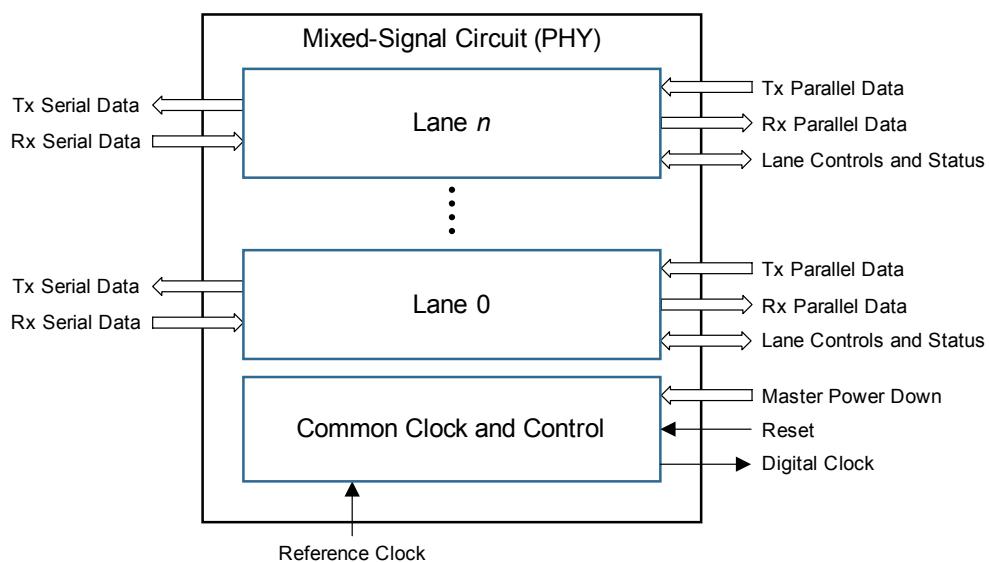
The third approach is multiple instantiation, where we add additional, but smaller, Port logic to handle the second Link, and some trivial steering logic. This approach has the advantage of simplicity, especially when using pre-verified IP. Each set of Port logic has its own normal application interface.

The following sections discuss the approaches in detail, followed by a summary comparison of the approaches.

H.2.2 PHY (Mixed-Signal) Bifurcation

PCI Express PHYs, with independent control and status signals per Lane, provide simple support for bifurcation using logic splitting, as shown in [Figure H-3](#).

Figure H-3 Logic Splitting in a PHY with a Modular Design



H.2.3 Digital Logic Approach 1: Multi-threading and Duplication

If we use multi-threading, we could design the PCI Express digital Port logic to operate in standard or bifurcated mode.

As mentioned before, dividing a x8 Link into two x4 Links halves the data rate in each x4 section of the Link. So, we can potentially maintain our data path and clock rate and, on consecutive clock cycles, alternate between the two x4 links.

The obvious problem is that we have to maintain state for each of the Links. Because the PCI Express protocol operates at a high speed, the logic is heavily pipelined. Alternating between x4 links would mean doubling the number of registers, and adding extensive selection logic and multiplexors. Therefore, gate count savings are minimal over multiple instantiation.

The replay and receive buffer memories could be partitioned if we add pointers and logic to handle the bifurcated case. The application interface would have to handle interleaved packets.

“Static” logic, such as the LTSSM and configuration registers, would have to be duplicated, and steering logic would be required to handle the two independent Links.

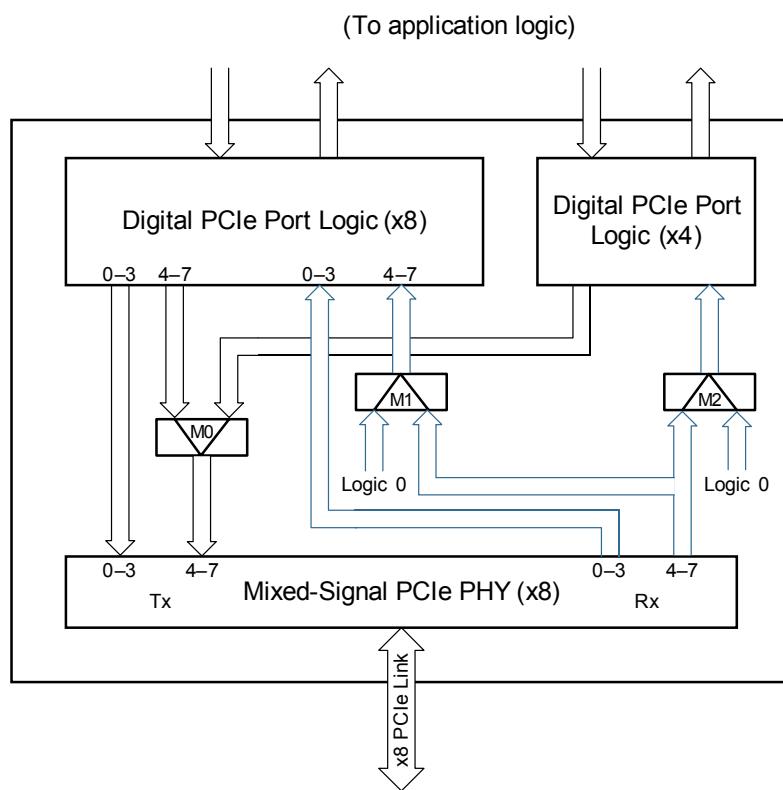
Another disadvantage to this approach is that standards-based logic, such as a PCI Express Port, requires extensive verification and certification over all combinations of operating modes. Verification of the multi-threading/splitting bifurcation approach would be very complex due to the high number of additional internal register states.

H.2.4 Digital Logic Approach 2: Multiple Instantiation

In the multiple instantiation approach, we instantiate additional, but smaller digital Port logic. [Figure H-4](#) shows an example where we have added standard 4 Lane (x4) PCI Express logic to the 8 Lane (x8) logic already present. Trivial steering logic chooses between x8 or dual x4 operation. Properly designed x8 Port logic automatically runs at x4 if only 4 Lanes are connected.

The PHY does not need an additional instance, if it is built with a Lane-oriented modular design. All multiplexing occurs in the lower-speed digital domain.

In this approach, each instance of the digital Port logic has its own standard application interface.

Figure H-4 An Additional Instance of Port Logic to Support Bifurcation

H.2.5 Summary Comparison of Bifurcation Approaches

Table H-1 shows the relative complexity of the multi-thread and multiple instance bifurcation approaches presented in this application note. A complexity rating of 1.0x refers to the original x8 core. Higher numbers indicate increasing complexity. The multi-instance approach results in significantly lower complexity and risk.

Table H-1 Relative Complexity of Bifurcation Methods

Characteristic	Multi-thread Approach	Multi-instance Approach
Gate Count	1.5x	1.7x
Digital Logic Complexity	2.0x	1.0x
Verification Complexity	4.0x	1.0x
Application Interface Complexity	3.0x	2.0x
Totals	11.5x	5.7x

H.3 Bifurcation Example

H.3.1 Overview

The following sections describe the logic to implement a x8 to x4 bifurcation example. The example uses Synopsys DesignWare digital and mixed-signal PCI Express IP (core and PHY).

H.3.1.1 IP Selection

For this example, we will use the following DesignWare IP blocks:

- ❖ **x8 core:** DesignWare PCI Express Root Complex Port (RC) Core with a 64-bit wide datapath, operating at 250 MHz. The receive and transmit interfaces between the digital IP and the mixed-signal IP transfer 8 characters per clock cycle per Link.
- ❖ **x4 core:** Same as x8, but the datapath is 32 bits wide, and 4 characters are transferred per clock per Link.
- ❖ **PHY:** Includes a clock module, and 8 PCIe Lanes with receive and transmit logic. The DesignWare PCI Express PHY includes independent controls and status signals for each Lane, to enable Port bifurcation.

H.3.1.2 IP Configuration

No special synthesis or runtime configuration is required for the digital cores. When only four Lanes are connected to the x8 core, the core will automatically configure itself for x4 operation.

H.3.2 Digital IP to PHY Connections

The digital IP to PHY interface signals are divided into per-Lane digital IP output signals, per-Lane digital IP input signals, non-per-Lane (power down) signals, and clocks and resets.

H.3.2.1 Per-Lane Digital IP Outputs

The digital IP output signals listed in [Table H-2](#) from the x8 core are connected directly to the corresponding input pins on the PHY for Lanes 0 through 3.

For Lanes 4 through 7, the signals are connected to the multiplexor shown in [Figure H-4](#).

Table H-2 Per-Lane Digital IP Output Signal Connections

Digital IP Signal	PHY Signal
mac_phy_txdata	tx{lane}_data
mac_phy_txdatak	tx{lane}_datak
mac_phy_txdetectrx_loopback	tx_detectrx[lane]
mac_phy_txcompliance	tx_compliance[lane]
mac_phy_rxpolarity	rx_polarity[lane]
mac_phy_txelecidle	tx_elecidle[lane]

H.3.2.2 Per-Lane Digital IP Inputs

The PHY output signals listed in [Table H-3](#) are connected directly to the corresponding input pins on the digital IP for Lanes 0 through 3.

For Lanes 4 through 7, the signals are connected to the multiplexor shown in [Figure H-4](#). Note that on the digital IP, multi-Lane data is presented as a single concatenated bus. The per-Lane signals must be obtained by performing slicing on given bus.

Table H-3 Per-Lane Digital IP Input Signal Connections

Digital IP Signal	PHY Signal
phy_mac_rxdata	rx{lane}_data
phy_mac_rxdatak	rx{lane}_datak
phy_mac_rxvalid	rx_valid[lane]
phy_mac_rxstatus	rx{lane}_status
phy_mac_phystatus	phystatus[lane]
phy_mac_rxelecidle	rx_elecidle[lane]

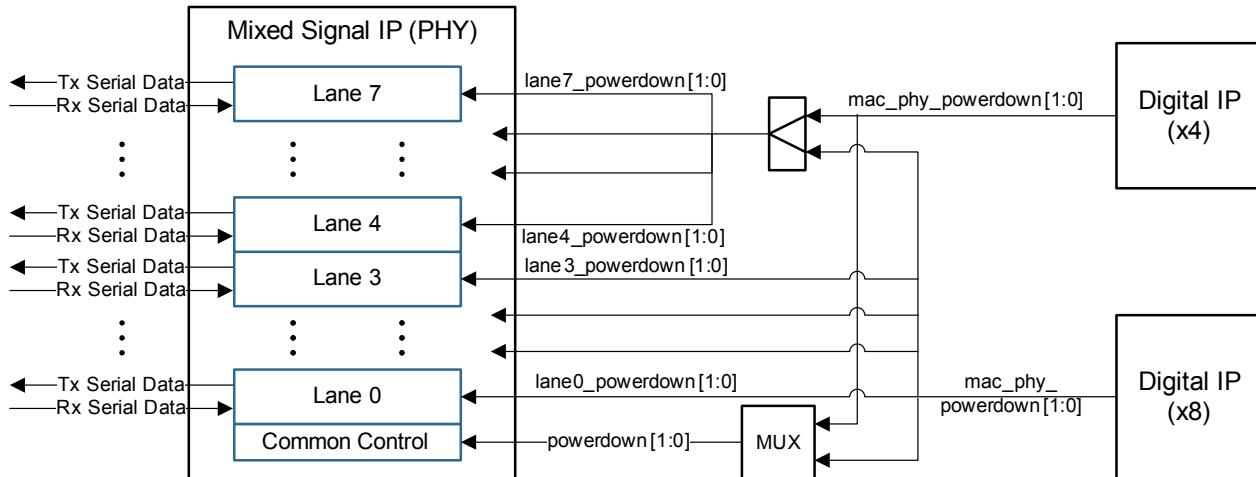
H.3.2.3 Digital IP signals (not per-Lane)

[Figure H-5](#) displays the connections between the digital IP power down outputs and the PHY power down input controls.

The PHY master power down is controlled from a mux, which selects the highest power state between the x8 and x4 cores. Note that when the x4 core is not in use, you will need to force its power down output to the lowest power state with some simple logic.

The per-Lane power controls are simply hardwired to the x8 PHY for Lanes 0 to 3, and selected by a mux for Lanes 4 to 7.

Figure H-5 Simple Logic Allows Two Digital IP Cores to Control PHY Power Down



H.3.2.4 Clocks and Resets

In a simple setup, the clocks and reset can be common to the x8 and x4 cores.

To lower power consumption, you may choose to separate the clocks and resets to the two digital cores. That way you can hold the x4 core in the reset state with its clock off when it is not being used.

H.3.3 Lane Reversal Considerations

When using bifurcation, Lane reversal is supported on the Links connected to the digital IP. Note that this typically requires that Lane 0 of the digital IP is part of the working Link.

H.3.4 Static or Dynamic Bifurcation

The simplest Port splitting scheme method is static bifurcation, where you know the desired operating mode before you release reset.

It is also possible to start up your chip with only the x8 core operating. You can wait until the x8 core is operating, and observe its operating width. If it is four or less and the Link is only using Lanes 0 to 3, then you could enable the x4 core, and allow it to begin operating.

Your on-chip firmware/software can determine the operating state and Lanes used by reading the PCI Express Link Status configuration register.

I App Note: Selecting PCI Express IP for your Design

I.1 Introduction

PCI Express, the next generation of the PCI bus, is being widely adopted in today's high-performance PCs, servers and embedded applications. This high bandwidth protocol keeps the same software interface and many of the key features of PCI, but has a number of differences and new features. The biggest changes with PCI Express are the use of serial data transfers and gigahertz clock speeds, which make the protocol more complex, but provide significant improvement in data throughput.

This application note provides a very brief introduction to the PCI Express protocol and explains how selecting the right digital and mixed-signal IP can accelerate the implementation of PCI Express functionality in your designs.

I.2 PCI Express Overview

PCI Express provides low pin count, high reliability, and high-speed data transfer at rates of 2.5 Gbps and up, for serial links on backplanes and printed wiring boards.

I.2.1 PCI Express System Example: PC Motherboard Based System

In the example PCI Express system shown in [Figure I-1](#), the dashed lines represent PCI Express Links, the purple boxes indicate plug-in cards, and the other boxes are components found on a system card.

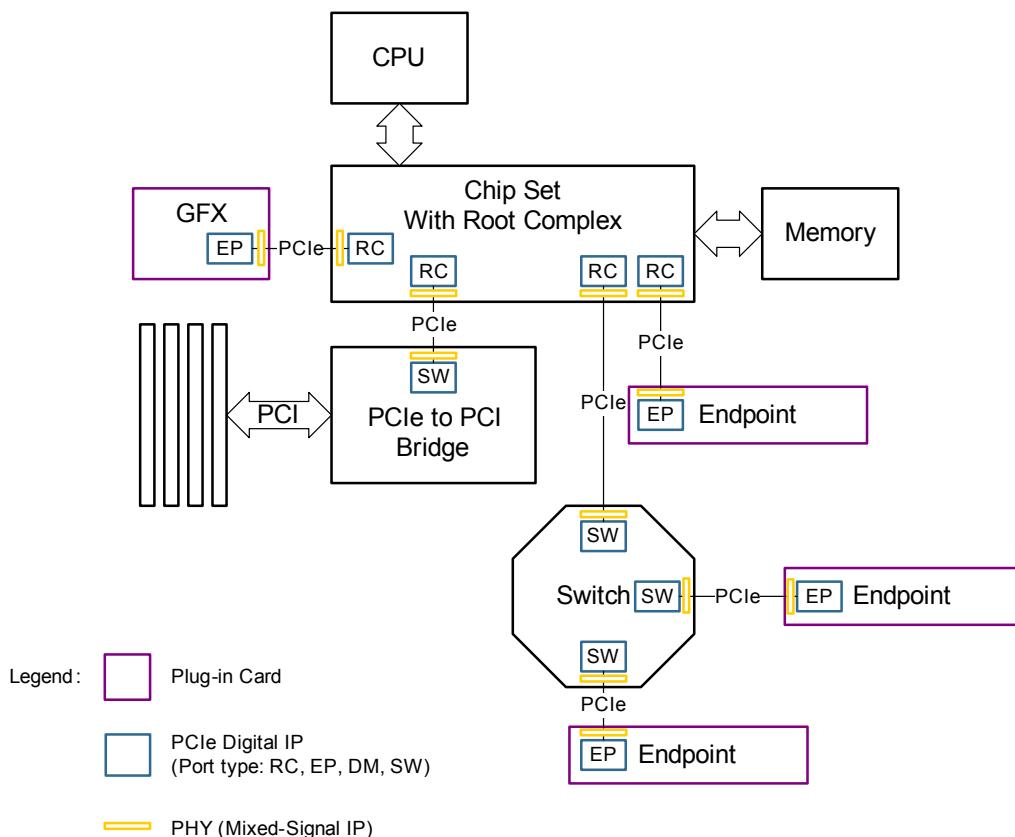
The blue and yellow boxes represent PCI Express IP, which may be digital (blue) or mixed-signal (yellow):

- ❖ **Root Port (RC):** Initializes and manages the PCI Express fabric
 - ❖ **Switch Port (SW):** Routes data between multiple PCI Express Links
 - ❖ **Endpoint Port (EP):** Associated with I/O devices and terminate a PCI express hierarchy
- or mixed-signal (yellow):
- ❖ **PHY:** Performs analog/digital conversion and serialization/deserialization

You can choose the throughput of each PCI Express Link by selecting the number of Lanes.

The PCI Express IP handles the protocol-related functions such as Link initialization, error recovery, power management, and data buffering.

Figure I-1 PC Motherboard Based System

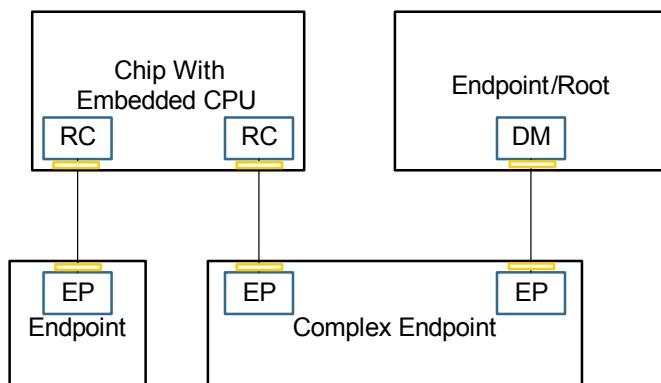


I.2.2 PCI Express System Example: Chip-to-Chip System

In the example shown in [Figure I-2](#), all circuits are on a single printed wiring board.

The Complex Endpoint chip includes two independent PCI Express Links. The Endpoint/Root chip includes an additional type of digital PCI Express IP, Dual Mode (DM), which can operate as a Root Port, as shown in [Figure I-2](#). The DM digital IP can also operate as an Endpoint; for example, when plugged into a PC motherboard slot.

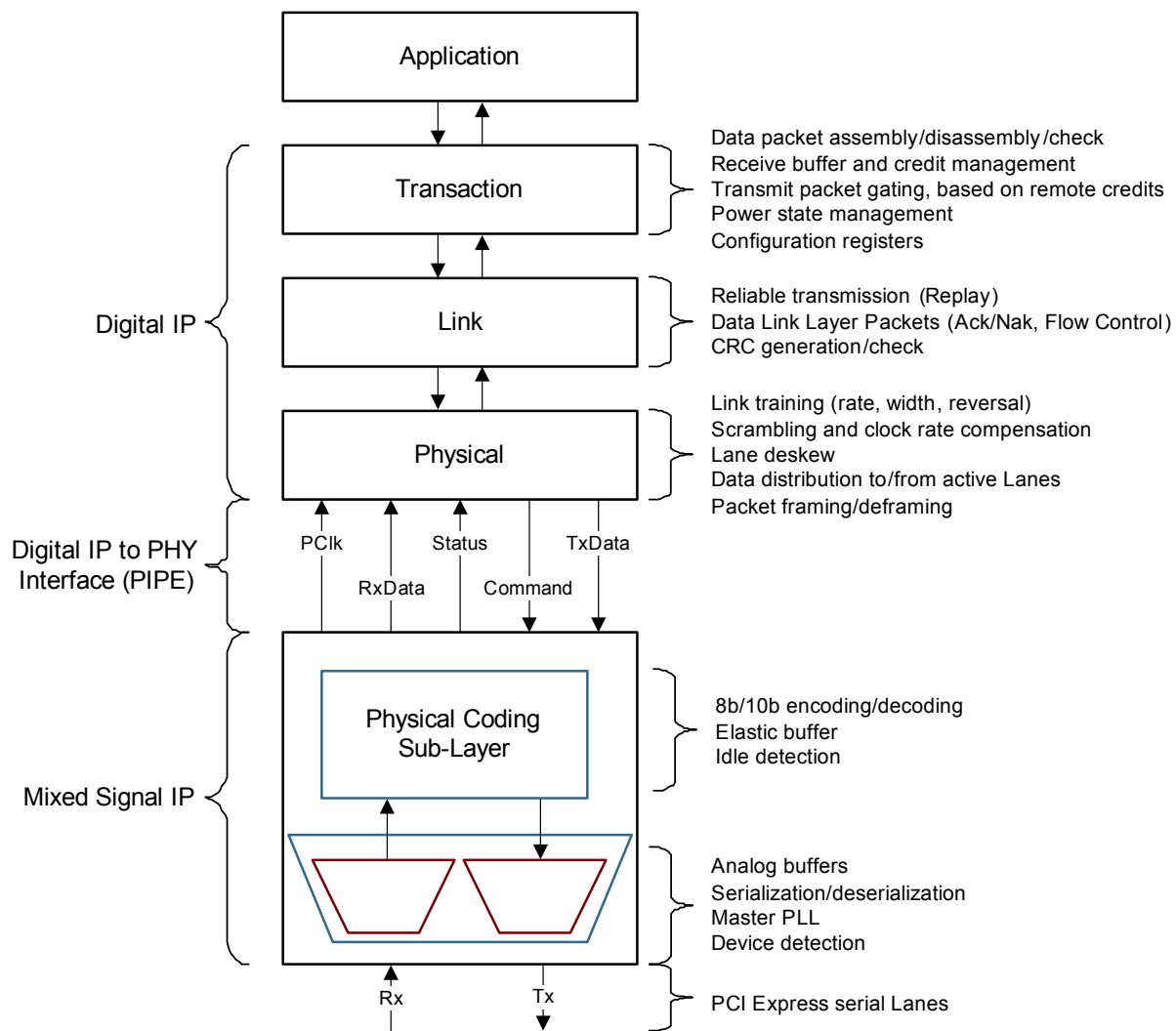
Figure I-2 Chip-to-Chip System



I.2.3 PCI Express Protocol Stack

The base PCI Express protocol stack is common to all of the IP shown in the examples.

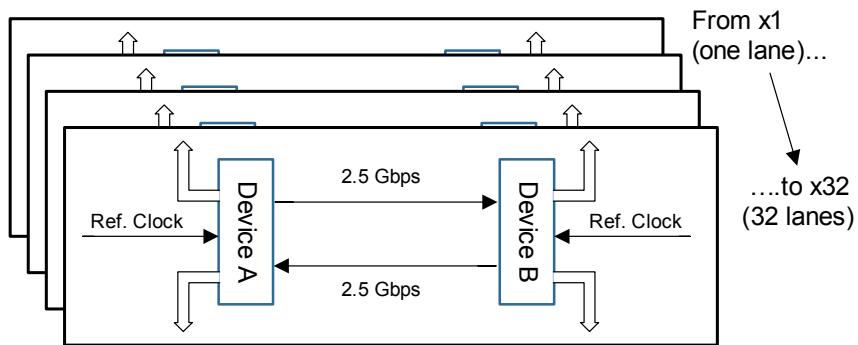
PCI Express IP hides most of the complexity of the protocol from your application logic, as shown in [Figure I-3](#).

Figure I-3 PCI Express Protocol Stack

I.2.4 PCI Express Links

As shown in [Figure I-4](#), each PCI Express Link:

- ❖ Is dual unidirectional with no sideband signals
- ❖ Is serial, with differential signaling
- ❖ Includes embedded clocking
- ❖ Operates at a scalable frequency (2.5 Gbps initially)
- ❖ Can be implemented in low-voltage silicon at 0.25u and smaller

Figure I-4 PCI Express Link

Data is transferred in packets, which include an address and a variable size data payload.

Reliability is provided by character checks, format checks, CRC code, automatic retransmission in case of error, and exchange of buffer space information, in the form of credits.

A handshake protocol to power down an idle Link is included, as well as Messages to handle interrupts, error reporting, and Hot-Plug events.

Configuration registers are available to customize Link behavior.

Wide Ports automatically configure to narrower Ports, as required.

Quality of service and isochronous traffic are supported through optional virtual channels (VCs).

I.2.5 Digital IP to PHY Interface (PIPE)

Transmit and receive data, as well as status and control, are transferred between the digital and mixed-signal IP on the PHY Interface for PCI Express (PIPE).

There are two standard options for transferring data across the PIPE:

- ❖ One byte per clock cycle per PCI Express Lane. In this case, the interface and the digital IP operate at 250 MHz.
- ❖ Two bytes per clock cycle per PCI Express Lane. In this case, the interface and the digital IP operate at 125 MHz.

I.3 Selecting PCI Express IP for Your Application

To add a PCI Express port to your chip you must:

- ❖ Select the Lane width
- ❖ Select mixed-signal IP including PIPE frequency and optional features
- ❖ Select digital IP type (RC, EP, DM, or SW) and optional features

I.3.1 Selecting the Lane Width

An N Lane PCI Express Port provides $N \times 2.5$ Gbps of raw throughput in each direction. Because of 8b10b encoding, packet payload size, and Link overhead, the actual throughput varies. [Table I-1](#) shows some example throughput values based on Lane width and packet payload size.



The values in [Table I-1](#) are a selection of reasonable examples. In a future application note, we will explain in detail how to calculate these numbers for your application.

Table I-1 PCI Express Link Throughput Examples

Lane Width	Real Throughput Examples (Gbps)			
	Packet Payload Size in Bytes			
	16	128	256	4096
x1	0.5	1.7	1.7	2.0
x4	2.0	6.8	7.0	7.8
x8	4.0	13.7	14.0	15.7
x16	7.9	27.4	28.0	31.4

I.3.2 Selecting Mixed-Signal PCI Express IP (PHY)

Mixed-signal IP is generally sold as a “hard macro”, which is tailored to your chip manufacturer’s process. Both the PCI Express and PIPE interfaces are standard. You can usually choose:

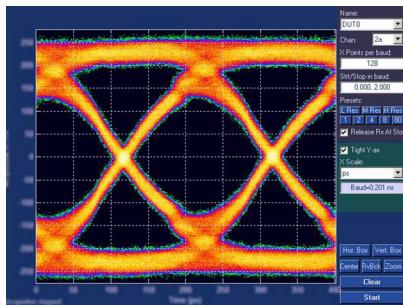
- ❖ The number of PCI Express Lanes
- ❖ Either a one-byte or two-byte PIPE. As explained above, you will determine the PIPE configuration and digital IP operating frequency when you make this selection.

In addition to these standard features, your PHY vendor may offer additional features, such as:

- ❖ Lower cost ICs with smaller die size (some PHYs are 50% smaller than others)
- ❖ Better performance margin (some have twice the sensitivity and jitter margin)
- ❖ Yield and reliability
- ❖ Built-in diagnostics and system test

For example, the Synopsys PCI Express PHY includes unique built-in diagnostics, which provide on-chip visibility into the actual performance of your 2.5-Gbps Links. [Figure I-5](#) shows actual scope data from the Synopsys PHY.

Figure I-5 Signal Display from Synopsys PCI Express PHY



I.3.3 Selecting Digital PCI Express IP Types

I.3.3.1 Digital IP for Basic Applications

Based on your application, operating frequency and Lane width, you may select PCI Express digital IP optimized for your application. [Table I-2](#) lists several examples of the range of Synopsys Endpoint (EP), Switch Port (SW), Root Complex Port (RC), and Dual Mode (DM) digital IP available to you.

Table I-2 Synopsys Digital Implementation IP Examples

IP Type Example (EP, SW, RC, DM)	Explanation	Comments
32-bit optimized x1	32-bit datapath, one Lane (x1). One Lane only operation allows extra multi-Lane logic to be removed for lower gate count and power.	Supports a single Lane (x1) with: <ul style="list-style-type: none"> • 8-bit PHY interface at 250 MHz, or • 16-bit PHY interface at 125 MHz.
32-bit	32-bit datapath, one Lane (x1) to four Lanes (x4)	<ul style="list-style-type: none"> • x1 to x4 with 8-bit PHY interface at 250 MHz, or • x1 to x2 with 16-bit PHY interface at 125 MHz
64-bit	64-bit datapath, one Lane (x1) to eight Lanes (x8)	<ul style="list-style-type: none"> • x1 to x8 with 8-bit PHY interface at 250 MHz, or • x1 to x2 with 16-bit PHY interface at 125 MHz
128-8	128-bit datapath, one Lane (x1) to 8 Lanes (x8)	Supports 16-bit PHY interface at 125 MHz
128-16	128-bit datapath, one Lane (x1) to 16 Lanes (x16)	Supports 8-bit PHY interface at 250 MHz

You can trade off gate count, operating frequency, and power for throughput.

I.3.3.2 Digital IP for Advanced Applications: Overview

It is relatively simple to select between Root, Switch, Endpoint, and Dual Mode Ports if your application fits one of the examples shown at the beginning of this application note.

However, if you have an advanced application, you may want to know more about the differences. The following sections summarize the differences between the types of digital IP; refer to the DesignWare IP product user/reference manuals for complete information.

I.3.3.3 Digital IP: Upstream vs. Downstream Differences

A PCI Express hierarchy contains one or more PCI Express Links, and includes a Root Port, optional Switch Ports, and Endpoint Ports.

Each Link in the hierarchy must include exactly one “downstream” facing Port and one “upstream” facing Port.

- ❖ Root Ports are always downstream Ports.
- ❖ Endpoint Ports are always upstream Ports.
- ❖ Switch Ports may be configured to be either upstream or downstream – see the PC system example at the beginning of this application note. In a Switch device, the Switch Port closest to the Root is always an upstream Port; all the other Switch Ports are downstream Ports.

Why does this matter?

- ❖ When the digital IP initializes the PCI Express Link, the upstream and downstream Ports use slightly different protocols to automatically configure the Link for width, and for Lane and data reversal.
- ❖ During idle times, the digital IP can autonomously transition the Link into a deep low-power state. This transition is requested by the downstream Port, and approved by the upstream Port.

I.3.3.4 Digital IP: Configuration Registers Differences

Each instance of PCI Express IP contains a set of configuration registers:

- ❖ Root and Switch Ports contain “Type 1” configuration registers.
- ❖ Endpoint Ports contain “Type 0” configuration registers.

Type 0 configuration registers:

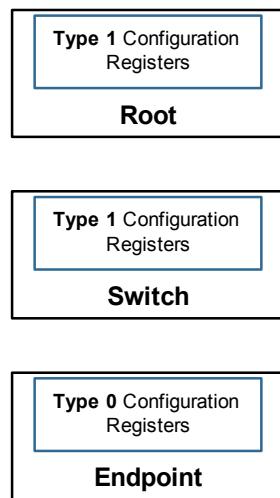
- ❖ Indicate to system software that this device is the “end” of a PCI Express hierarchy.
- ❖ Contain a full set of Base Address Registers (BARs) that help you filter and address map received packets.

Type 1 configuration registers:

- ❖ Indicate to system software that there are more devices to discover beyond this device.
- ❖ Contain limit registers to assist in packet routing to other devices.
- ❖ Contain only minimal BARs for packets mapped to the Type 1 device itself.

Some other configuration register differences:

- ❖ Endpoint devices may contain multiple copies of the configuration registers. This feature is used to implement multi-function devices.
- ❖ Root Ports include extra registers to summarize error status for a PCI Express hierarchy.
- ❖ Root and Switch Ports contain registers to manage Hot-Plug events.

Figure I-6 Configuration Space Types

I.3.3.5 Digital IP: Configuration Transaction Differences

Configuration transactions can only be initiated by Root Ports, and can only be responded to by Endpoint ports and upstream Ports.

Configuration transactions are used to:

- ❖ Determine the topology of a PCI Express hierarchy.
- ❖ Initialize the configuration registers after a PCI Express Link is initialized. Many values can be initialized in hardware; for example, by configuring the IP using the Synopsys coreConsultant tool. However, other values, such as memory space enable and base addresses, must be initialized through configuration transactions.
- ❖ Change the power state of a device.
- ❖ Read error report registers.

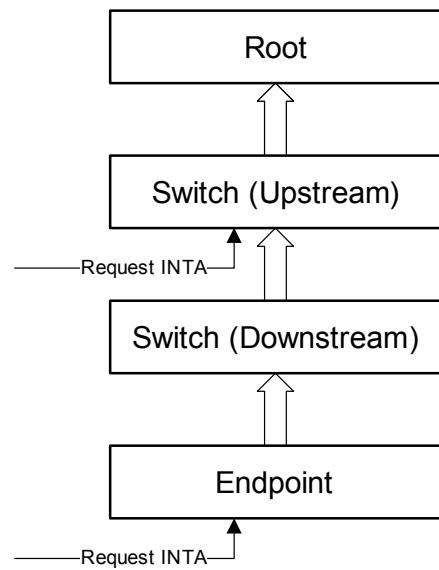
I.3.3.6 Digital IP: Interrupt and Error Message Differences

As described in detail in another application note, PCI Express devices emulate PCI interrupt wires (INTA, INTB, INTC, and INTD) by sending PCI Express Messages towards the Root Port, as shown in [Figure I-7](#). These legacy PCI interrupt Messages have the following characteristics:

- ❖ Endpoints and upstream facing Switch Ports may initiate legacy interrupt Messages.
- ❖ Downstream and upstream facing Switch Ports pass legacy interrupt Messages through to Switch core logic.
- ❖ Root ports may receive legacy interrupt Messages.

Other types of interrupt Messages (MSI, MSI-X) do not have these restrictions.

Error Messages are sent by PCI Express devices in response to Link errors. Endpoints initiate error Messages, Switch Ports initiate and pass along error Messages, and Root Ports receive error Messages.

Figure I-7 Interrupt Message Transfer

I.4 Implementing PCI Express in Your Design

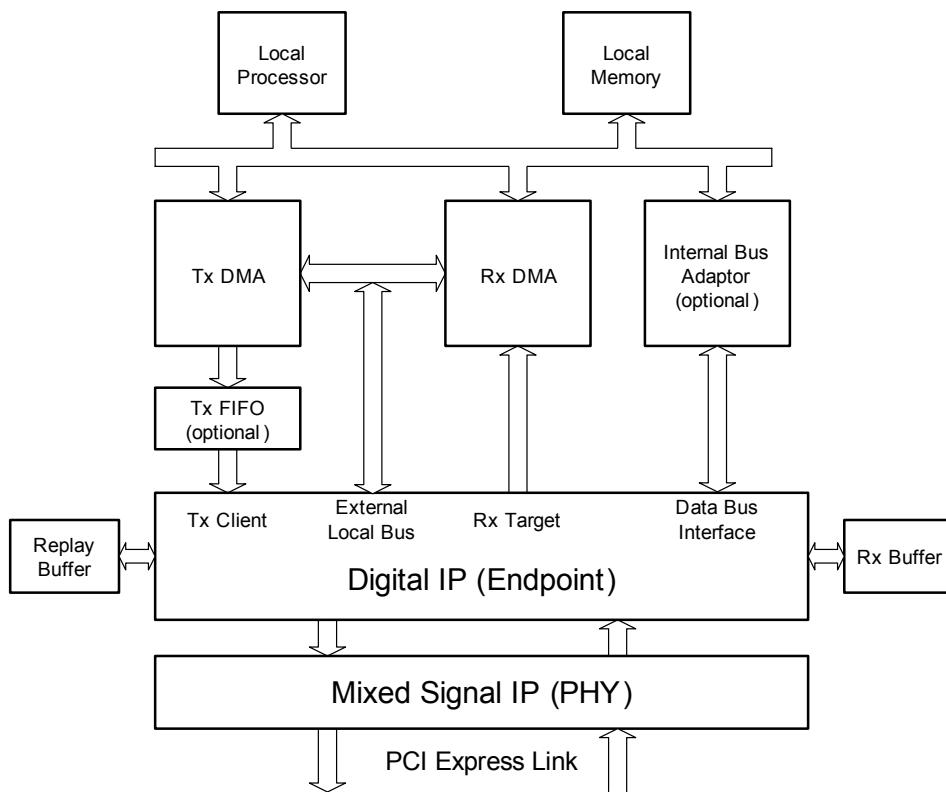
[Figure I-8](#) shows the major features of a simple Endpoint design. refer to the DesignWare PCI Express IP user/reference manuals for details.

The replay buffer is single-port RAM; the receive (Rx) buffer is a dual-port RAM. All logic to manage these buffers is included in the Endpoint IP. For the receive buffer, you may choose store-and-forward, cut-through, or bypass (no buffer) packet storage.

The transmit FIFO is optional – if your transmit DMA can continuously supply the data for an entire packet, the FIFO is not necessary.

The Internal Bus Adaptor is optional – it is only required if you wish to update the “read-only” fields in the IP configuration registers before Link communication begins. As an alternative, all of these fields can be configured at synthesis time with the Synopsys coreConsultant tool.

Figure I-8 Endpoint Block Diagram



The Digital IP interfaces shown in [Figure I-8](#) may be configured to fit your application:

- ❖ The transmit (Tx) client builds packets for you from data, address, and other attributes presented by your logic. It also gates your packets according to the PCI Express rules for buffer space (credits) at the other end of the Link. You can configure the IP to have up to three transmit clients.
- ❖ The receive (Rx) target interface disassembles validated packets and presents them to you as data, address, and other attributes.

- ❖ Use of the external local bus interface (ELBI) is optional. The ELBI provides a convenient way for the processor at the other end of the Link to read and write your local application control registers. No additional application hardware is required in this case. The Endpoint IP can be configured to map these register read/writes to the ELBI.
- ❖ The data bus interface (DBI) provides local access to the Endpoint configuration registers. Use of the DBI is also optional, as discussed above with respect to the Internal Bus Adaptor.

I.5 Summary

PCI Express is a robust interface and selecting the right IP can help solve the complexities of implementing the PCI Express protocol in your designs and accelerate your development process. The DesignWare IP for PCI Express is silicon-proven in customer designs and is the industry standard, powering the PCI-SIG protocol test card, and was the first PCI Express IP to pass the compliance test.

The DesignWare IP has gone through extensive interoperability testing with third party PHYs, verification IP, and hardware. By providing a complete solution for PCI Express including digital controllers, verification IP and mixed-signal PHY IP, Synopsys helps lower your integration risk and overall deployment costs, while saving you significant time and effort.

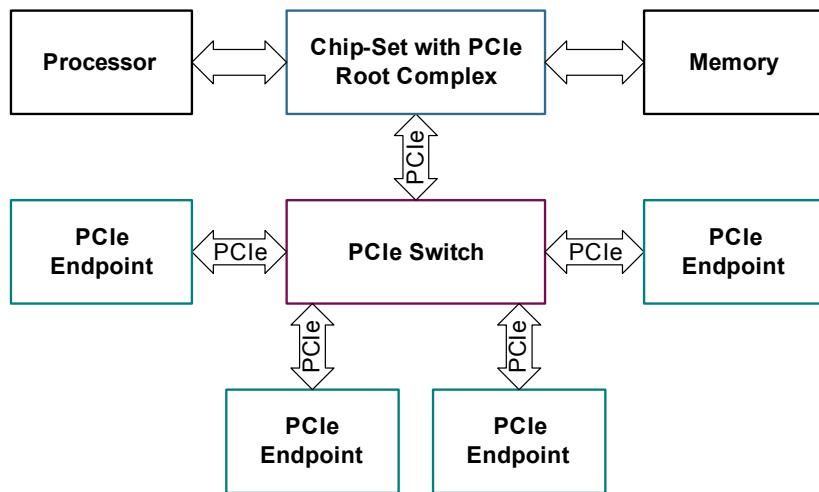
App Note: Introduction to PCIe Switches

J.1 Introduction

This application note is intended for beginning and intermediate PCI Express users. You can read this application note without any prerequisites, but you may want to read Chapter 2, “Selecting PCI Express IP for Your Design” in the PCI Express Application Notes for additional background information. Please refer to the PCI Express User and Reference Manuals for detailed information about the Switch core.

PCI Express provides a high speed, low-pin count, serial, chip-to-chip interface. Because PCI Express is a point-to-point interface, switches are used for fan-out. [Figure J-1](#) shows a switch connecting four endpoints to a processor and chip-set.

Figure J-1 A PCIe Switch Chip Provides Fan-Out to Multiple Endpoints



You can build a PCI Express switch by adding your switch core logic to pre-verified, configurable DesignWare digital and mixed-signal IP.

This application note will help you understand:

- ❖ The architecture of a PCI Express switch
- ❖ The functionality in the digital and mixed signal IP, and in your switch application logic
- ❖ The amount of work required to design a switch

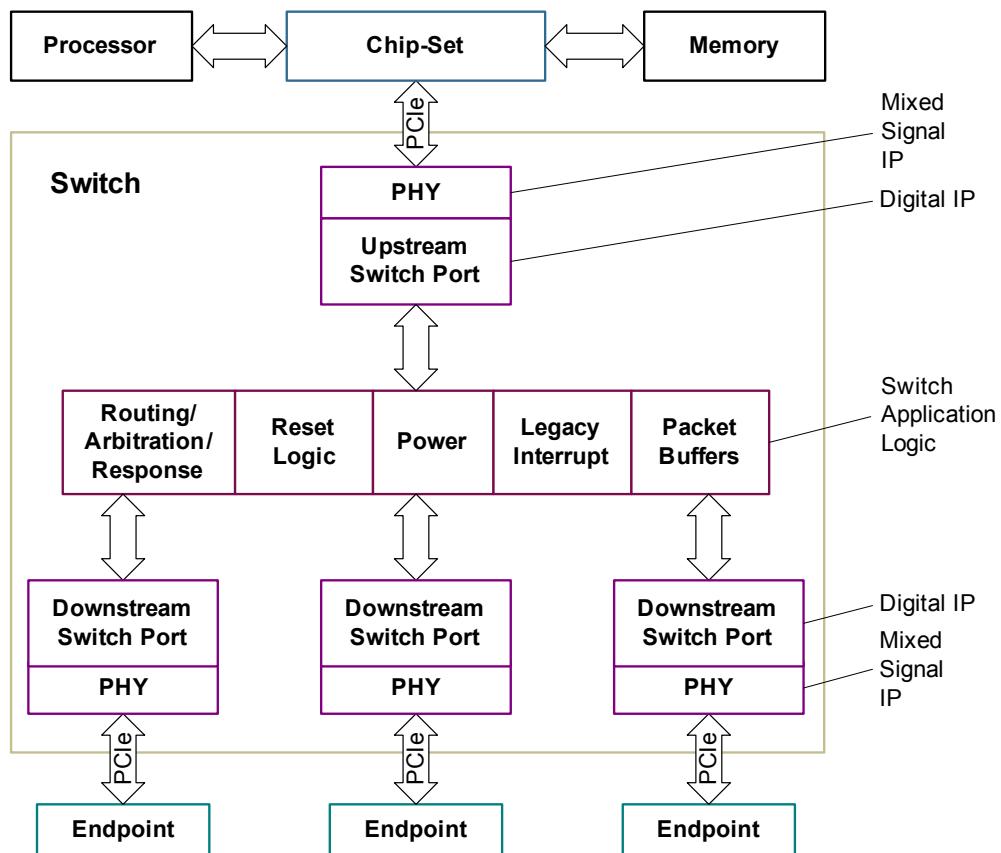
Topics include:

- ❖ Summary of switch architecture
- ❖ Mixed signal IP in a switch
- ❖ Digital IP in a switch
- ❖ Functionality of your switch application logic
- ❖ Summary of advanced and non-standard switch features

J.2 Switch Architecture

A switch typically includes your switch application logic, plus configurable PCI Express digital and mixed signal IP, as shown in [Figure J-2](#).

Figure J-2 A PCIe Switch Chip Includes User logic and IP Blocks



The digital IP at the top of the diagram (closest to the processor) is the “upstream,” or “upstream facing” switch port. The other instances (1 to 32) of digital IP are “downstream” or “downstream facing” ports.

Each of the PCIe links shown may be of different widths. For example, the upper link could be x4 (4 lanes wide), while the other links are x1. Each unit of width provides 2.5 GBits/second of raw throughput.

The switch application logic includes:

- ❖ Routing, arbitration, and response logic for PCI Express packets
- ❖ Packet buffering
- ❖ Power management
- ❖ Legacy interrupt logic
- ❖ Reset logic

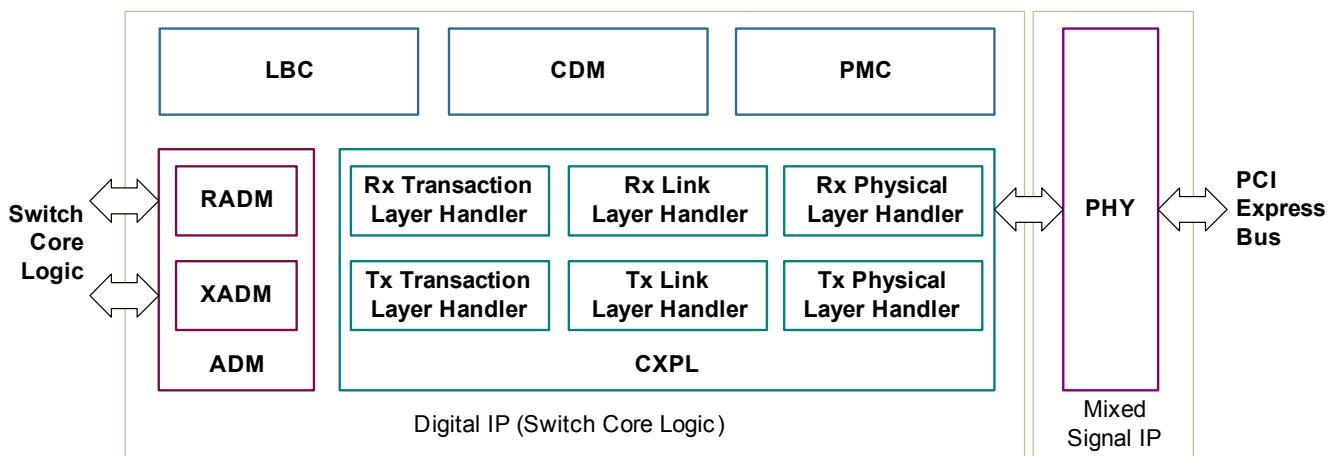
J.3 Digital IP

The DesignWare switch port digital IP (see [Figure J-3](#)) hides most of the PCI Express protocol from you. The IP includes the physical, link and transaction protocol layers, and performs all required character and packet-level encoding, decoding, error checking, automatic retransmission, credit and remote buffer space checking, packet building, power management, and configuration registers.

You customize DesignWare digital IP to your application with the coreConsultant GUI.

When you instantiate the IP in your design, an input pin is available to configure the core as either an upstream or a downstream port.

Figure J-3 PCI Express Digital (Switch Core) and Mixed Signal IP (PHY)



J.4 Mixed Signal IP

The mixed signal IP in the switch (see [Figure J-3](#)) is a standard PCI Express PHY. These circuits are designed for a particular manufacturing process and perform PCIe differential line reception/transmission, serialization/deserialization, idle detection, receiver detection, clock generation, and power controls.

The DesignWare PHYs also include on-board diagnostics.

J.5 Switch Application Logic Summary

This section describes the tasks to be performed by your switch application logic, and the support for those tasks built into the Synopsys IP.

J.5.1 Routing, Arbitration, and Response Logic

Routing is the steering of packets from one switch port to another. Routing depends on the type and address fields in received packets. The Synopsys IP disassembles valid packets and presents these fields to your switch application logic.

Arbitration is the selection of packets to be routed.

Response logic generates replies (completions) when a received packet targets a register inside the switch. The register might be located in your switch application logic, or inside one of the downstream digital IP cores.

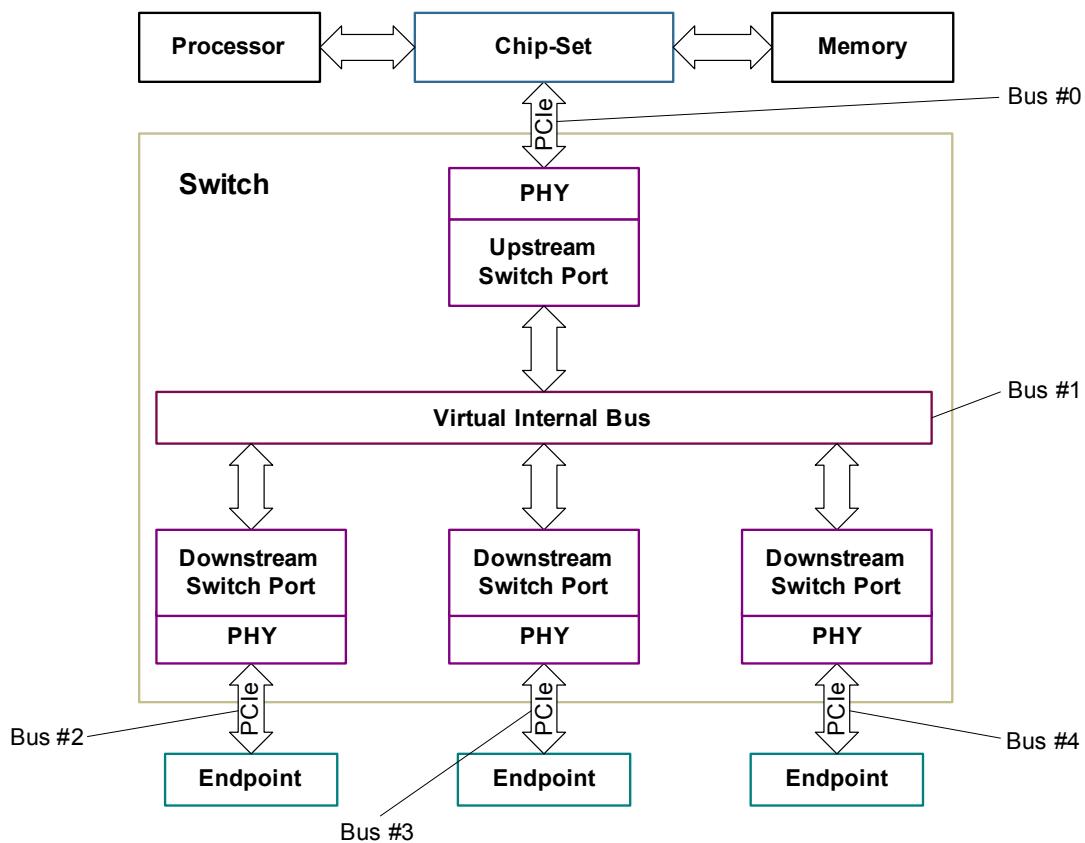
J.5.1.1 Configuration Packet Routing and Response

Configuration packets are sourced by the system processor and therefore are only received on the upstream switch port. These packets are used to read or write configuration registers in the digital IP blocks. The registers may be located in:

- ❖ The upstream switch port
- ❖ Any of the downstream switch ports
- ❖ In digital IP in an endpoint beyond the switch

Configuration packets (“configuration requests”) are routed by bus, device, and function number fields in the request packet address fields.

- ❖ Bus numbers are assigned by host software (see [Figure J-4](#)). Note that there is a “virtual bus” inside the switch core logic.
- ❖ Device numbers are assigned by you when you instantiate your downstream switch ports. This number will be used by your switch core logic when it responds to configuration transactions directed at the downstream switch ports.
- ❖ The function number is always zero for switch ports. Configuration transactions which pass through the switch may have non-zero function numbers

Figure J-4 An Example of Bus Numbers Assigned by Host Software

What are the most important fields in a configuration request packet?

The configuration transaction is directed to the correct register via fields in the received packet. These fields are presented to your application logic by the upstream switch port. Interesting fields are:

- ❖ Type - specifies if the request is read or write, and whether the request is directed at the upstream port registers (Type 0) or at other PCIe ports (Type 1)
- ❖ Address - gives you the bus, device, and register numbers for the request
- ❖ First BE - byte enables for configuration write requests

How do I handle Type 0 configuration packets in the switch core logic?

You will never receive Type 0 configuration requests in the switch core logic. These will be handled by the upstream port IP core.

How do I decide if a Type 1 configuration packet is directed at one of the downstream switch ports?

You examine the bus number in the packet address field. To the host software, the downstream switch ports appear as devices on a numbered bus. This bus number is assigned by the host software, and is displayed on the secondary bus number pins of the upstream switch port; you can compare it to the bus number in the configuration packet.

Note that you must use the upstream port's secondary bus pins, and *not* the downstream port's primary bus pins, which may be temporarily "out of sync" during bus number configuration.

How do I know to which downstream facing port a configuration packet is directed?

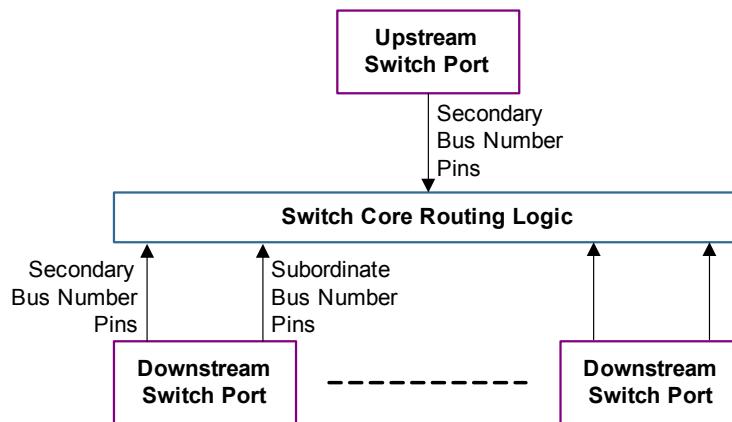
You examine the device number in the packet address field. When you build your switch design, you assign each downstream switch port a unique device number.

How do I direct a configuration request if the bus number does not match the switch's internal bus number?

Each of the downstream switch ports will display the range of bus numbers to be found "below" that port on the Secondary and Subordinate Bus Number pins (see [Figure J-5](#)). You can use this range to transmit the configuration request through the correct downstream port.

Note, that if the bus number in the configuration request matches the Secondary Bus Number, then you must change the configuration request type from 1 to 0 before you transmit the request via the downstream port.

Figure J-5 Each Switch Port Displays the Downstream Bus Numbers



How do I respond to a configuration request directed at a downstream port?

You use the downstream port's DBI interface to read or write the requested configuration register. You then send the response via the upstream switch port, which will build the response (completion) packet for you from the packet fields you supply. The fields include:

- ❖ Traffic Class, Attributes, BCM, Lower Address are all 0
- ❖ Requestor ID and TAG are copied from the original configuration request
- ❖ Length is one (one DWORD = 4 bytes)
- ❖ Completer ID is copied from the configuration request's address field (Bus, Device, and Function number)
- ❖ Byte count is four
- ❖ Completion Status is zero for a successful completion, and one for unsuccessful (see below for definition)
- ❖ Data is 32 bits (only for a configuration read request)

What happens if the bus and device numbers of the configuration request do not match any of the internal ports and do not match the range of any of the downstream ports beyond the switch?

This is a system software error. You should respond with an Unsupported Request status using the upstream switch port's Application Error Report pins.

Note that this "error" will occur intentionally during system initialization, when the host software checks to see how many downstream ports are present on the switch.

Do I have to check the function number in the configuration request address?

You should check the function number for configuration requests targeted at one of the downstream switch ports. The function number in this case should be zero; if not, you should respond as described above with an Unsupported Request status.

How does my switch core logic "capture" bus numbers for downstream switch ports?

Whenever your switch core logic executes a configuration register write, you must also write the bus number from the configuration request into the downstream switch ports' primary bus number register. This is a simple transaction on the port's DBI interface.

J.5.1.2 Memory and I/O Request Packet Routing

Memory and I/O packets are usually routed from one switch port to another. In some cases you will have memory mapped registers in your switch core.

Memory and I/O requests are routed in a similar manner as configuration requests, except addresses are used instead of bus and device numbers. Each downstream switch port presents the range of addresses "below" it on the corresponding pins.

In contrast to configuration packets, memory and I/O packets may be received from any port and sent to any other port. If the target address does not match the range of addresses below any downstream port, then you must send the packet to the upstream port.

The upstream switch port will reject any memory or I/O mapped packet which is not in the range of addresses programmed into its registers. Note that there might be "holes" in this memory space that can only be detected when your switch core logic checks the downstream ports' address ranges.

It is possible to have memory or I/O mapped registers in your switch core. When a switch port (usually upstream) receives such a request, it can handle this for you via the port's local bus interface (ELBI). The switch port IP will access the register, perform the required read or write, and form and transmit any required completion packet. This access is restricted to single DWORDS. The memory range for these registers is specified by a Base Address Register in the switch port.

J.5.1.3 Completion Packet Routing

Completion (response) packets are routed by the packet's Requestor ID field, which specifies the bus and device number of the original requestor. For completions received from a downstream port, the bus and device numbers might not be in the range of any of the downstream ports. If so, you must send the completion packet to the upstream port.

On the other hand, you might receive a completion packet from the upstream port whose bus number does not map into any of the downstream port bus ranges. In that case, you should discard the completion packet, and report an "Unexpected Completion" error using the upstream ports application error reporting pins.

J.5.1.4 Message Routing

The switch core response to messages depends on the type. This is explained in "[Error Messages and Error Detection](#)" and "[Power Management](#)".

J.5.1.5 Cross Bar vs. Bus Routing

You also have to choose how to physically transfer packets between switch ports. Two obvious choices are a common bus or a crossbar. A common bus may be appropriate if all or most traffic is downstream to upstream, or vice versa.

J.5.1.6 Preservation of Reserved Packet Bits

Certain packet header bits are marked as "reserved," and therefore are always zero in the current (1.1) version of the PCI Express specification. However, a switch is required to transfer these bits *without modification*. This is to ensure that your switch design is compatible with future versions of PCIe.

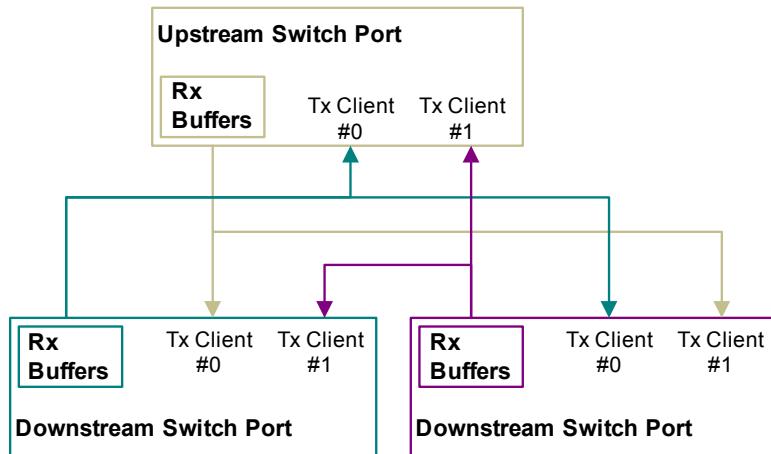
J.5.2 Packet Buffering

Because a switch port's transmit interface may be busy, or unable to transmit a packet immediately, there is a requirement to buffer received packet data. Buffering is an advanced topic requiring considerable analysis and modelling. Three approaches are presented here as examples:

J.5.2.1 Receive Side Buffering

This is the simplest approach (see [Figure J-6](#)). You use the available built-in receive buffers in each switch port to store packets when the destination port is busy. You can reduce latency by using cut-through buffering, which allows the IP core to drive a received packet to your application logic before the packet is fully verified. This allows you to examine the packet's type and destination address, and immediately begin sending it to its destination. You can also stop the packet if the destination is busy, or cannot accept the packet.

Figure J-6 Packet Buffering with Digital IP Receive Buffers



What are the advantages of receive buffering?

Receive buffering uses the IP's receive buffers and logic, which includes circuitry for PCI Express ordering rules. The receive buffers also handle all PCI Express credit (buffer space) reporting requirements, and arbitrate between packet types and virtual channels. No additional transmit buffers are required.

The Synopsys digital IP transmit clients arbitrate between packet sources for you.

What are the disadvantages of receive buffering?

If you halt a packet from an input port, all other packets from that port may be blocked. If the blocking lasts more than a short time, you might violate PCI Express protocol rules, which require certain packets to pass others (to avoid deadlock).

J.5.2.2 Transmit Side Buffering

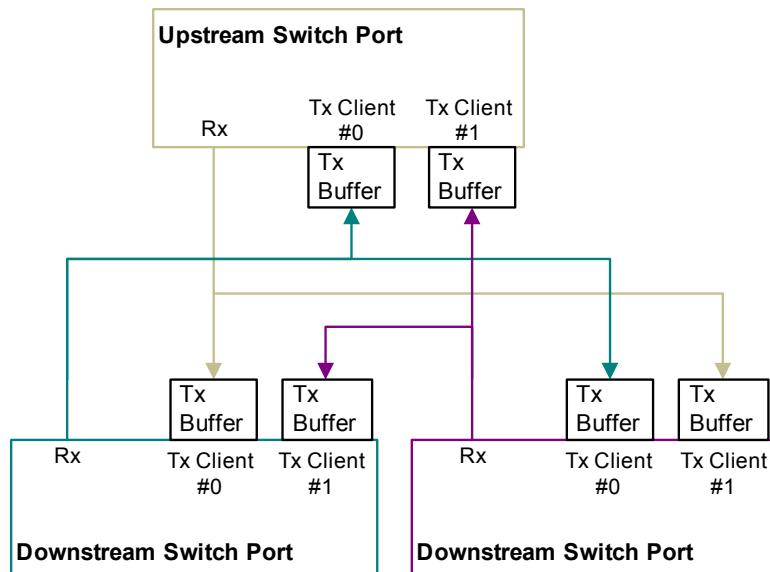
If you choose transmit buffering, you use packet buffers at each switch port's transmit client. There is generally one transmit buffer per packet source at each switch port. Each of these buffers in turn includes a memory area for each packet type (posted, non-posted, completion).

What are the advantages of transmit buffering?

If you provide one transmit buffer per packet type per switch port, you can allow smaller packets to pass blocked larger packets.

In the scheme shown in [Figure J-7](#), the digital IP performs arbitration between receive packet ports.

Figure J-7 Transmit Buffering Prevents Head of Line Blocking



What are the disadvantages of output buffering?

You must add buffering and buffering logic, including logic for PCI Express ordering rules, and deletion of partially received bad packets.

In addition, you must send buffer space update pulses from the transmit buffer back to the receive port IP. The IP will use the pulses to send credit packets for you, as required.

J.5.2.3 Switch Core Buffering

Switch core buffering is the most complex, but most powerful packet storage scheme.

You can provide very large amounts of storage, combine on-chip and off-chip memory, and reallocate memory freely at system startup time to handle different applications.

You can also reallocate storage during packet transfer if you are careful with PCI Express credit management.

J.5.3 Error Messages and Error Detection

There are three cases.

1. As mentioned above, you may detect packet errors in your switch logic. These are easily reported via the application error reporting pins on the upstream facing switch port. Errors detected by the upstream facing port are handled internally by the port.
2. You may receive error message packets from one of the downstream switch ports. These are easily recognized because the port disassembles the packet header for you. A bit in the upstream port's Bridge Control Register determines whether your switch core logic should drop these error message packets, or send them to the upstream port.
3. Finally, a downstream port may detect an error condition, and determine (based on internal register settings) that an error message should be transmitted upstream. A pin for each type of message (correctable, uncorrectable, fatal) is available.

J.5.4 Power Management

J.5.4.1 PME Turn Off Protocol

When a PME Turn off message is received from the upstream switch port, the core logic should transmit this message to each downstream port (following posted write ordering rules). At this point, the core logic must wait for a PME Turn OFF Ack message from the downstream ports. These message are discarded, but when every downstream port has responded, the upstream port should be signalled to send a PME Turn Off Ack. These operations are facilitated by switch port pins.

J.5.4.2 PME Event Messages

These messages are simply passed from downstream ports to the upstream port. It is also possible for your switch core logic to send a PME message upstream towards the host.

J.5.4.3 Set Slot Power Limit

Your switch core logic must detect configuration writes to the Slot Capabilities register of the downstream ports, and generate a Set Slot Power Limit message on the corresponding link. You must also generate this message if a link attached to a downstream port transitions to the link up (DL_Up) state.

You will be able to easily detect these configuration writes since your core logic handles the configuration accesses to the downstream port configuration registers.

J.5.4.4 Link Power Down States (L2/L3, L1, L0s)

The DesignWare cores automatically execute the PCI Express link power down protocol for you. However, the switch core logic is required to consolidate and transfer some power state information between upstream and downstream switch ports:

- ❖ Indicate to the upstream port that all downstream switch ports are in the L1 power down link state
- ❖ Indicate to the upstream port that all downstream switch ports are in the L0s power down link state
- ❖ Indicate to the downstream switch ports that the upstream switch port is in the L0s power down link state
- ❖ Indicate to the upstream port that a downstream port has started an exit from the L1 power down link state

J.5.4.5 D-state Packet Filtering

The host software may program each of the switch ports (actually, the associated link) into one of four PCI-compatible power states. The programmed power state is indicated on switch port output pins.

In states D1, D2, and D3, only configuration requests may be transmitted downstream; only messages and completions may be transmitted upstream. Any packets that violate these rules should be discarded, and an Unsupported Request is signalled, as previously explained.

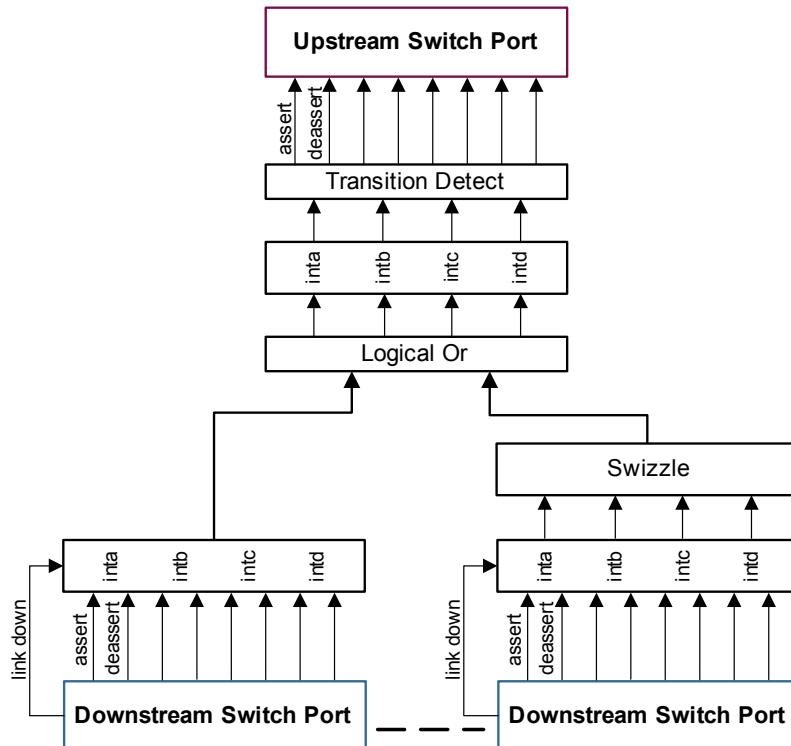
J.5.5 Legacy Interrupt Messages

Legacy interrupt messages are transmitted from PCIe endpoints toward the host. These messages are used by the switch core to build virtual wires to emulate the INTA through INTD wires used in PCI. Your switch core logic must keep track of these virtual wires for each downstream port. These downstream virtual wires are combined into upstream virtual wires in the switch core logic (see [Figure J-8](#)); any change in the state of the upstream virtual wires is sent to the root complex, via the upstream switch port.

Wires from the downstream ports are “swizzled” before being combined. This is a simple rotation (e.g., INTA -> INTB, INTB -> INTC, etc.) based on the downstream port’s device number. See the PCI Express specification (“INTx Interrupt Signalling”) for more information.

You also have to monitor the state of the downstream ports’ links. If any of them go to the DL_Down status, you have to deassert the virtual interrupt wires associated with that port.

Figure J-8 Legacy Interrupts are Combined in Your Switch Core Logic



J.5.5.1 Clock Gating

You can use the coreConsultant configuration tool to specify fine grain synthesis clock gating for the digital IP core. You can also shut off the clock to most of the IP core, when indicated by the clk_req_n output signal. Refer to the DesignWare PCI Express user/reference manuals for details.

J.5.5.2 Forward beacon/WAKE# upstream

PCI Express allows downstream devices to request power-up via a Beacon signal transmitted on the PCIe wires, or via the legacy PCI WAKE# wire. The DesignWare PCI Express core supports beacon detection and transmission in the mixed signal IP, and wake up signals in the digital IP.

J.5.6 Reset and Link Down

For a Switch, the following must cause a hot reset to be sent on all downstream ports:

- ❖ Setting the Secondary Bus Reset bit of the Bridge Control register of with the Upstream Port
- ❖ The Data Link Layer of the Upstream Port reporting DL_Down
- ❖ Receiving a hot reset on the Upstream Port

The switch cores provide input and output pins to help your switch core perform these operations.

J.5.7 Packet Arbitration

At each switch port's transmit interface, you must select from available packets.

Selecting from packets in different Virtual Channels (see below) is known as VC arbitration.

Selecting from packets in the same Virtual Channel but from different receive ports is known as port arbitration.

Both of these types of arbitration are fully described in the PCI Express specification along with optional and required features.

You may be able to use the built in VC and port arbitration in the switch port.

J.6 Advanced Switch Feature Summary

J.6.1 Digital IP Initialization

Each of the digital IP cores contains configuration register bits that are Read Only to host software. You can configure the reset values of these bits using the DesignWare coreConsultant GUI.

For greater application flexibility, you may want to load these read-only bits after reset, but before link communication begins. In this case, you may set the switch core's app_ltssm_enable input pin to logic zero at reset time. When you have written all of the configuration bits from your switch core logic into the digital IP, you should set the input pin to logic one; at this point link initialization will begin.

You should release the upstream port's app_ltssm_enable pin last, so that your switch will be ready to respond to received configuration requests.

J.6.2 Poisoned TLPs

Generally, your switch core logic should ignore the poisoned bit in packets, unless the packet is directed at a switch port, or at the switch core logic. Refer to the PCI Express specification for details.

J.6.3 Virtual Channels and Traffic Classes

Each of your switch ports may be configured in the DesignWare coreConstltant GUI with one to eight Virtual Channels (VCs).

A port will have a set of receive buffers for each Virtual Channel. You can choose to implement these in a single physical memory, if you use the buffer logic in the digital IP. Alternatively, you can implement these in your switch core logic or in transmit buffers as discussed above. Multiple virtual channels allow higher priority traffic to pass lower priority traffic.

The digital IP transmit clients will present you with information on the available buffer space at the other end of the link for each virtual channel. You can use this information plus traffic class and the PCI Express ordering rules to select packets for transmission.

Each packet has a traffic class field with a value between 0 and 7. System software programs each port with information to

- ❖ map physical virtual channels to logical virtual channels (VCID)
- ❖ map traffic classes to logical virtual channels

Note that each link (port) in your system may use a different number of VCs, a different set of VC IDs, and a different mapping of traffic classes to VC IDs

When packets from multiple VCs are available for transmission on the same port, the switch must arbitrate between them. Refer to the PCI Express specification for details of VC arbitration.

J.6.4 Hot Plug

Downstream switch ports may be associated with a "slot" into which users may plug or unplug cards with system power on. The PCI Express protocol includes support for this "hot-plug" activity via indicators, retention latches, interlocks, and attention buttons that are transformed by the switch port into status register bits and message requests.

The Synopsys digital IP includes the pins and registers required for hot plug support. The IP also signals your switch core when a message is required due to a change in hot plug state.

J.6.5 Locked Transactions

PCI Express switches must support locked transactions (from the upstream port) for compatibility with legacy software.

To support locked transactions, your switch logic must watch for Successful Completion Lock completion packets routed from one of the downstream ports to the upstream port. When this occurs, you must block all packets on Virtual Channel zero from any of the other downstream ports to the locked downstream port.

The lock is terminated when your core logic receives an Unlock message from the upstream port. This message should be sent to the locked port (or to all downstream ports). Pins are available for this function.

J.6.6 Ordering Rules

When you select packets for transmission, you must follow PCI Express ordering rules to preserve Producer Consumer Ordering as well as to prevent deadlock. Refer to the other PCIe application notes and the PCI Express specification for more information.

A note on message reception: You may have noticed that the switch cores indicate the reception of certain messages with dedicated pins. You also have the configuration option of receiving and decoding these messages yourself on the normal packet interface (RTRGT1). Whichever option you choose, remember to take ordering into account. For example, an interrupt message should always be forwarded upstream *after* any posted write packets received before the interrupt.

J.6.7 Deadlock Prevention

Some additions to the PCI Express specification are under consideration to prevent deadlock. For example, "Upstream Ports (Switches) acceptance of a posted or completion packet must not depend upon the transmission (on a downstream port) of a non-posted packet within the same virtual channel" (PCI Express 1.1 C13 Errata).

J.6.8 Bifurcation

You may have downstream pins on your switch chip that you would like to use in a flexible manner, e.g., a set of pins that can be either a single x8 PCIe link, or two independent x4 links. This is known as bifurcation, and is explained in the PCI Express (All Cores) Application Notes.

J.7 Non-Standard Switch Feature Summary

J.7.1 Non-transparent Switches

In a normal PCI Express system, the host software discovers and configures the entire system by traversing switches to find all endpoints in the system.

Switches that support this are known as “transparent” switches.

Software knows that a switch port is not the end of the PCI Express hierarchy when it reads the configuration type - a switch port always shows a Type 1 space.

If you replace one of the downstream switch ports with an endpoint, address translation logic, and another endpoint, you could connect two different PCI Express hierarchies together. Note that this is not allowed by the PCI Express specification, but can be made to work when you have some control of system software.

J.7.2 Reconfigurable Upstream / Downstream Ports

The DesignWare PCI Express ports may be configured via an input pin as upstream or downstream. You could take advantage of this to change a downstream port to an upstream port in a fail-safe multi-host system. Note that this would require some endpoints in your system to operate as an endpoint *or* a root port; this is possible with the DesignWare PCI Express Dual Mode core.

J.7.3 Application in Switches

You may wish to replace one switch port with your own logic to allow inclusion of applications in the switch chip. As mentioned above, you must have some control of the system software, since this is not legal in the PCI Express specification.

J.8 Conclusion

You can readily add fan-out to your PCI Express designs by combining available DesignWare digital and mixed-signal IP with your own switch core design.

K

App Note: Calculating PCI Express Throughput

This application note defines the throughput calculation with respect to the PCI Express core, and provides guidelines to enable you to meet your desired throughput by selecting the appropriate configuration parameters from those supplied with the DesignWare PCI Express core.

This application note describes the throughput calculation, parameters that impact the calculation (such as the maximum payload size selection), and matching-up the appropriate application interface to obtain the best system performance from the Synopsys PCI Express core.

Please refer to the PCI Express core user and reference manuals for detailed information about the Synopsys PCI Express core, or contact your local support center for additional information.

K.1 PCI Express Throughput

PCI Express bandwidth is 2.5 Gb/s (gigabits per second), per lane, when operating at the Gen 1 data rate. Because data is encoded using 8b10b encoding (encoding used with digital differential signaling to maintain a constant average DC bias point), the effective maximum throughput is 250 MB/s (megabytes per second), per lane, calculated as follows:

$$2.5 \text{ Gb} * 8\text{b}/10\text{b} = 2 \text{ Gb} * 1\text{B}/8\text{b} = 250 \text{ MB/sec per lane}$$

Throughput for other lane widths are shown in [Table K-1](#).

Table K-1 Theoretical Throughput vs. Lane

Lane Width	Throughput after 8b/10b (MB/s)
x1	250
x2	500
x4	1000
x8	2000
x16	4000

K.2 Configuring the PCI Express Core With Respect to Throughput

The PCI Express core is available in 32, 64, and 128 bit data path widths (32 optimized for a one lane application).

The PCI Express core supports two clock frequencies. For the FPGA environment, the clock from the PHY and the core clock can be configured as 125 MHz. For the ASIC, the core can also support the 250 MHz clock.

- ❖ 125 MHz => 16 bit PHY (one clock per two symbols)
- ❖ 250 MHz => 8 bit PHY (one clock per symbol)

Whether the clock is 125 MHz or 250 MHz, the throughput from the PIPE is the same, 250 MB/s per lane. The throughput to the application interface will be different, based on the clock frequency selected in the core.

[Table K-2](#) shows the throughput for 32, 64, and 128-bit data path widths configured with a 125 MHz clock. [Table K-3](#) shows the throughput for 32, 64, and 128-bit data path widths configured with a 250 MHz clock.

Table K-2 Throughput for Data Widths with a 125 MHz Clock

Data Path Width (Bits)	Max Throughput @ 125 MHz (MB/s)	Lane Width
32	250	x1
32	500	x2
64	1000	x4
128	2000	x8

Table K-3 Throughput for Data Widths with a 250 MHz Clock

Data Path Width (Bits)	Max Throughput @ 250 MHz (MB/s)	Lane Width
32	1000	x4
64	2000	x8
128	4000	x16

K.3 Effective Throughput

The PCI Express IP does not limit the throughput, but you must follow the PCIe protocol. The key protocol features are:

- ❖ 8b10b encoding at the physical layer. This takes away 20% of the raw bandwidth.
- ❖ Acknowledge and flow control updates at the data link layer (DLLPs). This takes away 1% to 5% of the remaining bandwidth.
- ❖ Packet overhead at the transaction layer. Your design choices have a great effect here:
 - ◆ Small packets may take away 75% of the bandwidth!
 - ◆ Large packets may take away as little as 1%.

K.3.1 Effective Throughput Calculation

[Figure K-1](#) identifies the TLP package.

Figure K-1 TLP Package



K.3.1.1 Packet Level: (Start and End, Link CRC, Header)

[Table K-4](#) summarizes throughput calculations based on a payload size from 16 to 4K bytes.

Table K-4 Effective Throughput

	Payload Bytes	Header Bytes	ECRC Bytes	Link Bytes	Calculation	Percent Throughput
Worst Packet	16	16	4	8	16/44	36%
Typical Packet	128	16	0	8	128/152	84%
Typical Packet	256	16	0	8	256/280	91%
Typical Packet	512	16	0	8	512/536	96%
Best Packet	4096	12	0	8	4096/4116	99%

**Note**

A 128-byte payload size yields about 67% of the net throughput. Increasing the payload size to 512 bytes increases the net throughput to 92%. Increasing the payload size from 512 bytes to 4096 bytes only contributes an increase of 8% in the net throughput, and the storage requirements are more than doubled. In addition, a large payload size may have an impact on performance due to re-transmission of TLPs. Therefore, 256-byte and 512-byte payload sizes are the most popular choices. Most chipsets support 128 -byte or 256-byte payload sizes, with 512 bytes gaining in popularity.

K.3.1.2 Link Layer: (Flow Control and ACK/NAK DLLPs)

DLLPs should be sent as often as required to avoid a negative impact on the TLP throughput. The core sends a pending DLLP if there is no competing TLP traffic.

Sending ACK/NAK and Update FC is controlled by timers. The core provides flexibility to fine-tune these as required. The default setup is minimal flow control (one per time-out) and minimal ACK/NAK device latencies. The link partners retry buffer (payload) size may require a change in the ACK/NAK/Update FC time-out to obtain optimal system latencies.

The core provides a feature to accumulate up to 255 ACKs before issuing an ACK. This feature offers-fine tuning of ACK frequency impact. The core transmits ACKs at fixed intervals, by default.

[Figure K-2](#) identifies the DLLP package.

Figure K-2 DLLP Package



[Table K-5](#) summarizes the throughput calculations.

Table K-5 Effective Throughput

Bytes	Result	Percent Throughput
Worst Packet One ACK plus one FC per 128 (8 packets of 16) bytes of data => 2 DLLPs per 8 data packets	$(8*44) / (8*44 + 2*8)$	95%
Typical Packet One ACK plus one FC per 1.4 (256 byte payload) bytes of data => 2 DLLPs per 1.4 packets	$1.4*280 / (1.4*280 + 2*8)$	96%
Best Packet One ACK plus one FC per 4096 bytes of data => 2 DLLP per data packet	$4116 / (4116 + 2*8)$	99%

K.3.2 Other Factors Impacting Throughput

Some other factors that impact throughput are 8b10b encoding, link packets, and data packet overhead. Buffer sizing should be accurate to avoid a harmful impact of larger ACK/NAK or Update FC latencies by the link partner. The core does a very effective autosizing of the buffers taking into consideration the impact of these parameters.

$$\text{Effective Throughput (ET)} = \text{Raw Throughput} * (\text{NL} \times 2.5 \text{ Gb})$$

Consideration of lane width and payload size results in the following:

Factor	Throughput
8b10b	*(80%)
Link Packet	*(95% to 99%)
Data Payload	*(36% to 99%)

[Table K-6](#) identifies lane width and payload vs. throughput.

Table K-6 Lane Width and Payload vs. Throughput

Real Throughput Gb/s vs. Data Payload				
Lane Width	16	128	256	4096
x1	0.5	1.7	1.7	2.0
x4	2.0	6.8	7.0	7.8
x8	4.0	13.7	14.0	15.7
x16	7.9	27.4	28.0	31.4

K.4 Configuring PCI Express with Other Applications

The following are examples of PCI Express core configurations for specific applications:

K.4.1 Configuring the PCI Express Core for Gigabit Ethernet Applications

- ❖ From the Gigabit Ethernet spec: 1000 Mbits/s = 125 MB/s (required)
- ❖ From the PCIe core, 125 MB/s is the minimum required throughput.

If you select a 125 MHz, 32-bit application interface, the bandwidth you can get is:

$$125 \text{ MHz / bits} \times 4 \text{ byte / s} = 500 \text{ MB/s} > 125 \text{ MB/s (required)}$$

So you can configure the PCI Express core with a 32-bit application interface and a 125 MHz clock with a 128-byte payload size operating in x1 at the PIPE interface.

Or,

- ❖ From the 10 Gigabit Ethernet spec: 10,000 Mbits/s = 1250 MB/s or for a sustained host bandwidth 1400 MB/s - 1800 MB/s (required)
- ❖ From the PCIe core, 1800 MB/s is the minimum required throughput.

If you select a 125 MHz, 128-bit application interface, the bandwidth you can get is:

$$125 \text{ MHz / bits} \times 16 \text{ byte / s} = 2000 \text{ MB/s} > 1800 \text{ MB/s (required)}$$

So you can configure the PCI Express core with a 128-bit application interface and a 125 MHz clock with a 256-byte payload size operating in x8 at the PIPE interface.

K.4.2 Configure the PCI Express core for SATA II Applications

- ❖ From the SATA, Gen2 spec: 3000 Mbits/s = 375 MB/s (required)
- ❖ From the PCIe core, 375 MB/s is the minimum required throughput.

If you select a 250 MHz, 32-bit application interface, the bandwidth you can get is:

$$250 \text{ MHz / bits} \times 4 \text{ byte / s} = 1000 \text{ MB/s} > 375 \text{ MB/s (required)}$$

So you can configure the PCI Express core with a 32 bit application interface and a 250 MHz clock with a 256-byte payload size operating at x4 at the PIPE interface.

Or,

- ❖ From the SATA, Gen2 spec: 3000M bits/s = 375 MB/s (required)
- ❖ From the PCIe core, 375 MB/s is the minimum required throughput.

If you select a 125 MHz, 64-bit application interface, the bandwidth you can get is:

$$125 \text{ MHz / bits} \times 8 \text{ byte / s} = 1000 \text{ MB/s} > 375 \text{ MB/s (Required)}$$

So you can configure the PCI Express core with a 64-bit application interface and a 125 MHz clock with a 256-byte payload size operating at x4 or x8 at the PIPE interface.

K.5 Summary

This application note explains how to meet your desired throughput by selecting the appropriate configuration parameters from those supplied with the DesignWare PCI Express core. This application note describes how throughput is calculated, and the factors that may affect the calculations. Refer to the DesignWare PCI Express core user and reference manuals for instructions on configuring the core using the supplied coreConsultant User Interface, and a detailed description of the selectable parameters.



App Note: How to Tie Off Unused Lanes

Not all lane widths supported by the core can be directly configured through coreConsultant. For example, to implement a x2 (or x1) core at 250 Mhz, you select the x4 core option from coreConsultant. In this case, the additional lanes must be tied off properly in order for the core to function. This application note describes the signal assignments that enable this functionality.

The following example applies to the 32-bit, 250 MHz core only, but the same technique may be applied to other core data path width and lane combinations.

The first important requirement is that Lane0 signals must always be active. When wiring up a Link to connect two devices, the lane number of each devices' port should match up such that Lane0 of one device's port connects to Lane0 of the remote device's port.

To activate only Lane0, please ensure that the following connections are made in the Lane0 PIPE interface signals. Similar connections should be followed for other lanes that are to be made active.

Lane0 Input Signals

1. phy_mac_rxdata_i[15:0] must be connected to RxData[15:0] of Lane0 of the PHY.
2. phy_mac_rxdatak[1:0] must be connected to RxDataK[1:0] of Lane0 of the PHY.
3. phy_mac_rxvalid[0] must be connected to Lane0 of the PHY.
4. phy_mac_rxstatus[2:0] must be connected to RxValid[2:0] of Lane0 of the PHY.
5. phy_mac_phystatus[0] must be connected to PhyStatus of Lane0 of the PHY.
6. phy_mac_rxelecidle[0] must be connected to RxElecIdle of Lane0 of the PHY.

Lane0 Output Signals

1. mac_phy_txdata[15:0] must be connected to TxData[15:0] Lane0 of the PHY.
2. mac_phy_txdatak[1:0] must be connected to TxDataK[1:0] Lane0 of the PHY.
3. mac_phy_txcompliance[0] must be connected to TxCompliance of Lane0 of the PHY.
4. mac_phy_txelecidle[0] must be connected to TxElecIdle of Lane0 of the PHY.
5. mac_phy_txdetetrx_loopback[0] must be connected to TxDetectRx/Loopback of the Lane0 of the PHY
6. mac_phy_powerdown[1:0] must be connected to PowerDown[1:0] of the Lane0 of the PHY.

In addition to connecting the above Lane0 signals, the following signals must be connected for Lane1, Lane2, and Lane3 (or any lane to be made inactive).

1. phy_mac_rxstatus[11:3] must be driven to 0.
2. phy_mac_phystatus[3:1] must each be connected to the Lane 0 PhyStatus[0].
3. phy_mac_rxlecidle[3:1] must be driven to 0.
4. phy_mac_rxvalid[3:1] must be driven to 0.
5. All other inputs signals should be driven to 0.
6. All other output signals may be left unconnected.

During Link Initialization and training, the for Lane1, Lane2, and Lane3 will be recognized as not having a receiver termination, and therefore, these lanes will be turned off and the negotiated link width will be x1.